SWARM ROBOTICS WORKSHOP

Dr. Andrew Vardy
Bio-inspired Robotics (BOTS) Lab
Department of Computer Science
Department of Electrical and Computer Engineering
Memorial University of Newfoundland
St. John's, Canada

WHERE I'M FROM

SWARM ROBOTICS?

- "Swarm robotics is the study of how a large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among the agents and between the agents and the environment."
  - [Şahin, E. (2004). Swarm robotics: From sources of inspiration to domains of application. In International workshop on swarm robotics (pp. 10-20). Springer, Berlin, Heidelberg.]

- "Swarm robotics is the study of how a large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among the agents and between the agents and the environment."

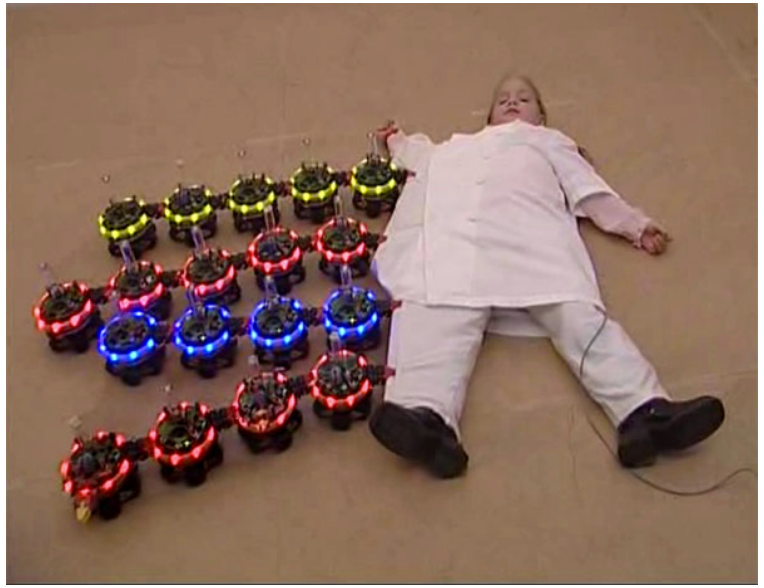
- "Swarm robotics is the study of how a large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among the agents and between the agents and the environment."

- "Swarm robotics is the study of how a large number of relatively simple ~~physically embodied agents~~ **robots** can be designed such that a desired collective behavior emerges from the local interactions among the agents and between the agents and the environment."

Kilobots: 1024

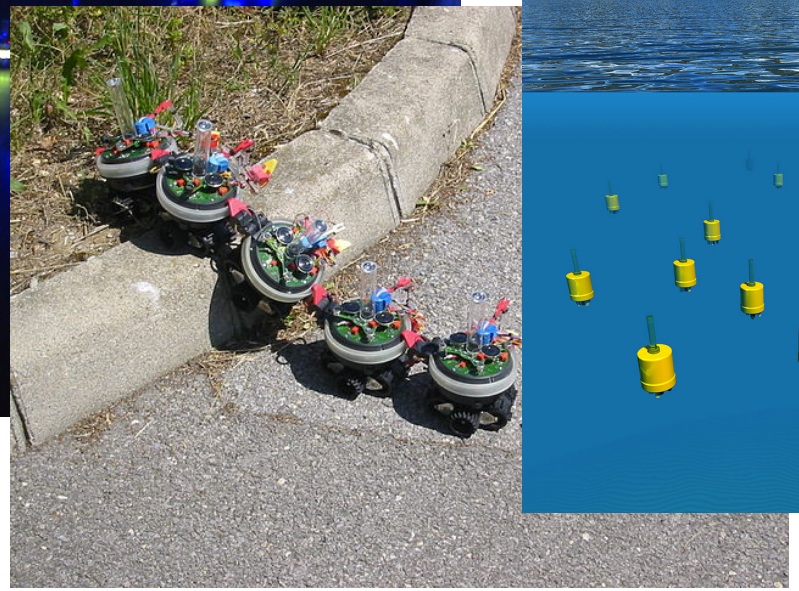
Many limited robots

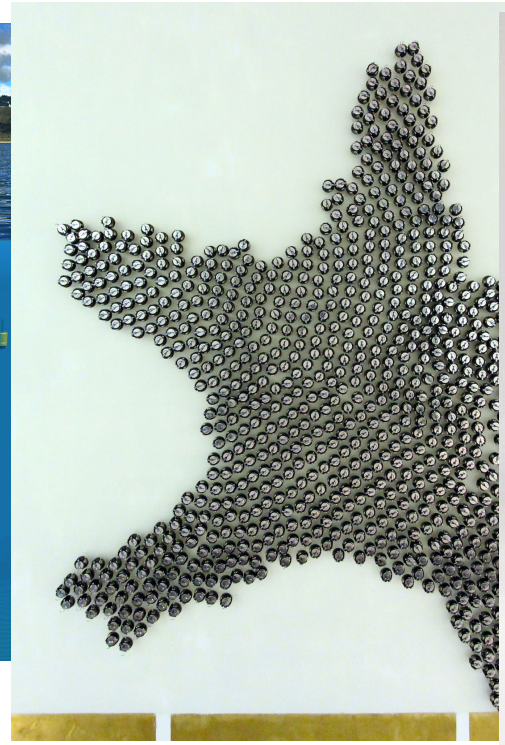
S-bots: 10-20



Fewer, more capable robots

- "Swarm robotics is the study of how a large number of relatively simple ~~physically embodied agents~~ **robots** can be designed such that a desired collective behavior emerges from the local interactions among the agents and between the agents and the environment."

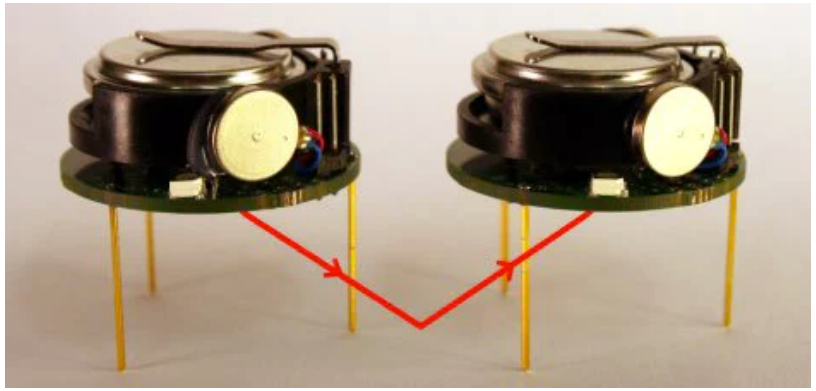


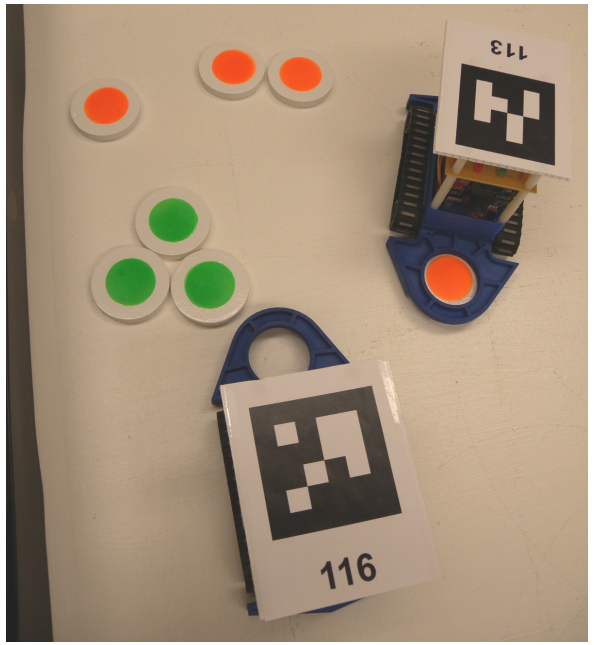


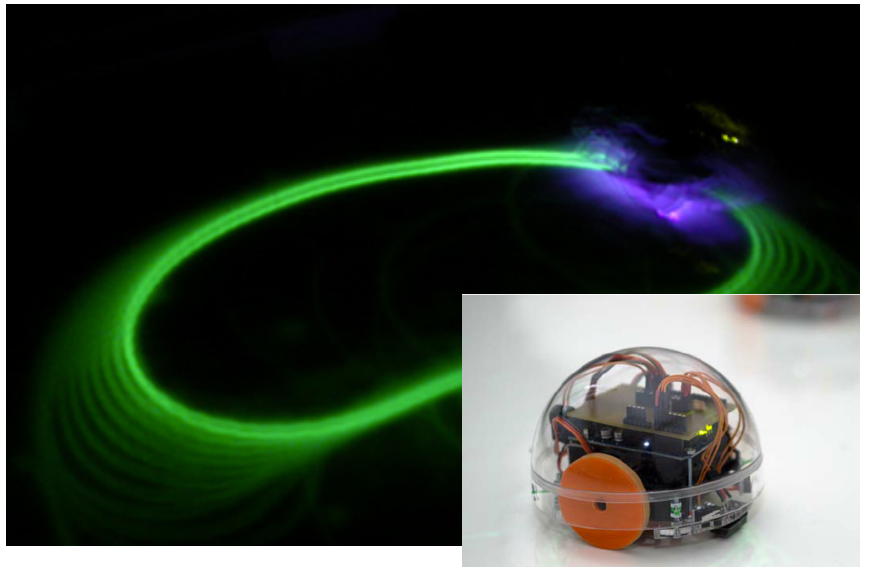
- "Swarm robotics is the study of how a large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among the agents and between the agents and the environment."

Indirect communication (perceive results of others actions)

Direct communication (local)







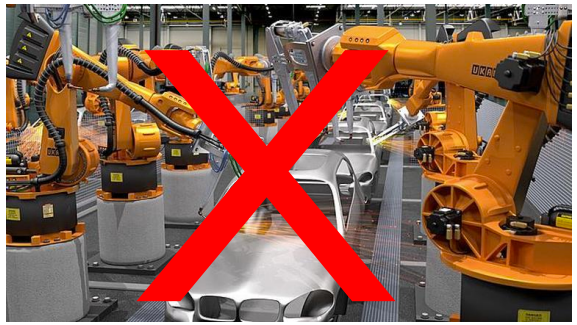
- "Swarm robotics is the study of how a large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among the agents and between the agents and the environment."

CHARACTERISTICS OF SWARM ROBOTICS

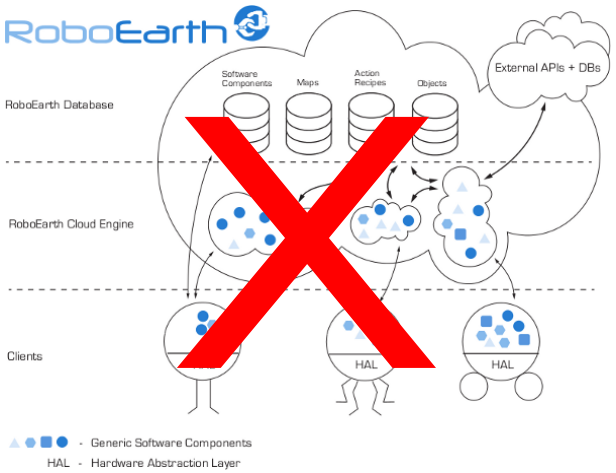
- Swarm Robotics: A multi-robot system with the following characteristics:
  - robots are **autonomous**;
  - robots are **situated** in the environment and can act to modify it
  - robots' **sensing and communication capabilities are local**
  - robots **do not have access to centralized control** and/or to global knowledge
  - robots **cooperate** to tackle a given task

- Brambilla, M., Ferrante, E., Birattari, M., & Dorigo, M. (2013). Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1), 1-41.

- robots are **autonomous**;



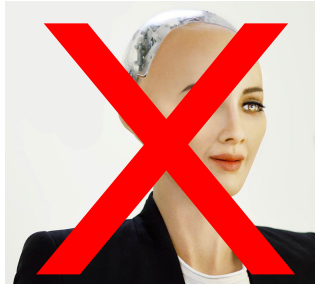
- robots are **situated** in the environment and can act to modify it



- robots' **sensing and communication capabilities are local**

- robots **do not have access to centralized control** and/or to global knowledge

- robots **cooperate** to tackle a given task



- Definition:
  - "Swarm robotics is the study of how a large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among the agents and between the agents and the environment."

- Characteristics:
  - robots are **autonomous**;
  - robots are **situated** in the environment and can act to modify it
  - robots' **sensing and communication capabilities are local**
  - robots **do not have access to centralized control** and/or to global knowledge
  - robots **cooperate** to tackle a given task

What robots exist that adhere to this definition and exhibit these characteristics?

NATURAL ROBOTS: THE HONEYBEES

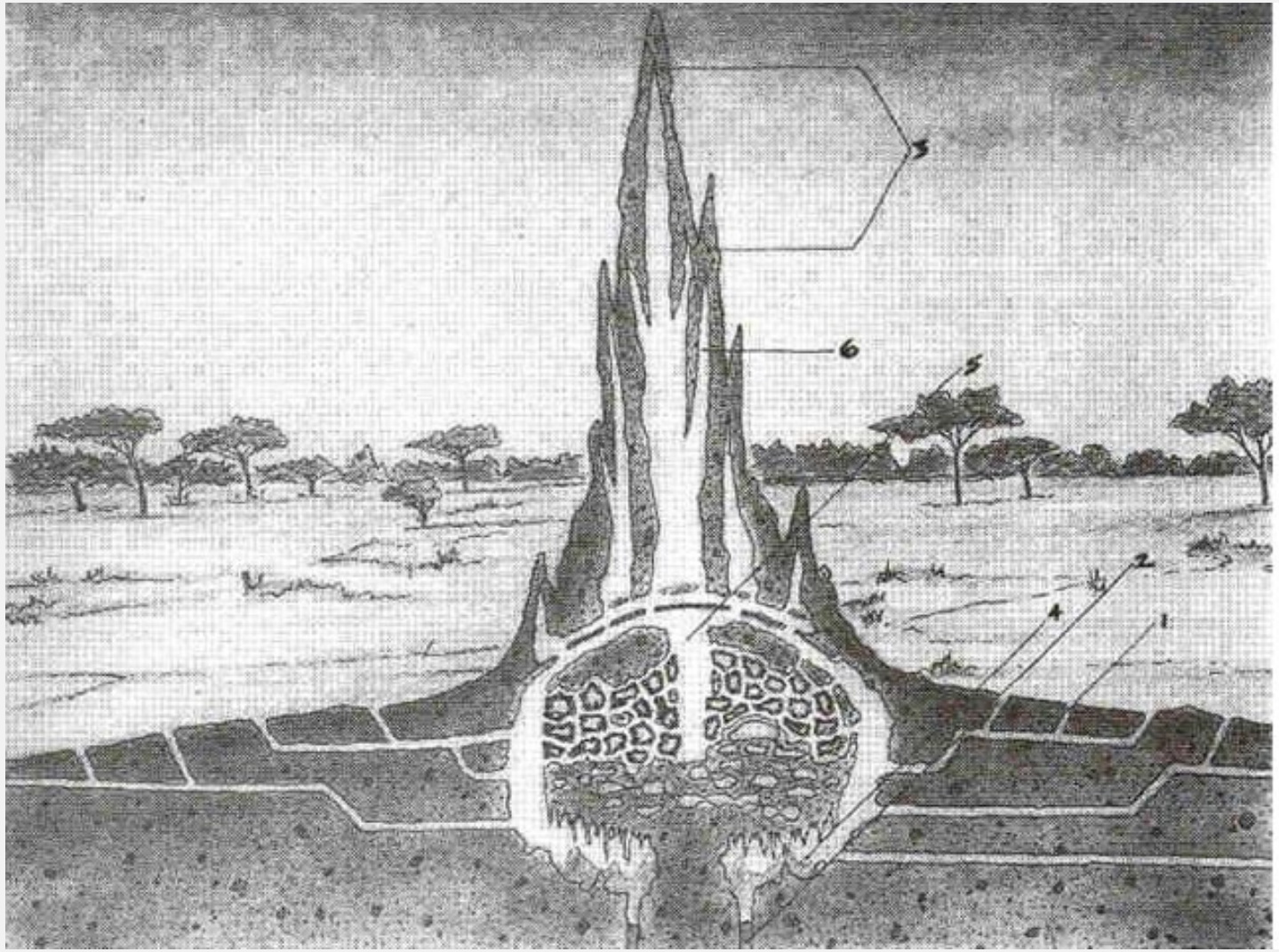
- Democratic nest selection
  - Seeley, T. D. (2010). *Honeybee democracy*. Princeton University Press.
- Waggle dance communication

NATURAL ROBOTS: THE ANTS

- Extremely successful family
  - 15 - 20% of terrestrial biomass
- Leafcutter ants
  - Invented agriculture millions of years before us!

NATURAL ROBOTS: THE TERMITES

- Termite mounds taller than a computer scientist
  - Construct extremely sophisticated structures…



1. night entrance and exit;
2. underground water supply for drinking and cooling nest;
3. "lungs" that expel rising hot air;
4. Cool air eventually sinks back to the cellar;
5. Warm air rises via central air duct;
6. Interior oxygen diffuses through the chimneys.

Sophisticated architecture and engineering from insects with small brains, poor vision, and no central coordination

SELF-ORGANIZATION

- "Self-organization is a set of dynamical mechanisms whereby structures appear at the global level of a system from interactions among its lower-level components."
  - Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). Swarm intelligence: from natural to artificial systems
- Individual ants, bees, termites, and robots interact locally… yet a global pattern emerges
  - **THE CENTRAL MYSTERY!**

OUR TOOLS:

BLOCKLY AND WAGGLE

WAGGLE

- We will be using Waggle, an online simulation and programming environment
  - 2-D Physics: Matter.js
  - Visual programming: Blockly
- There are other tools for simulating swarms of robots:
  - Agent-based simulators: Netlogo, MASON
  - Robot simulators: ARGoS, V-REP, Gazebo
- Why another simulation tool?


Realistic
Gazebo
V-REP
ARGoS
Waggle
MASON
Netlogo
Abstract
Flexible
Easy to Use

BLOCKLY

- A library for building visual programming apps
  - Started by Google, now open-source
  - Used for hundreds of educational apps on the web and on Android / iOS devices
- Programming for kids?
  - Yes
  - ”But we’re not kids”
  - Blockly is an easily learned language for people with various levels of programming experience

EXERCISE #1: BLOCKLY GAMES 20 MINUTES

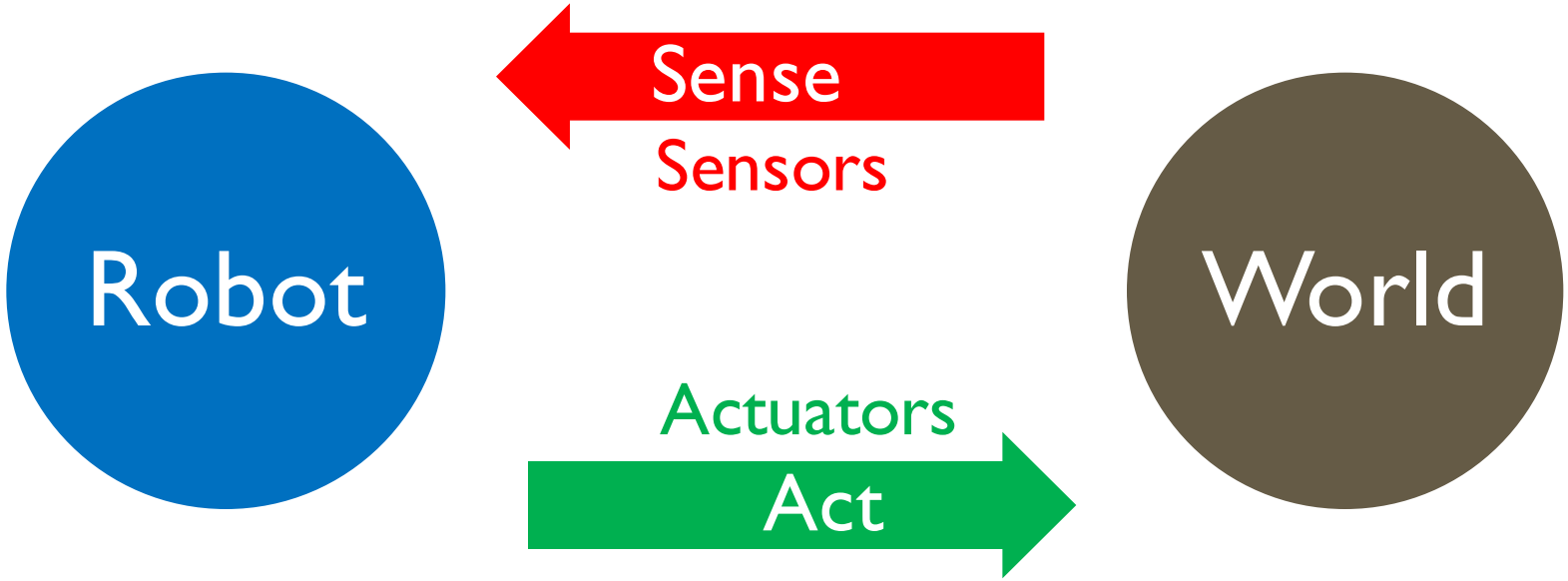
- Go to Blockly Games:
  - https://blockly-games.appspot.com/
- Complete "Puzzle"
- Complete as many levels as you can of "Maze"

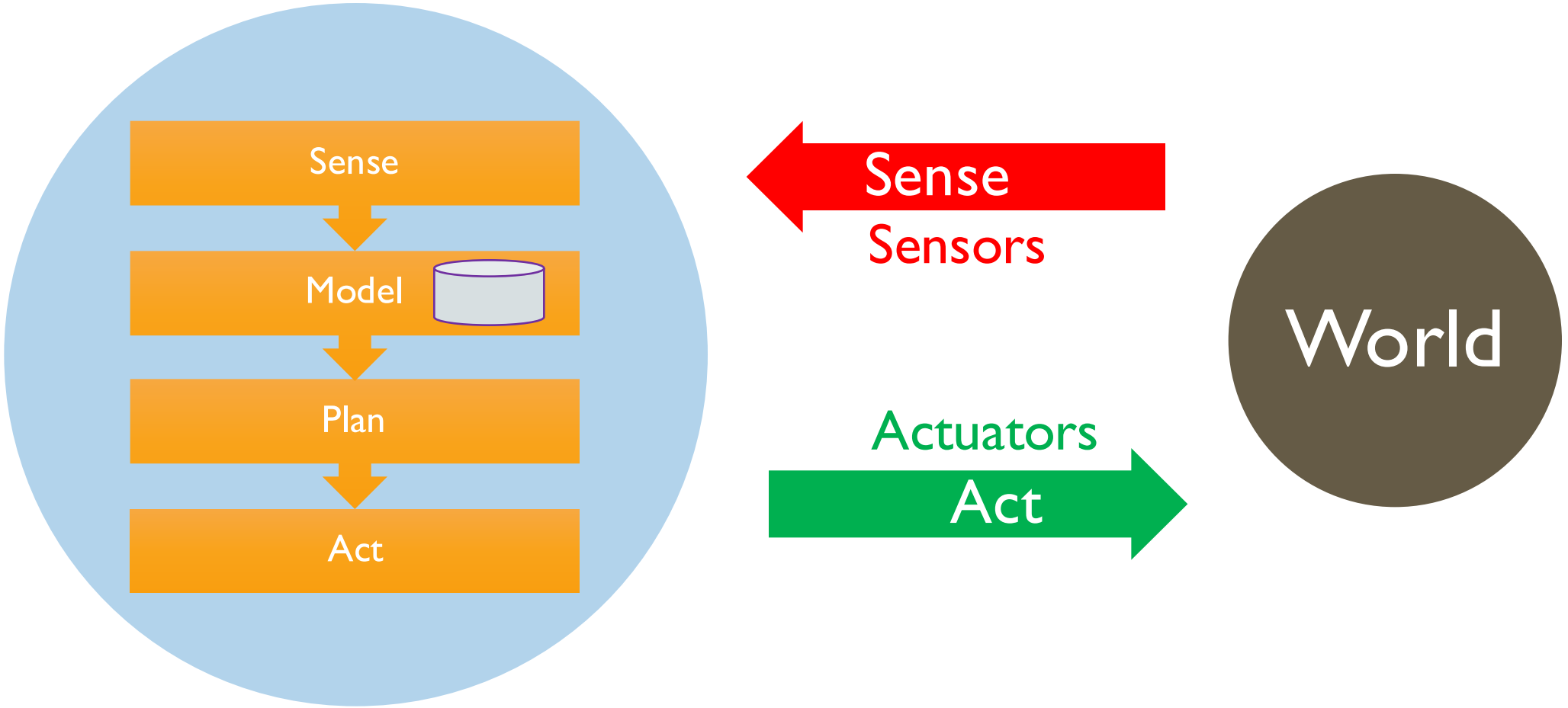
- Experienced programmers:
  - There are interesting challenges as you progress into the higher levels

ROBOTICS CONCEPTS

THE SENSE / ACT CYCLE

- Robots exist in a constant cycle of perception (via sensors) and movement (via actuators)

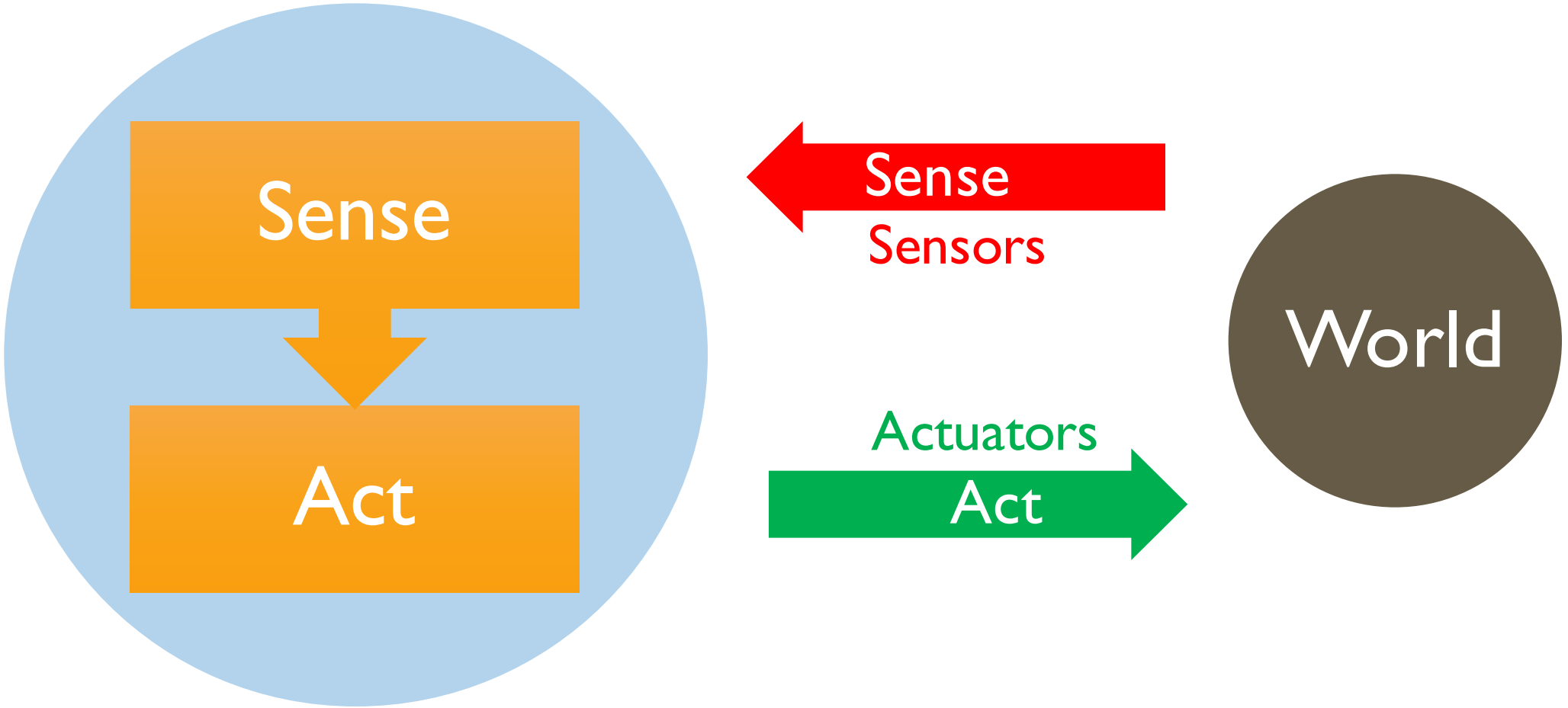




- The robot's controller may be very **deliberative**
  - **Carefully build a model of the world, plan a trajectory, then execute it**

- Or the controller may be **reactive**
  - **There is a direct mapping of sensor states to actions**
  - **The model (if any) is small**

We will be using reactive controllers



TUTORIAL LEVEL

THE WAGGLE PAGE

- Go to the main Waggle page:
  - http://bots.cs.mun.ca/waggle/
  - Note that these slides are available here
- Select the **Tutorial** level
- Take note of the following features…

Simulation area

Robot (try click and drag)

Reset

Allow Movement

Show Sensors

Timescale 1

Desired Population 1

Actual Population 1

Clear

Load

Choose File No file chosen

Save As

blocks.xml

Show Javascript

Show XML

Status: A-OK: Please design your controller below

Sensors

Actions

Memory

Logic

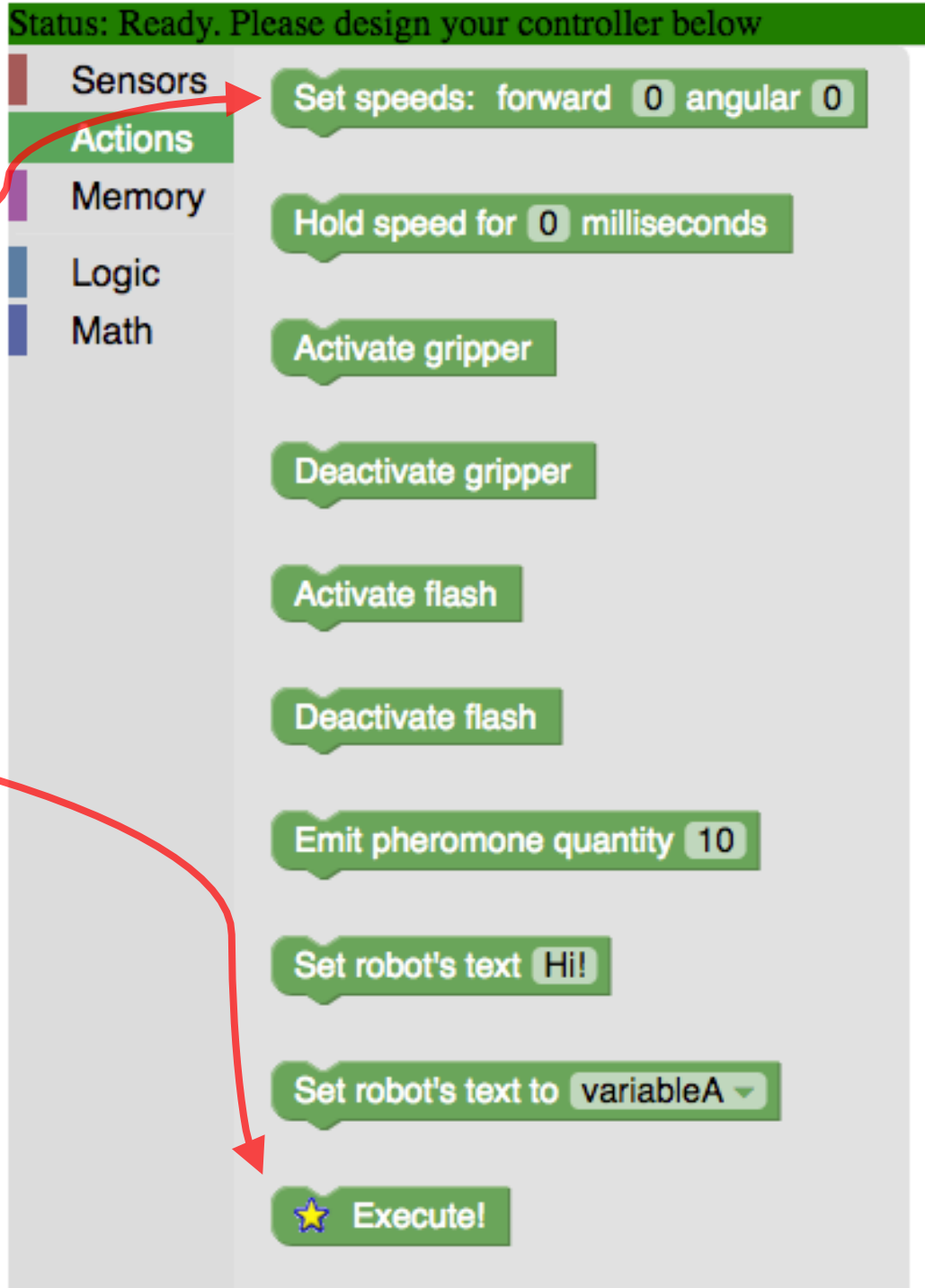
Math

} Robot-specific Blockly categories

Blockly area

- Within the Blockly area, click on the "Actions" category
- Now click-and-drag on "Set speeds" and pull it into the open white area
- Click on the '0' next to "forward" and change it to some number in the range [-10, 10]
  - Robot does not move
- Click-and-drag "Execute" beneath "Set speeds" so that they connect like this:

  - Now the robot moves!



OUR FIRST CONTROLLER!

- So this is our first controller:

- A controller senses the world, then acts
  - But this controller is so basic, it doesn't even do any sensing
- The controller is executed by the simulator many times per second
  - There is no need for a "loop", the controller gets called over-and-over by default

OUR FIRST CONTROLLER!

- So this is our first controller:

- A controller senses the world, then acts
  - But this controller is so basic, it doesn't even do any sensing
- The controller is executed by the simulator many times per second
  - There is no need for a "loop", the controller gets called over-and-over by default

DEFERRED EXECUTION

- Notice how the robot did not move without "Execute"
  - The controller uses a **deferred execution** model, meaning that nothing happens until "Execute". The controller is executed top-to-bottom and each block in the "Actions" category sets something about the action the robot will take. However, no action is taken until "Execute".
- e.g. The following controllers all behave the same because the speeds set by the blocks on the bottom overwrite the speeds set above:

EXERCISE #2:

BASIC MOVEMENT

5 MINUTES

- Experiment with "Set speeds" to derive controllers that travel in...
  - Straight lines
  - Circles
- Note the importance of 

- Experiment with positive / negative speeds
  - Try the following:
    - Counter-clockwise circle, moving forwards
    - Counter-clockwise circle, moving backwards

EXERCISE #2: REVIEW

- Move forward (speed 10):

- Turn clockwise (speed 10):

  - Set angular speed negative for counter-clockwise turns

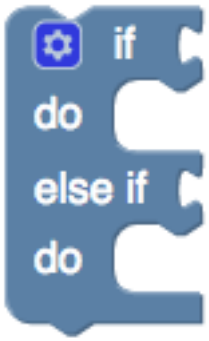
- Move in a tight clockwise circle:

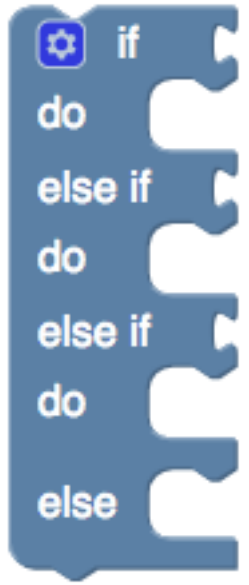
  - Reduce angular speed for a larger circle; Reduce forward speed for a smaller circle
- Use negative values for backward motion and counter-clockwise turns

THE CONDITIONAL BLOCK

- Under "Logic" select the conditional block:

- Connected to the top part will be a condition block which must evaluate to True or False
  - If the condition block is True, the block(s) within the "do" will execute
- Click on the star to customize this block. Here are some possibilities:

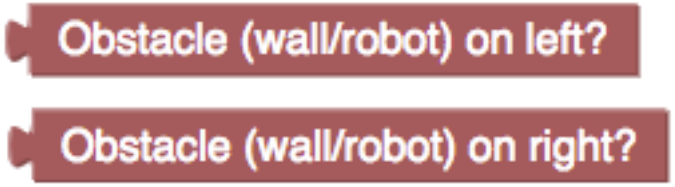




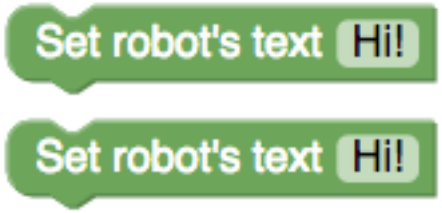
CONDITIONAL + SENSORS

- Create an if – else if – else conditional block:

- Select the "Sensors" category and drag in these two blocks:

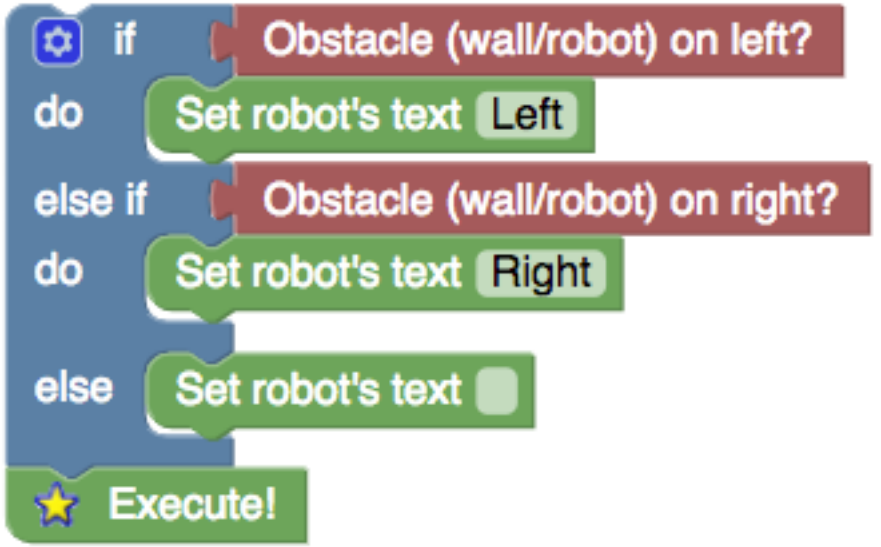


- Now under the "Actions" category drag in two of these:



- Also under "Actions" drag in an

- Connect the blocks together like this:



- Note that you have to customize the text for each "Set robot's text" block
- Drag the robot around, bumping into the walls to test it

EXERCISE #3:

OBSTACLE AVOIDANCE

5 MINUTES

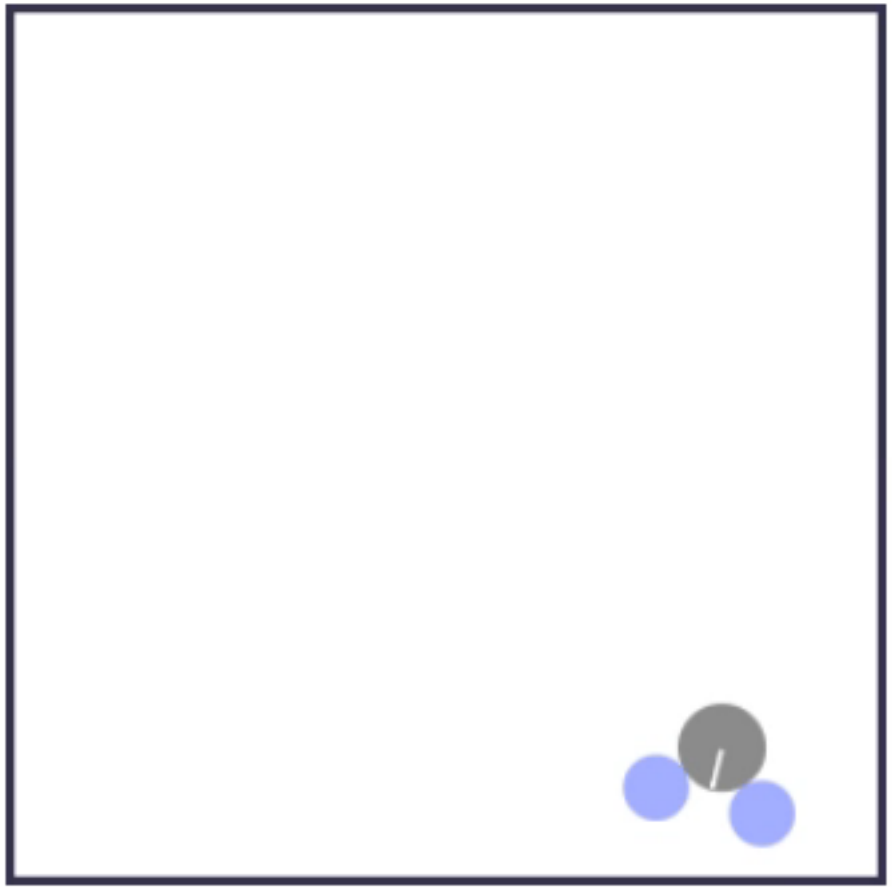
Conditions:

Should work with any number of robots

- Replace the 

blocks with 

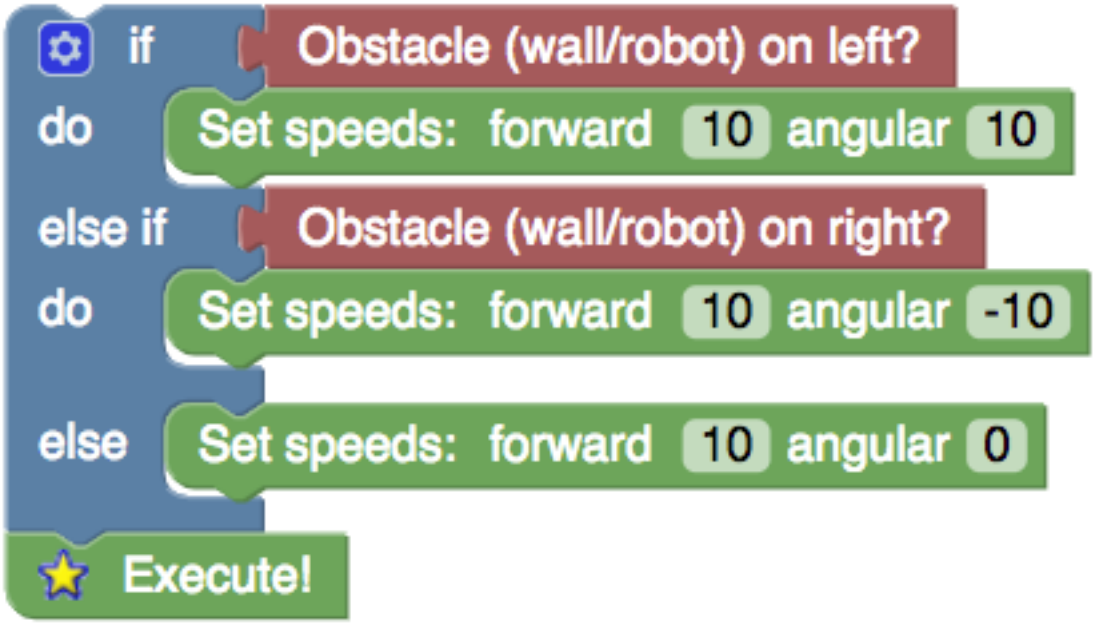
blocks and adjust the speeds to achieve this:



- Note: Should still work when bouncing against walls on the right

EXERCISE #3: REVIEW

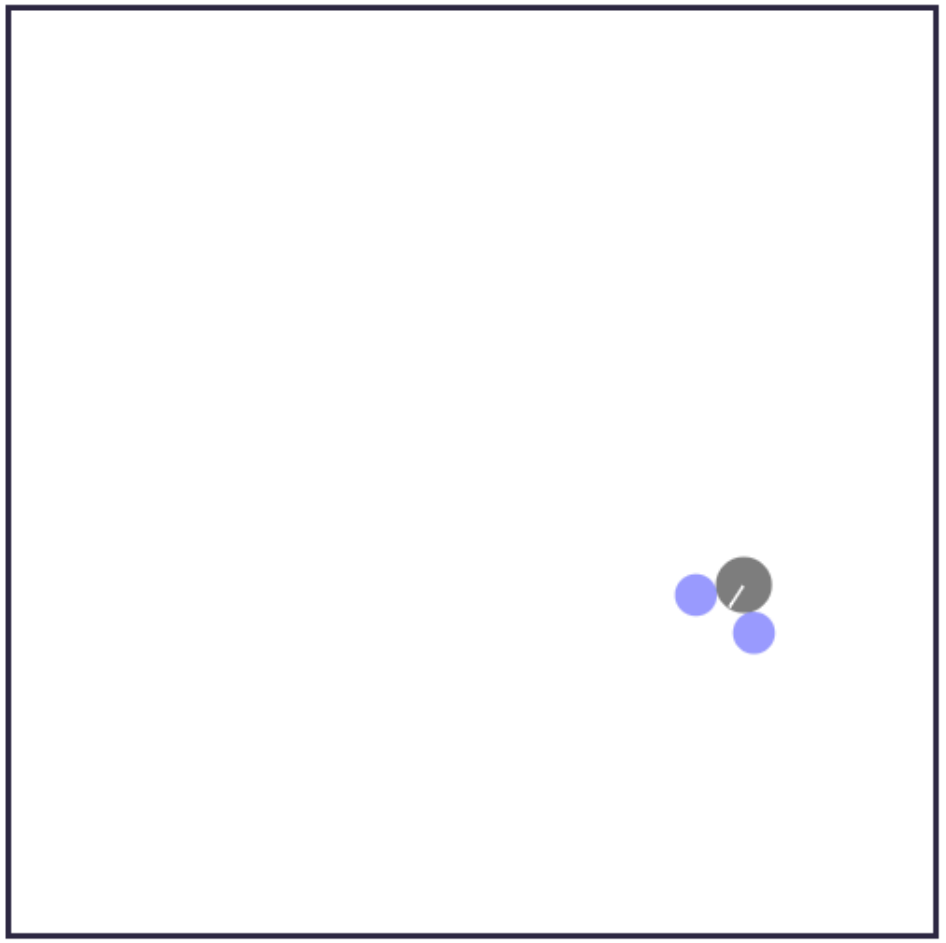
- Your solution should look something like this:
- You could reduce the speeds a little, but the robot might hit the wall (not a big problem)



WAGGLE: TOUR OF FEATURES AND BLOCKS

WAGGLE FEATURE / BLOCK TOUR

- Continuing on the Tutorial level, we will introduce some further features of the waggle page...



Clear | Load | Choose File | avoid_obstacles.xml | Save As | blocks.xml | Show Javascript | Show XML

Status: Good

- Sensors
- Actions
- Memory
- Logic
- Math

```
if   Obstacle (wall/robot) on left?
do   Set speeds: forward 10 angular 10
else if   Obstacle (wall/robot) on right?
do   Set speeds: forward 10 angular -10
else   Set speeds: forward 10 angular 0
Execute!
```

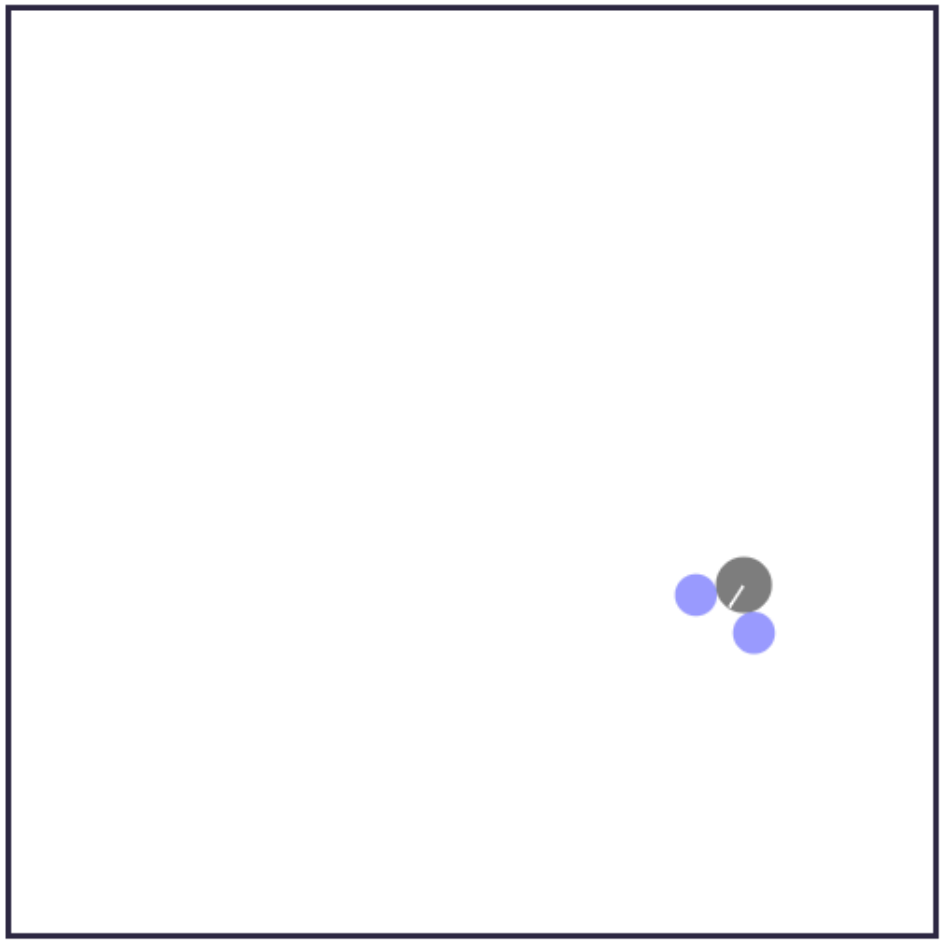
Reset | Allow Movement | Show Sensors

Timescale 1

Desired Population 1

Actual Population 1

Try adding more robots with this slider



Clear | Load | Choose File | avoid_obstacles.xml | Save As | blocks.xml | Show Javascript | Show XML

Status: Good

- Sensors
- Actions
- Memory
- Logic
- Math

Reset | Allow Movement | Show Sensors

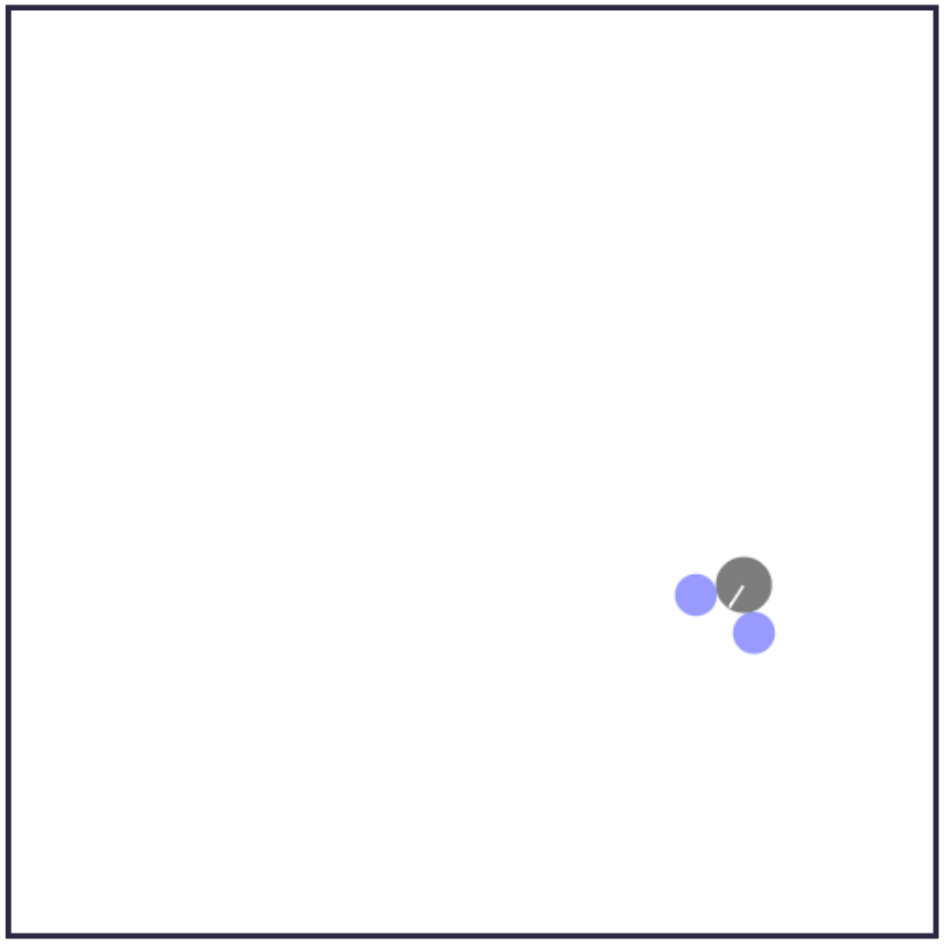
Timescale 1

Desired Population 1

Actual Population 1

Try adjusting this slider to see the simulation run in slow-motion

NOTE: Adjusting "Timescale" breaks up movement into smaller increments; This changes the sim's behaviour and is therefore not exactly the same as "slowing time".



Reset | Allow Movement | Show Sensors

Timescale 1

Desired Population 1

Actual Population 1

Clear | Load | Choose File | avoid_obstacles.xml | Save As | blocks.xml | Show Javascript | Show XML

Status: Good

- Sensors
- Actions
- Memory
- Logic
- Math

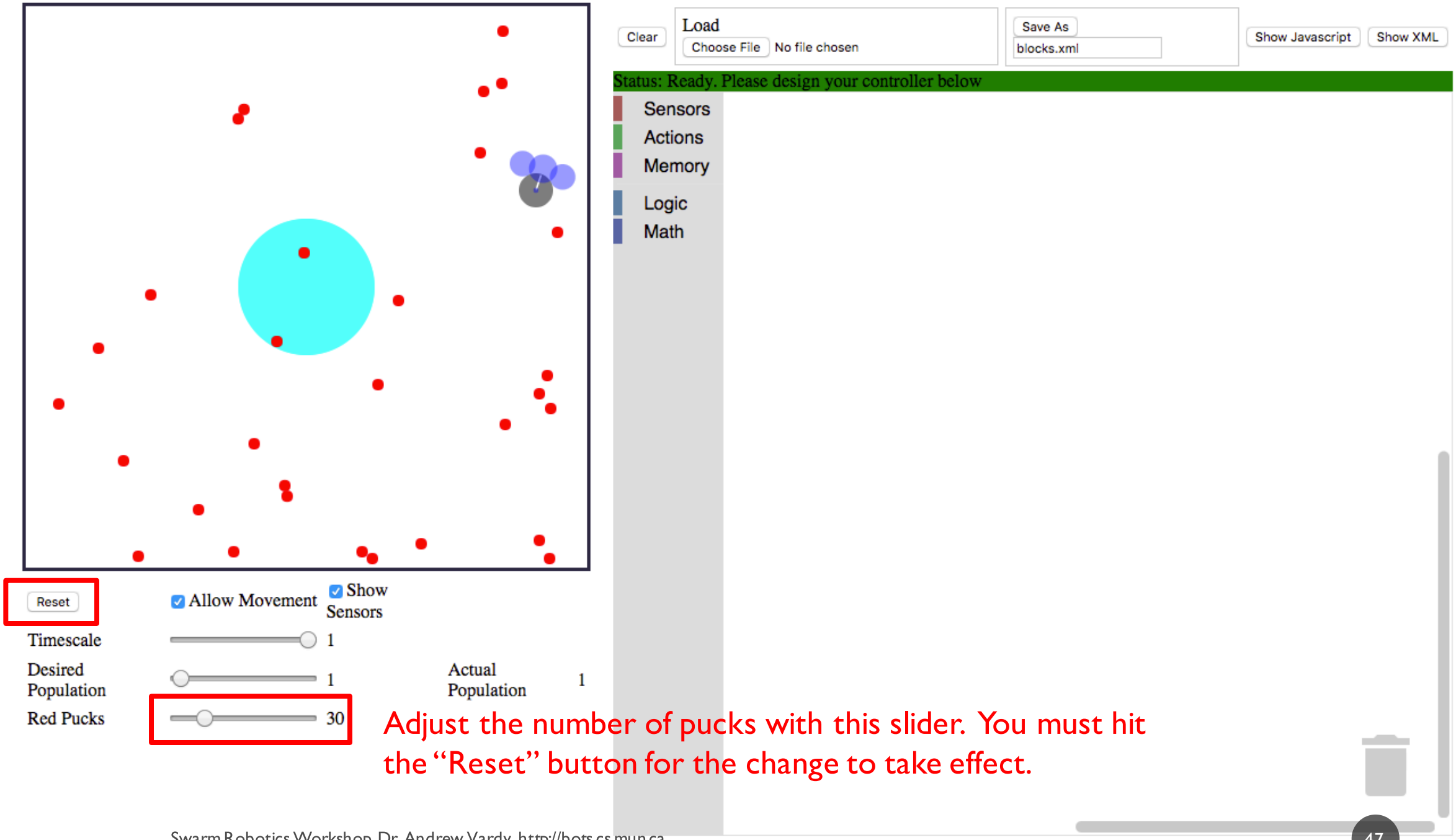
if Obstacle (wall/robot) on left?
do Set speeds: forward 10 angular 10
else if Obstacle (wall/robot) on right?
do Set speeds: forward 10 angular -10
else Set speeds: forward 10 angular 0
Execute!

Use these buttons to save and load your controller. When saving, the file saved will be set in this box and it will always be saved into your Downloads folder.

Save your controllers throughout the workshop and give them sensible names. If you hit the browser's refresh button, the current controller will be lost!

WAGGLE FEATURE / BLOCK TOUR

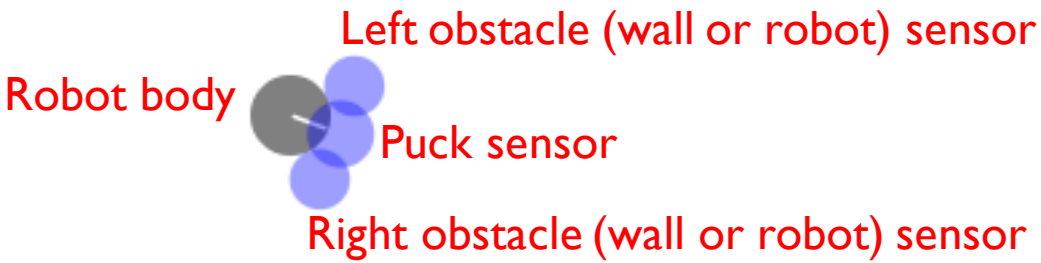
- Our next major topic is **Object Clustering**
- Prior to introducing the swarm robotics perspective on this, we will take a tour through the features and blocks needed for clustering
- Go to the main Waggle page (or just hit your browser's back button):
  - http://bots.cs.mun.ca/waggle/
- Select the **Pre-clustering** level



Adjust the number of pucks with this slider. You must hit the "Reset" button for the change to take effect.

SENSORS

- Sensors are attached to the robot in different configurations; Here is the configuration we will focus on for now:



- Note that the obstacle sensors cannot sense pucks and the puck sensor cannot sense obstacles
- When a sensor detects its target object, the sensor is shaded red, otherwise it remains blue

THE GRIPPER

- The robot can grasp a sensed puck by creating a virtual spring between itself and the puck

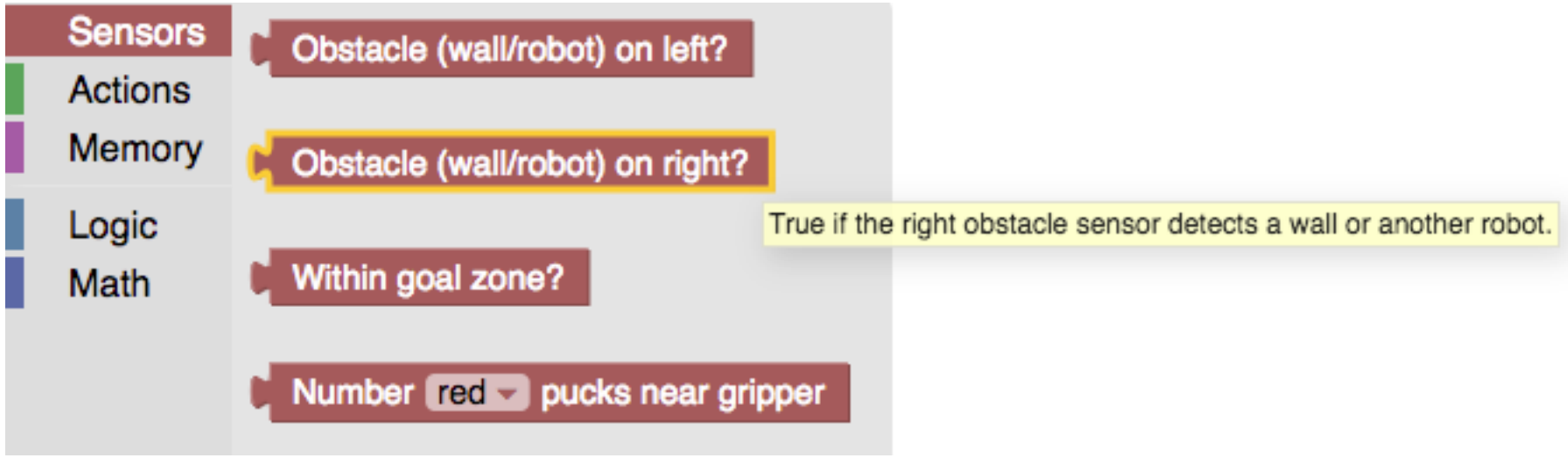
- Note that the sensor does not detect a grasped puck

DEMO: PUCK COUNT

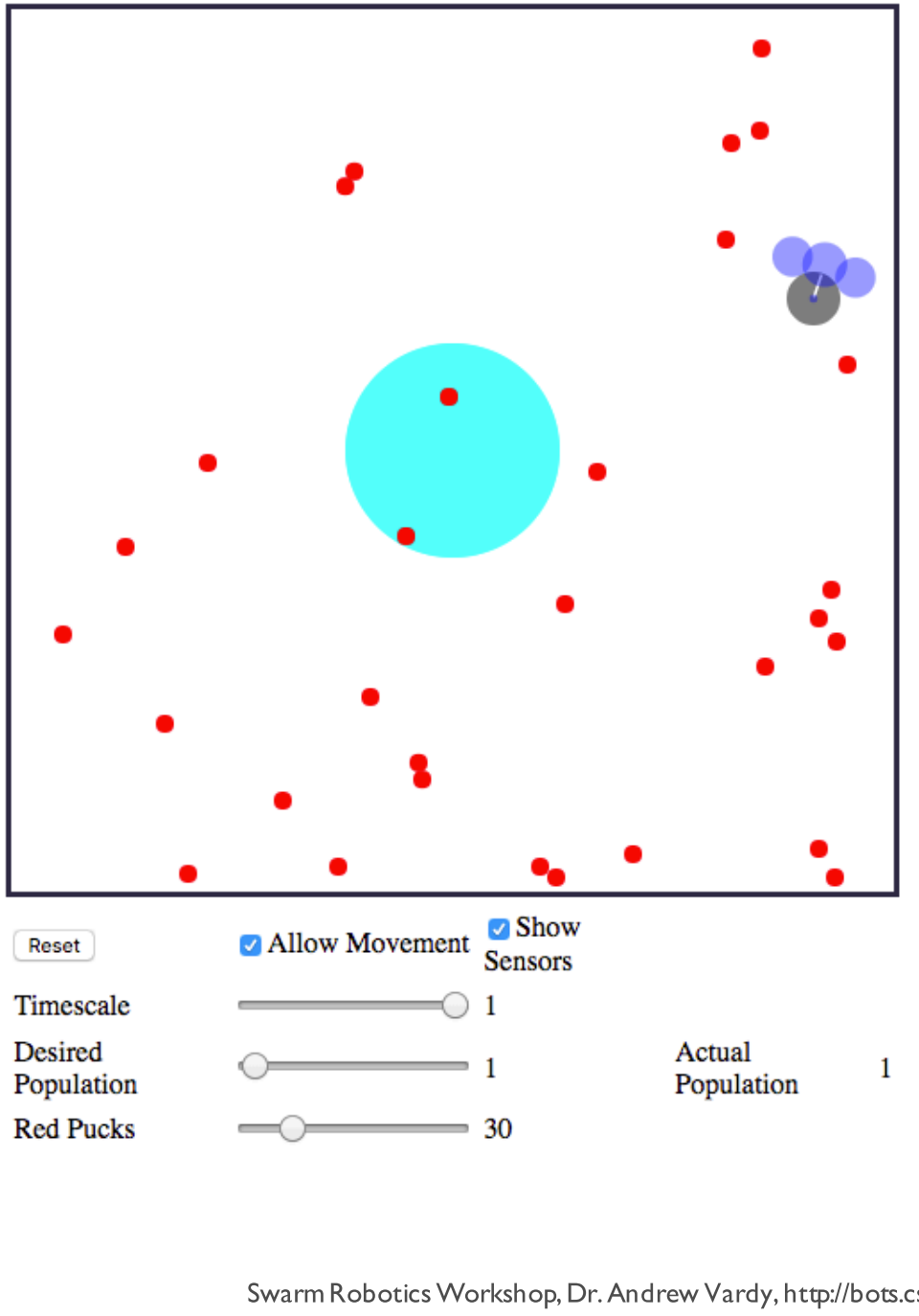
- The main Waggle page has demo controllers that you can load:
  - http://bots.cs.mun.ca/waggle/
- Download `grip_and_count.xml`
- Now go into the Pre-clustering level, click on "Choose File" and load it
- Drag the robot around; As soon as it comes close to a puck it will grip it
- Notice how it counts (at least up to 4) the number of pucks within sensor's radius, but not including the puck held

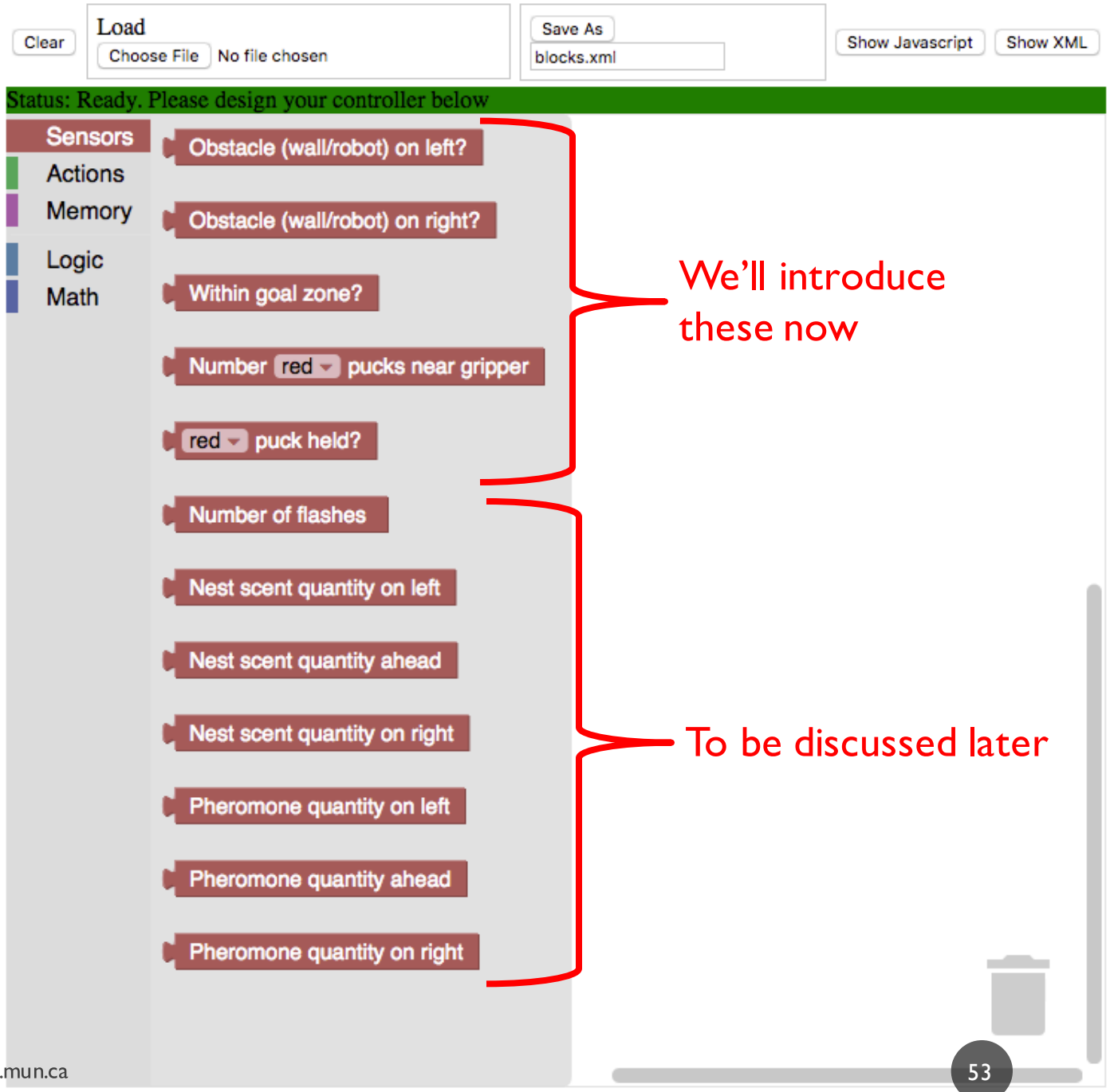
BLOCKLY CATEGORIES

- We will now tour through the various blocks available
- You can always get a helpful pop-up on any block by hovering your mouse above it:



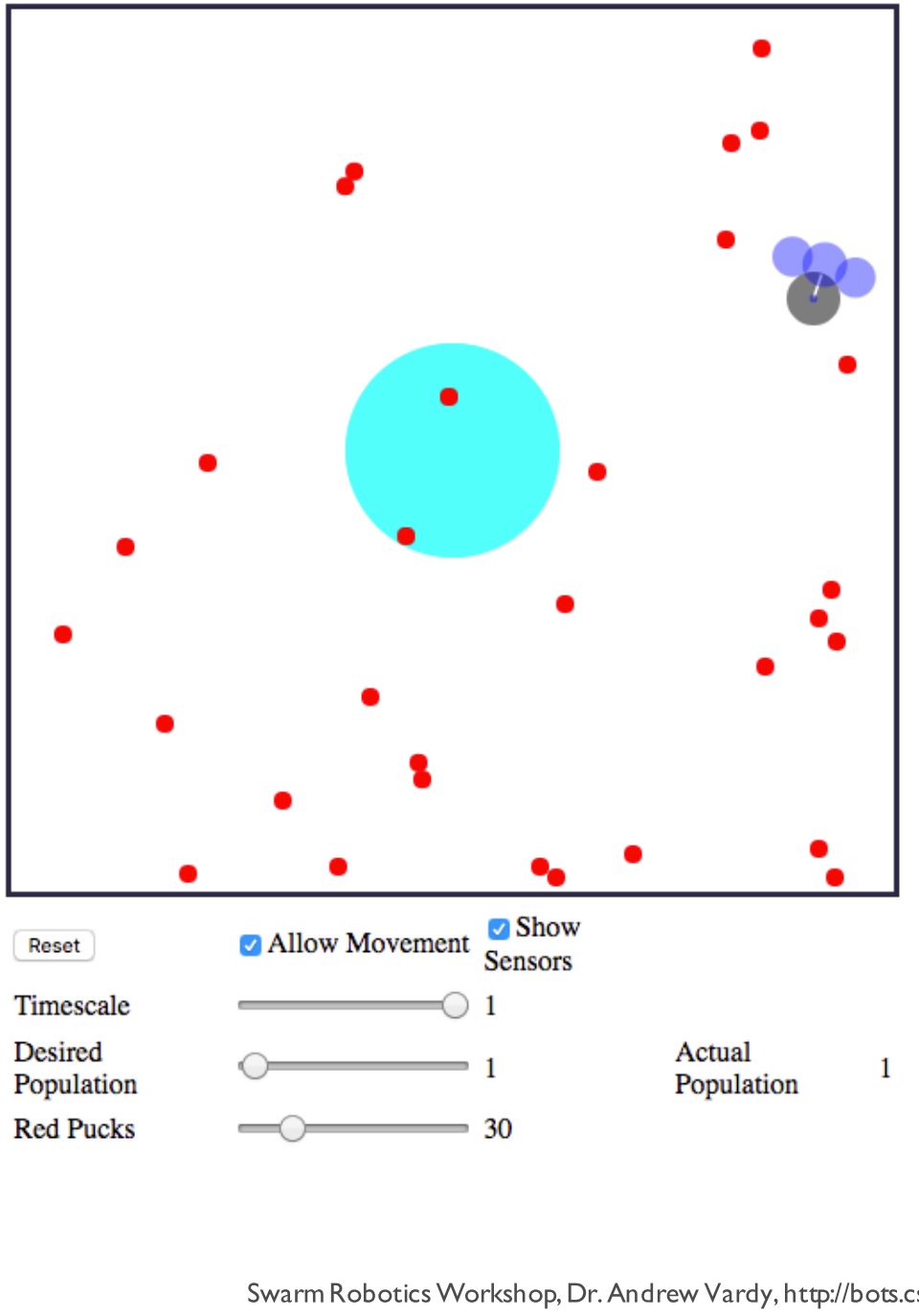
“SENSORS” CATEGORY

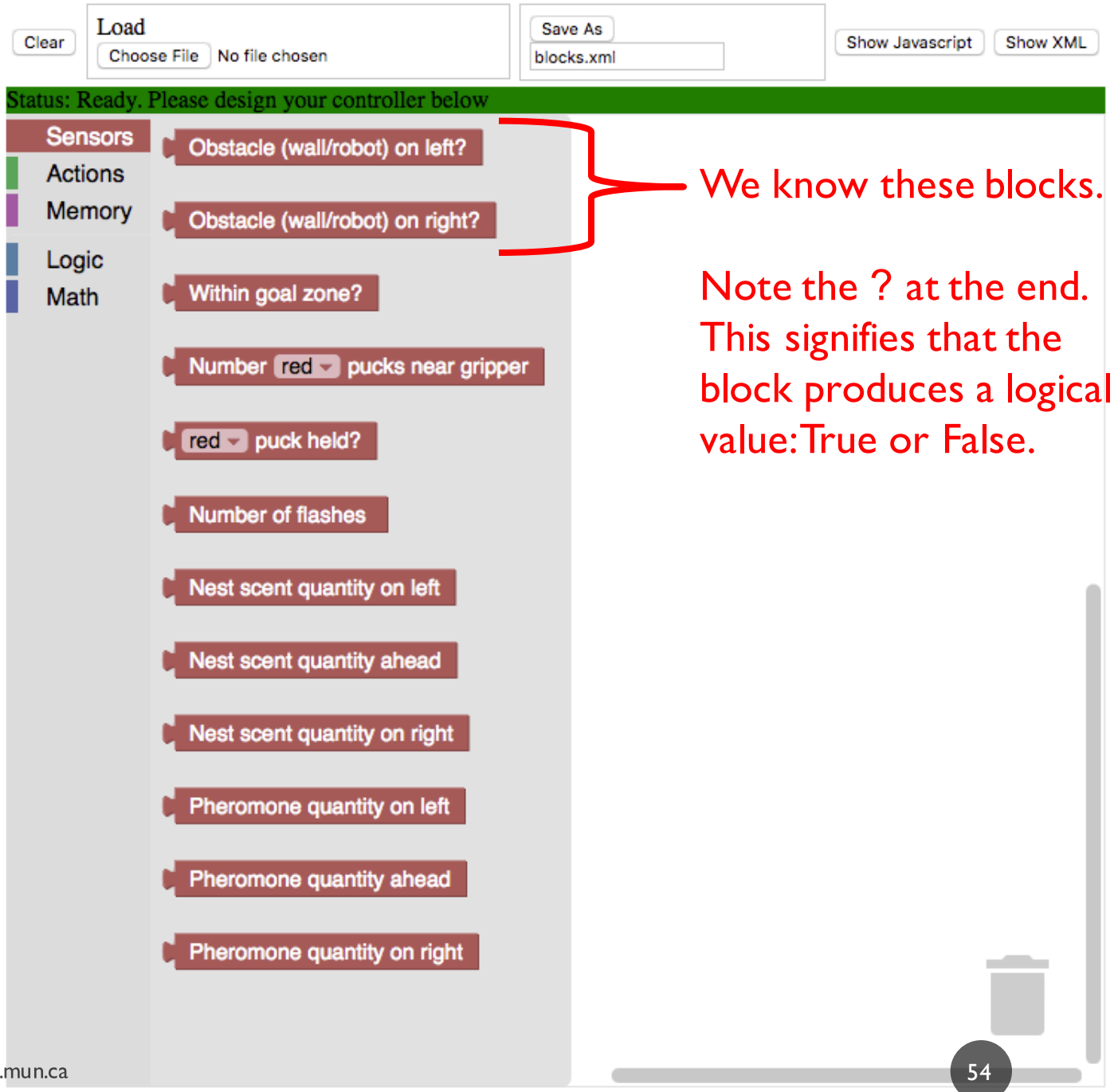




We'll introduce these now

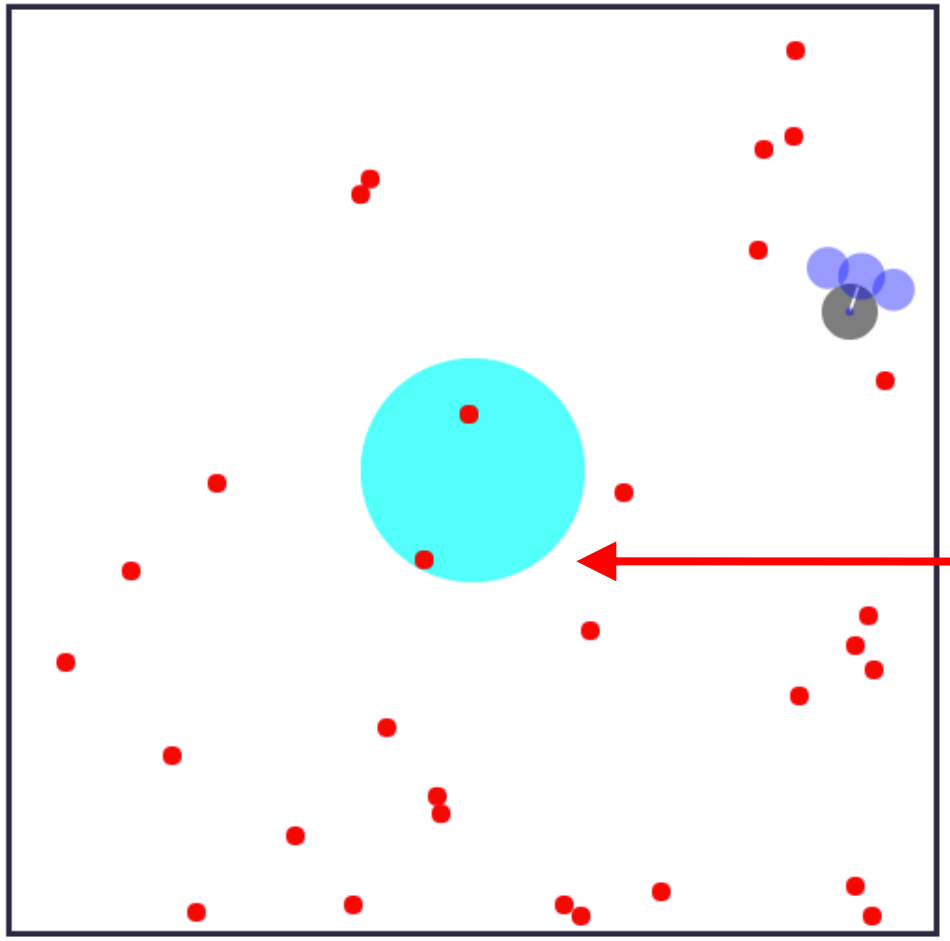
To be discussed later





We know these blocks.

Note the ? at the end. This signifies that the block produces a logical value: True or False.



Reset | Allow Movement | Show Sensors

Timescale 1

Desired Population 1 | Actual Population 1

Red Pucks 30

Clear | Load | Choose File | No file chosen | Save As | blocks.xml | Show Javascript | Show XML

Status: Ready. Please design your controller below

- Sensors
- Actions
- Memory
- Logic
- Math

Obstacle (wall/robot) on left?

Obstacle (wall/robot) on right?

Within goal zone?

Number red pucks near gripper

red puck held?

Number of flashes

Nest scent quantity on left

Nest scent quantity ahead

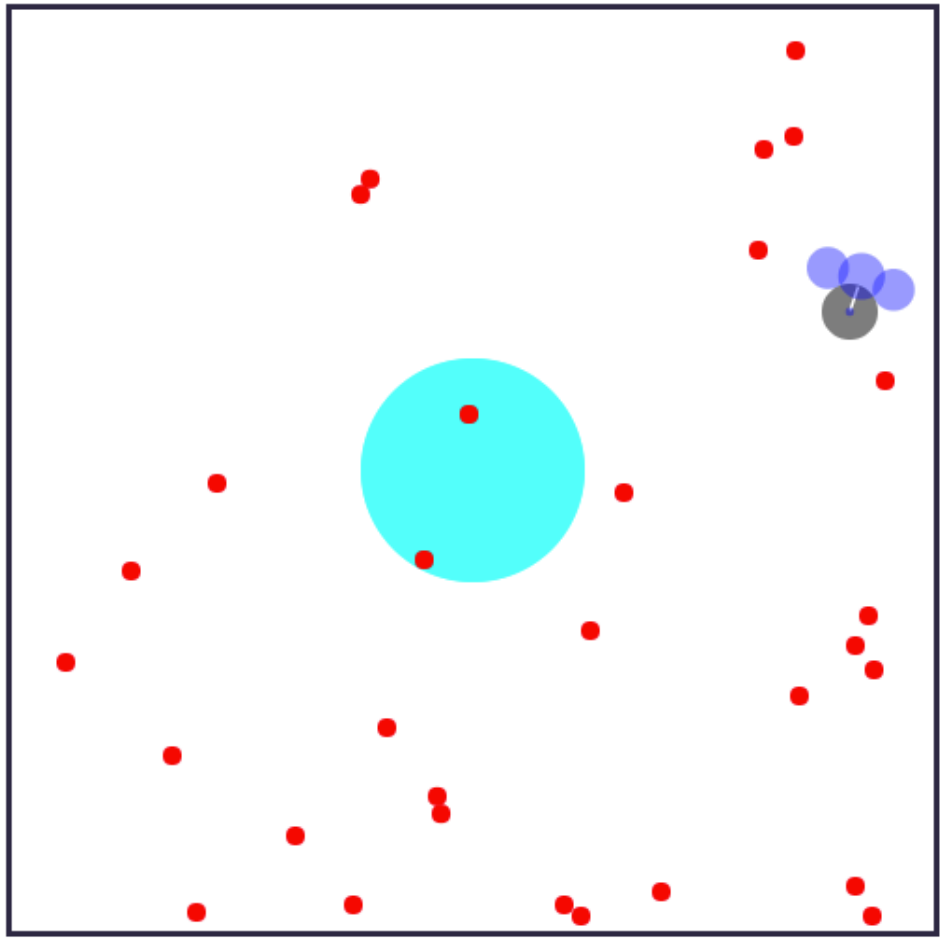
Nest scent quantity on right

Pheromone quantity on left

Pheromone quantity ahead

Pheromone quantity on right

Indicates whether or not the centre of the robot is within the **goal zone**.



Reset | Allow Movement | Show Sensors

Timescale 1

Desired Population 1

Actual Population 1

Red Pucks 30

Clear | Load | Choose File | No file chosen | Save As | blocks.xml | Show Javascript | Show XML

Status: Ready. Please design your controller below

- Sensors
- Actions
- Memory
- Logic
- Math

Obstacle (wall/robot) on left?

Obstacle (wall/robot) on right?

Within goal zone?

Number red pucks near gripper

red puck held?

Number of flashes

Nest scent quantity on left

Nest scent quantity ahead

Nest scent quantity on right

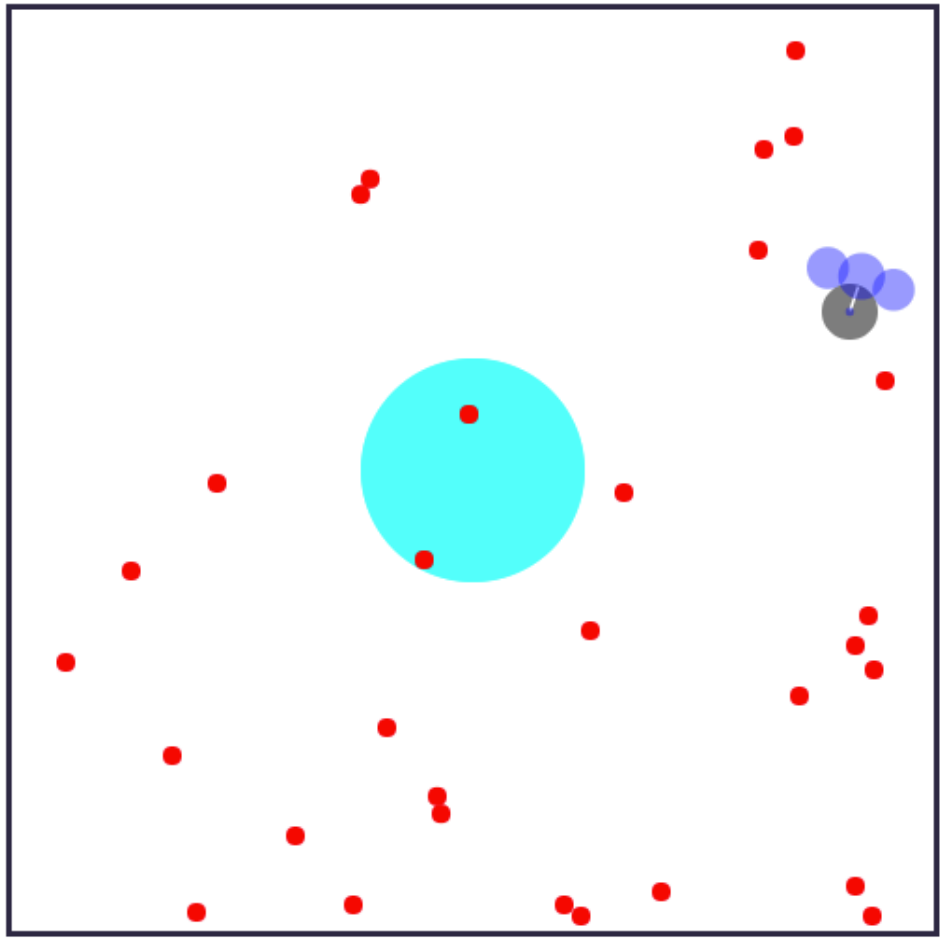
Pheromone quantity on left

Pheromone quantity ahead

Pheromone quantity on right

Gives the number of red pucks in the sensor near the gripper.

The result is a number, not a logical value.



Reset | Allow Movement | Show Sensors

Timescale 1

Desired Population 1

Actual Population 1

Red Pucks 30

Clear | Load | Choose File | No file chosen | Save As | blocks.xml | Show Javascript | Show XML

Status: Ready. Please design your controller below

- Sensors
- Actions
- Memory
- Logic
- Math

Obstacle (wall/robot) on left?

Obstacle (wall/robot) on right?

Within goal zone?

Number red pucks near gripper

red puck held?

Number of flashes

Nest scent quantity on left

Nest scent quantity ahead

Nest scent quantity on right

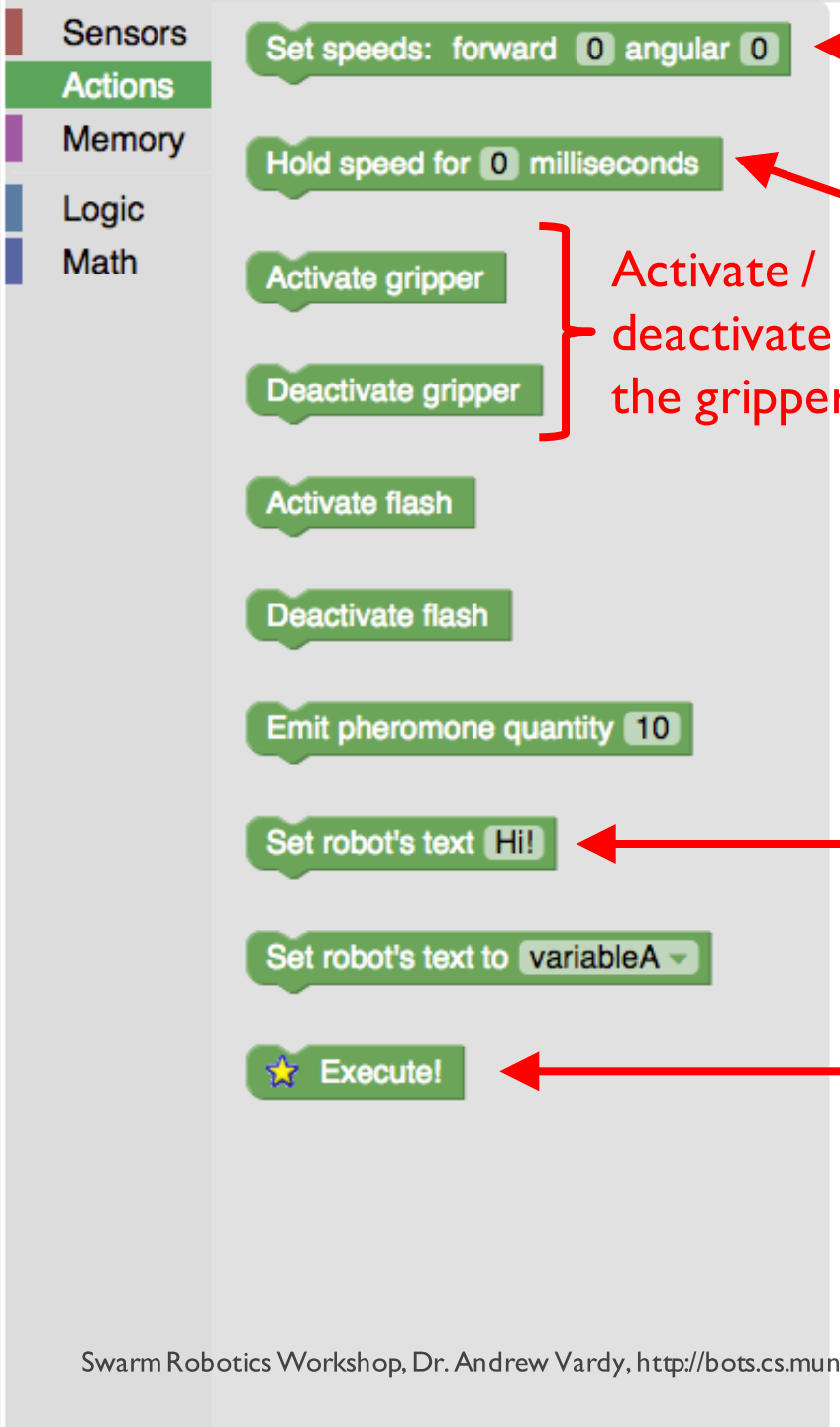
Pheromone quantity on left

Pheromone quantity ahead

Pheromone quantity on right

Logical value indicating whether a puck is currently held or not

"ACTIONS" CATEGORY



Sets forward and angular speeds (as we've seen)

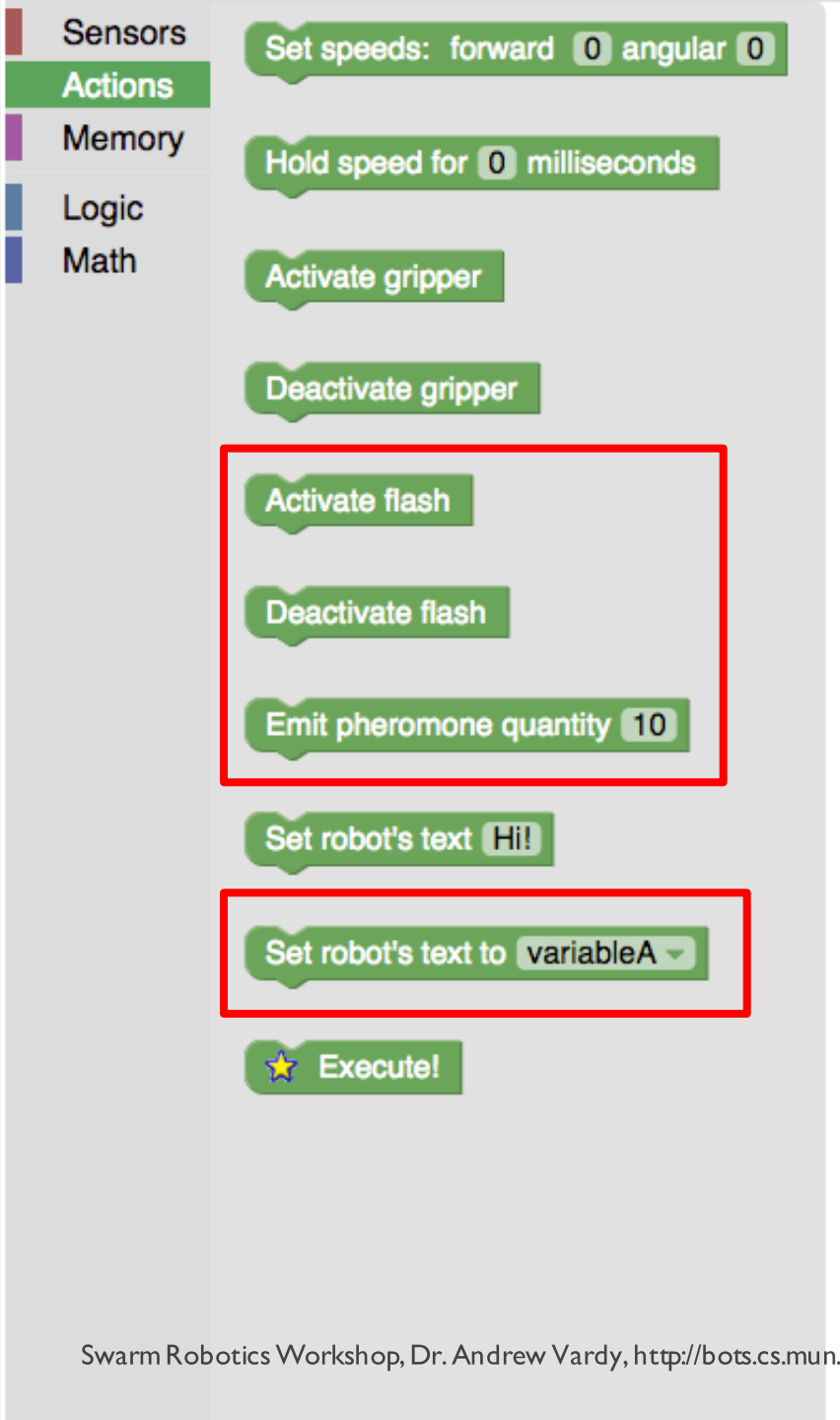
Activate / deactivate the gripper

After "Execute", holds the last commanded speed for this number of milliseconds. Very useful for extended actions such as backing away from something or turning by a desired angle.

Note that while a speed is held, the controller is essentially frozen and cannot receive sensor data.

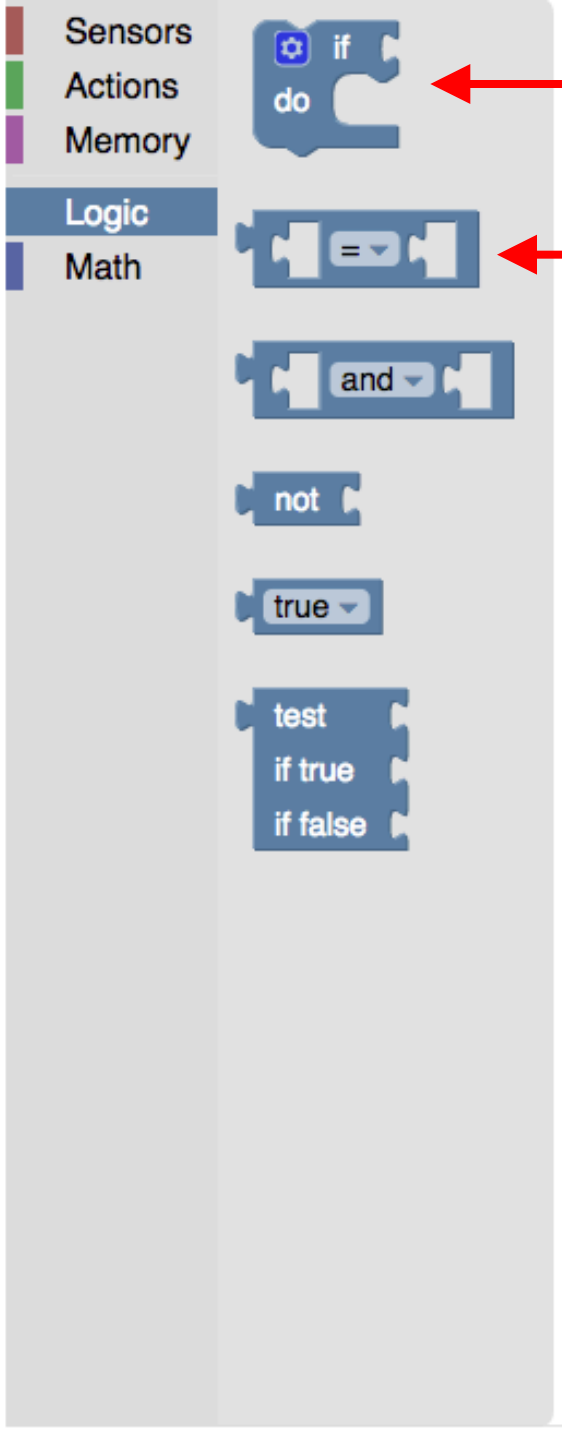
Displays the given text on top of the robot

Causes the controller to execute for one time step



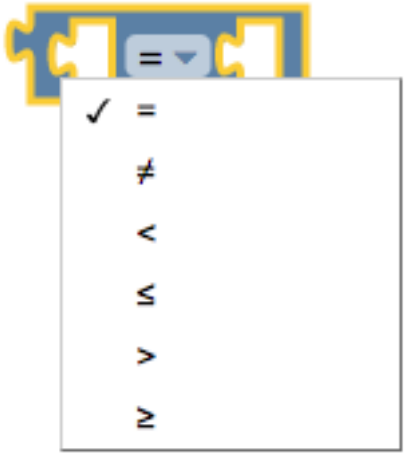
Remaining action blocks will be described later

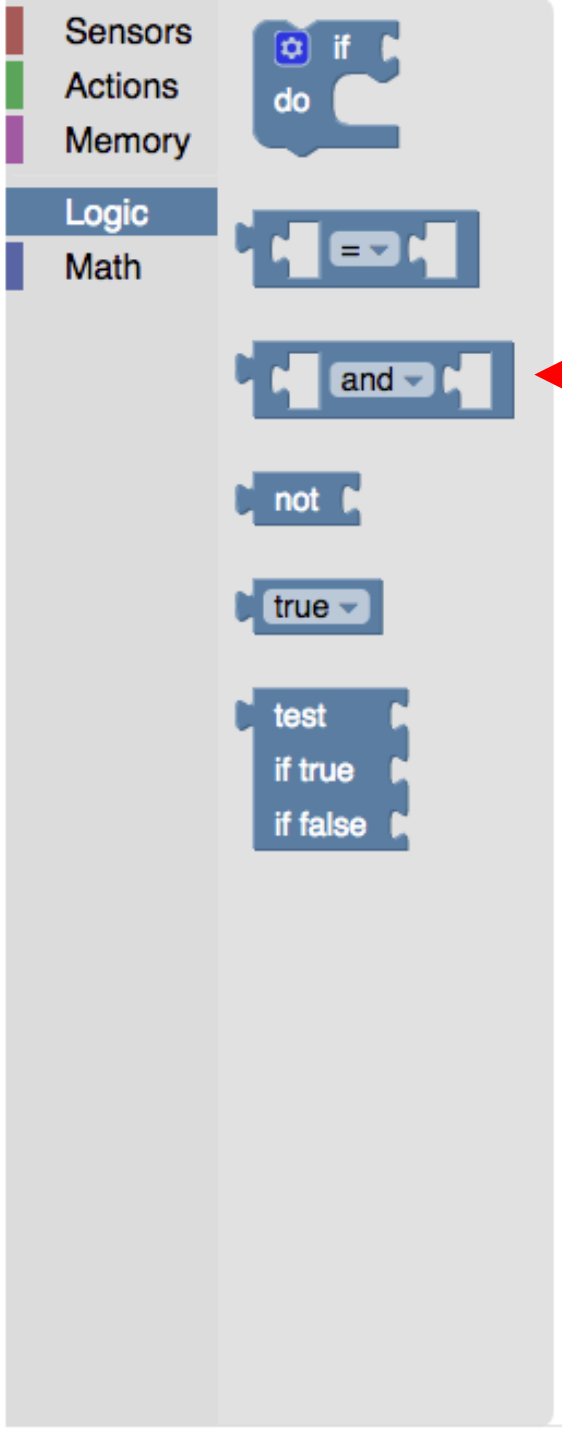
"LOGIC" CATEGORY



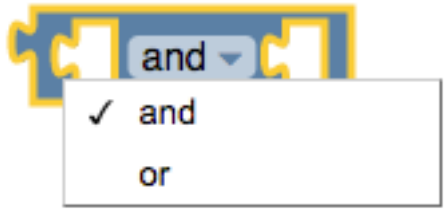
The conditional block. Configure by clicking star. **Super important!**

Relational operator. Compares the two number blocks and produces a logical value. By default tests numbers for equality, but this can be configured:



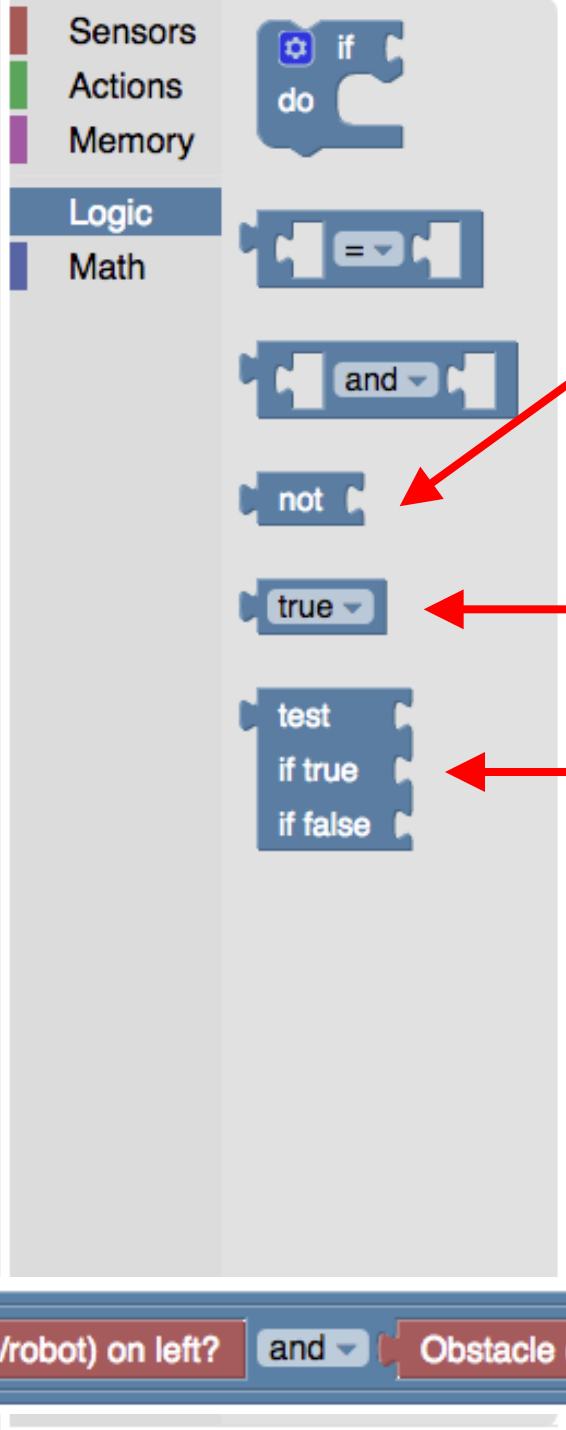


Compares logical values. The two slots are for the logical blocks to compare. By default, produces True only if both values are True, but this can be configured:



and: True only if both inputs are True

or: True if one or both inputs are True

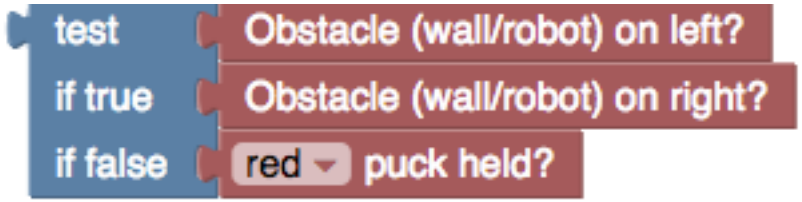


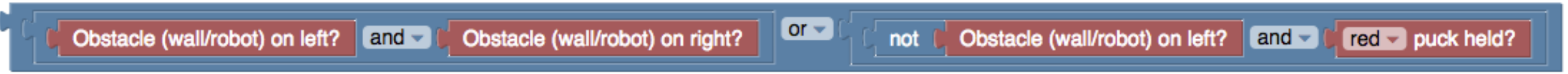
Negates a logical value:
- True becomes False
- False becomes True

Always produces True (or False if selected)

(I've never used this... but you might like it)

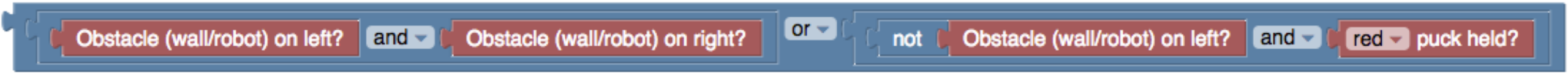
If the "test" condition is true, returns the same logical value as the "if true" condition. Otherwise, returns the same logical value as the "if false" condition. e.g. the following commands are equivalent:



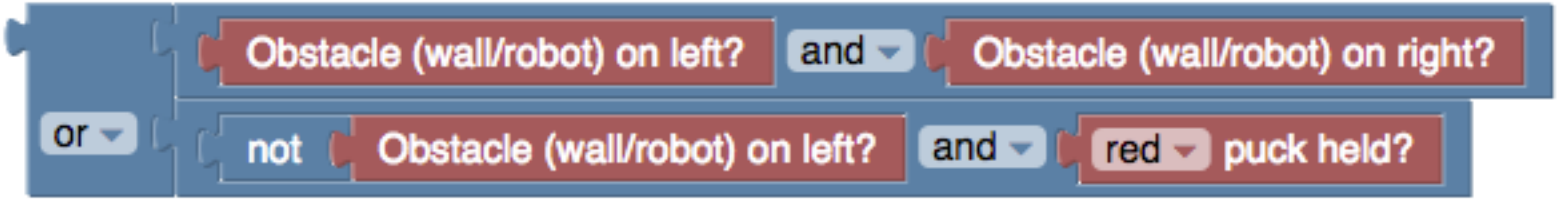


BOOLEAN EXPRESSIONS

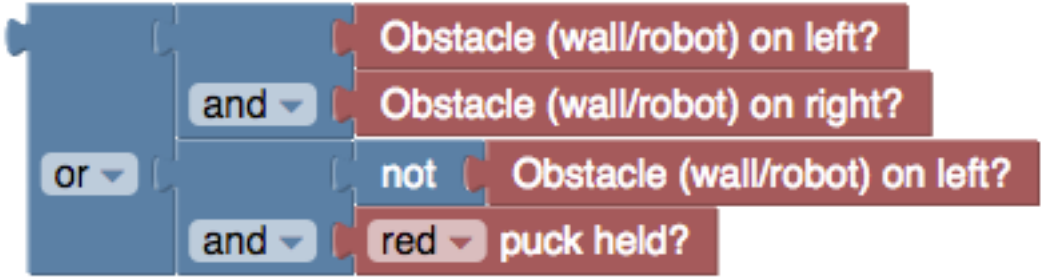
- You will sometimes need to form long expressions like this:



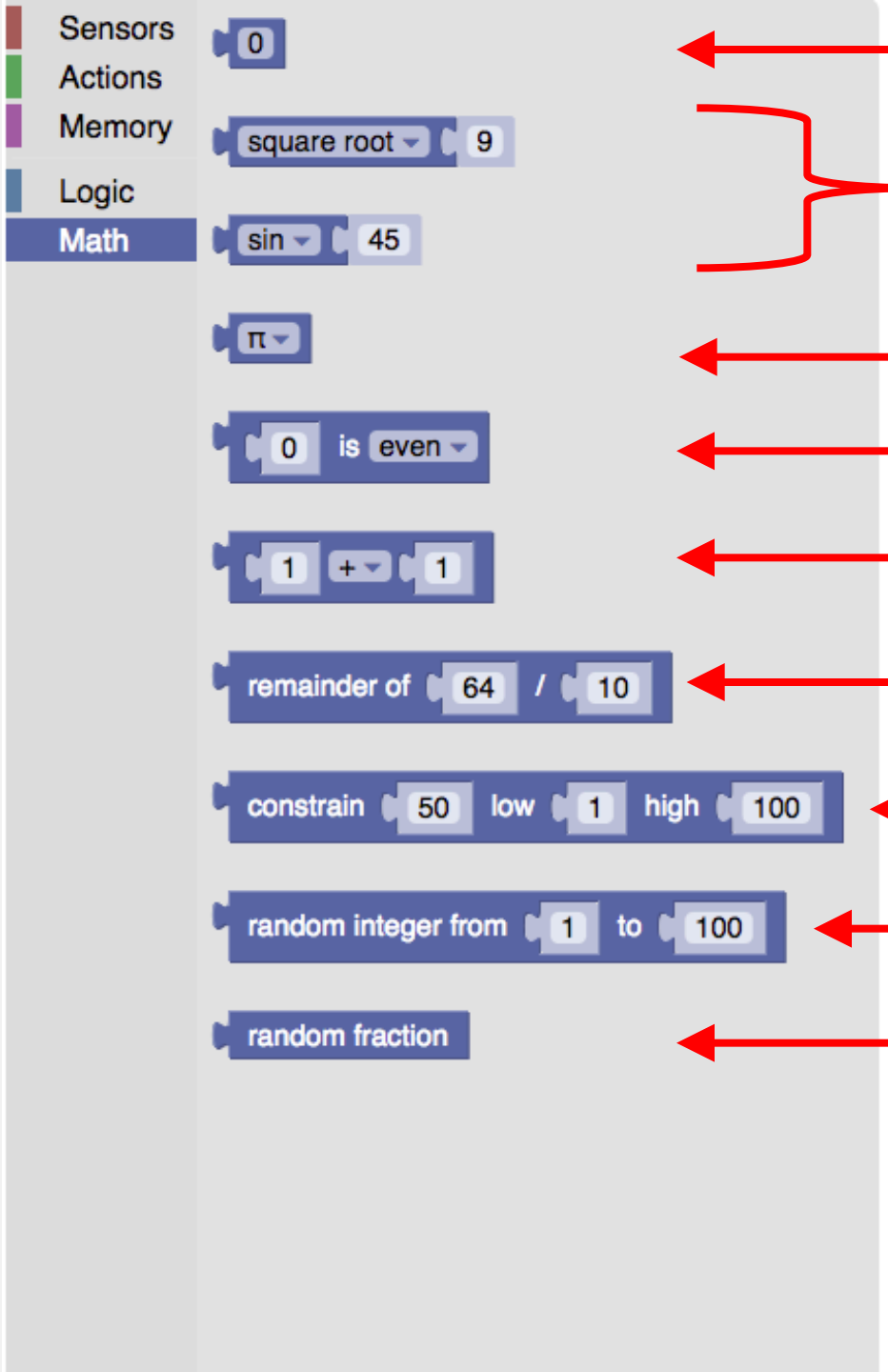
- By right-clicking and selecting "External Inputs" you can re-format like so...



- Applying "External Inputs" to the sub-blocks we get to this form...



"MATH" CATEGORY



A fixed number

Unary operations on numbers

A fixed special number

Check number property (produces logical value)

Binary operation

Computes remainder after division

Constrains number within limits

Random integer within limits

Random number from 0 to 1

EXERCISE #4:

DON'T BUMP THE PUCKS!

15 MINUTES

Conditions:

1 robot

- Start with the obstacle avoidance controller:

- In the pre-clustering level create/load this controller; Increase the number of pucks to 100 and hit "Reset"
  - The robot bumps into the pucks
- Your goal is to create a controller that avoids the pucks, bumping into them as little as possible

EXERCISE #4:

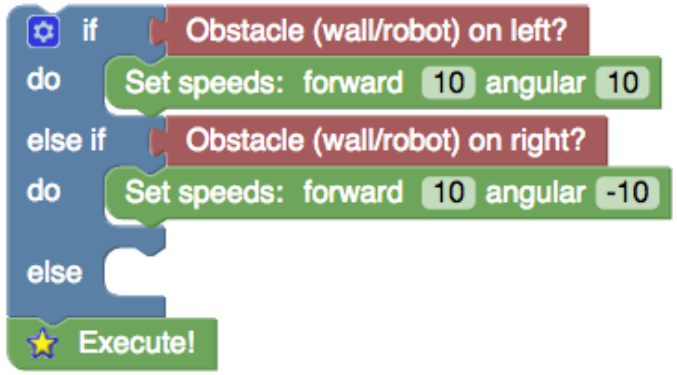
DON'T BUMP THE PUCKS!

15 MINUTES

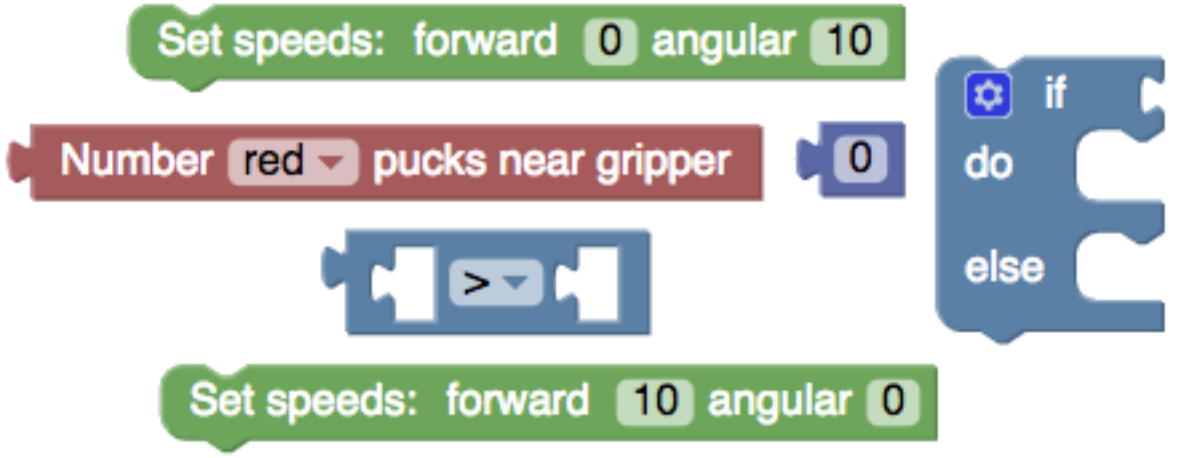
Conditions:

1 robot

- Modify the controller as follows:



- Rearrange the following blocks to go into the empty slot:



- The resulting controller won't be perfect, but it should collide with fewer pucks

EXERCISE #4:

DON'T BUMP THE PUCKS!

15 MINUTES

Conditions:

1 robot

- Your controller will still probably collide with the occasional puck
- Try rearranging your controller and tuning parameters to see if you can further reduce collisions

EXERCISE #5:

FORAGING

20 MINUTES

Conditions:

10 or more robots

- Write a new controller that wanders about, looking for pucks. Upon encountering a puck it should pick it up. If a robot carrying a puck encounters the goal zone, it should drop it.
- You will need the following components (along with others from the obstacle avoidance controller):

- CHALLENGES:
  - Robots may collide with pucks already in the goal zone. How can we handle this?
  - If using fewer robots, they may not explore the environment fully. How can we handle this?

OBJECT CLUSTERING

ANT CEMETARY CONSTRUCTION

- Biologists have noticed many fascinating examples of social insects organizing their environments
- e.g. Colonies of ants will cluster dead ants together, seemingly without any planning or supervision
- A computational model was proposed to explain this behaviour:
  - [Deneubourg, J. L., Goss, S., Franks, N., Sendova-Franks, A., Detrain, C., & Chrétien, L. (1991, February). The dynamics of collective sorting robot-like ants and ant-like robots. In *Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats* (pp. 356-363).]

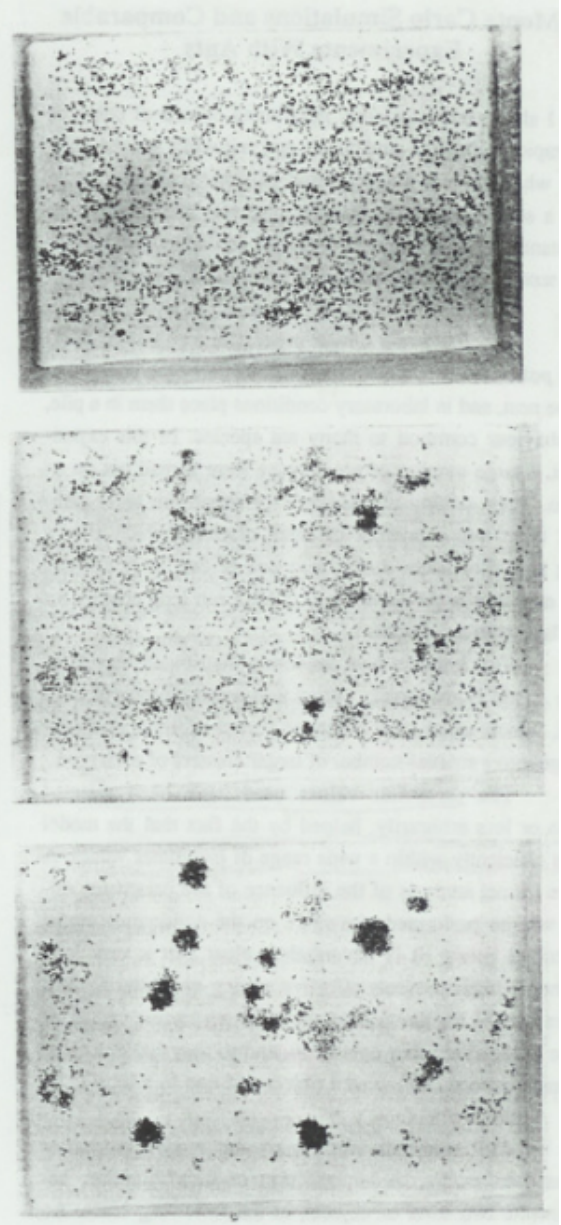
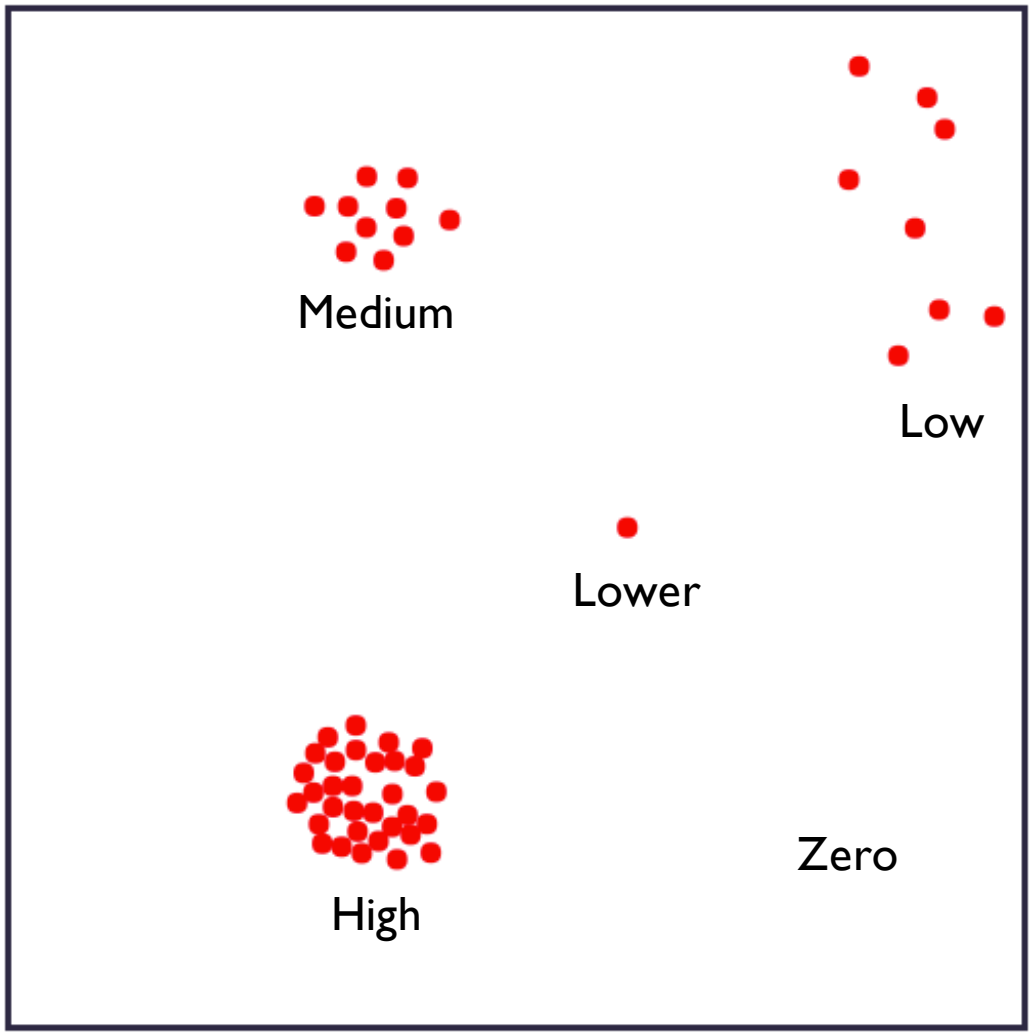


Fig. 2. Clustering in a colony of *Pheidole pallidula*. 4000 corpses were placed on a 50x50cm arena, and photos taken at time 0, 20 and 68 hrs. Small evenly spaced clusters rapidly form, and later merge into fewer larger clusters.

- Deneubourg et al's model:
  - Agents measure **local object density** by maintaining a short-term memory and counting the number of recent object appearances
  - Agents walk randomly and pick-up or deposit objects as a probabilistic function of local object density

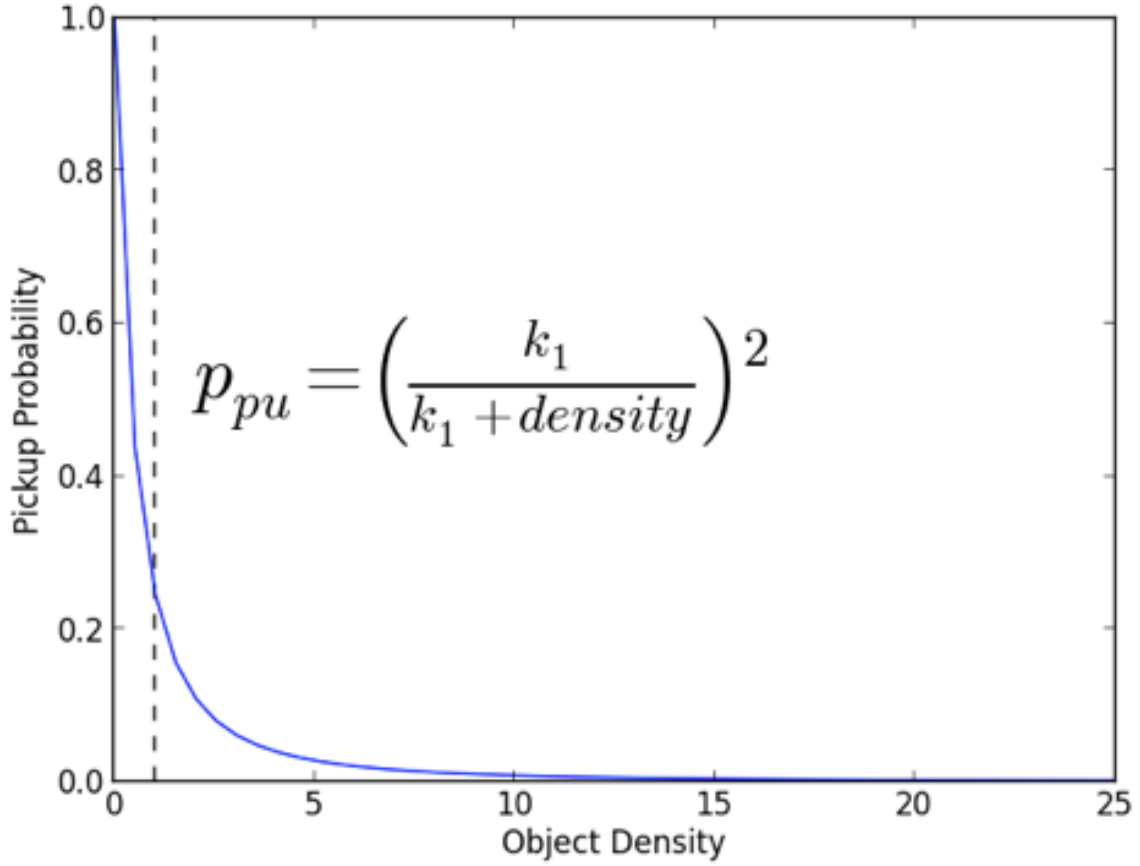
Density	Pick-up Probability (if not carrying)	Deposit Probability (if carrying)
Low	High	Low
High	Low	High

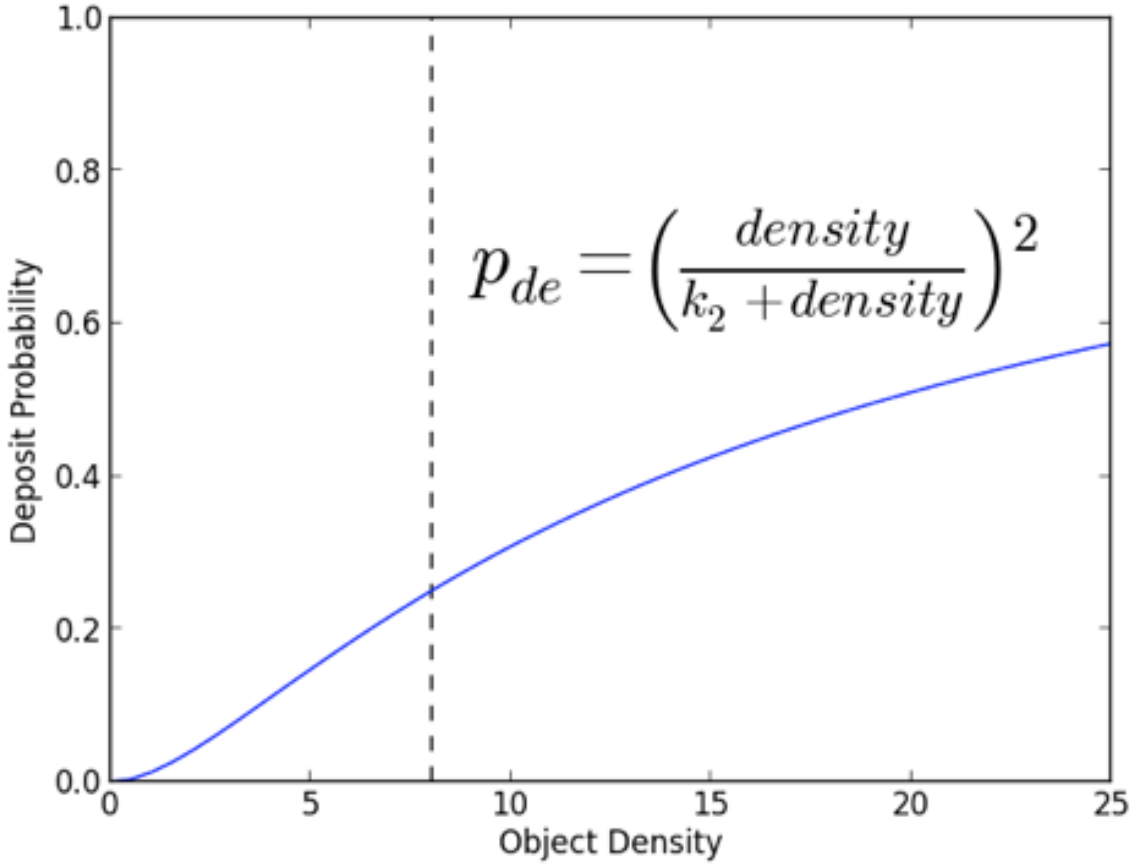
Local object density (cartoon version)



Not carrying

Carrying





- Experiments on the computational model closely match the results of biological experiments
- But we have to be careful:
  - Similarity of results does not necessarily imply similarity of method

Left: Computer Model

Right: Biological Experiment

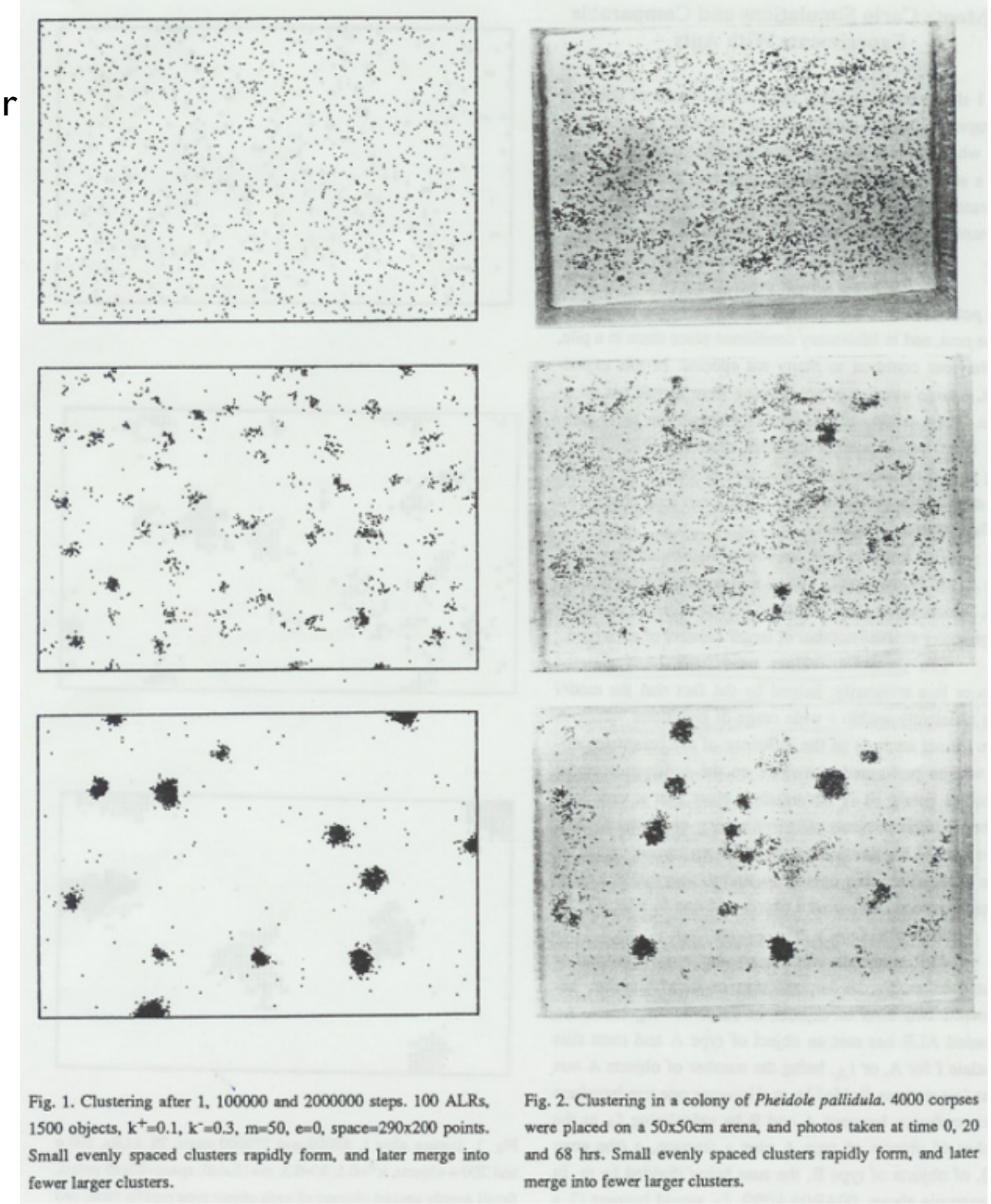


Fig. 1. Clustering after 1, 100000 and 2000000 steps. 100 ALRs, 1500 objects, k^+=0.1, k^-=0.3, m=50, e=0, space=290x200 points. Small evenly spaced clusters rapidly form, and later merge into fewer larger clusters.

Fig. 2. Clustering in a colony of *Pheidole pallidula*. 4000 corpses were placed on a 50x50cm arena, and photos taken at time 0, 20 and 68 hrs. Small evenly spaced clusters rapidly form, and later merge into fewer larger clusters.

BECKERS ET AL

- Beckers et al. wrote a paper detailing their experiments in swarm robotic clustering
  - Beckers, R., Holland, O. E., & Deneubourg, J. L. (1994, July). From local actions to global tasks: Stigmergy and collective robotics. In *Artificial life IV* (Vol. 181, p. 189).
- Unlike the [Deneubourg et al., 1990] model, the robot's pick-up / deposit behaviour is implicit

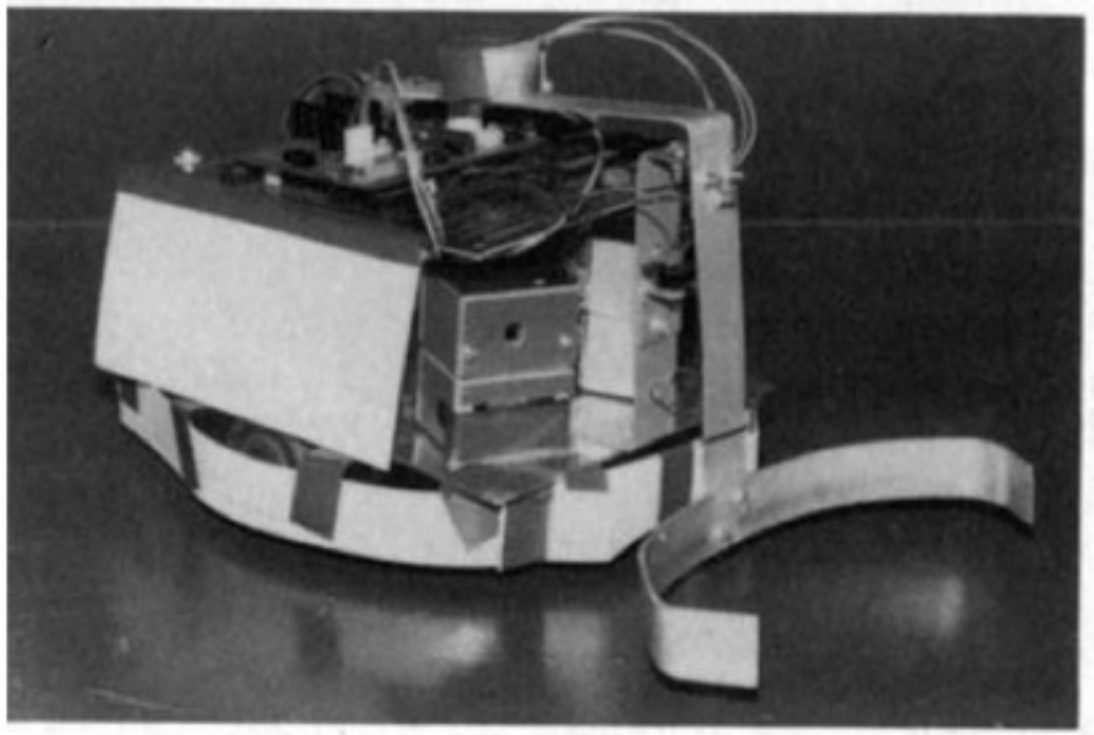


Figure 1: Robot equipped with a gripper for object Gathering. Experiments were carried out with from 1 to 5 robots of the same type,

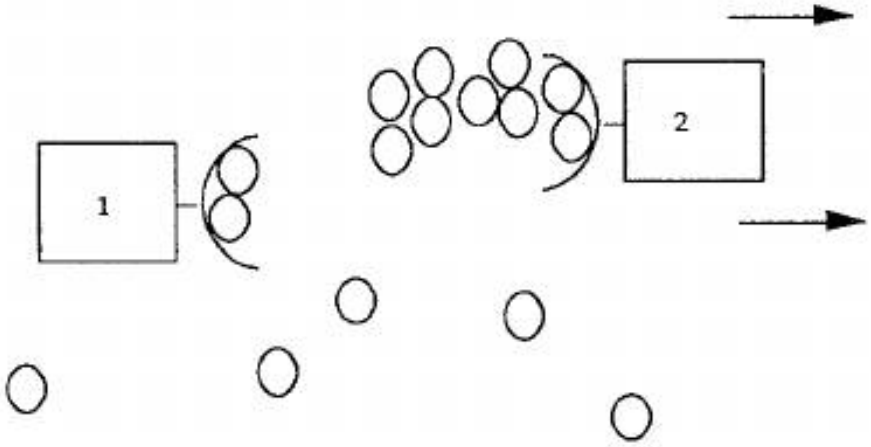


FIGURE 4.15 Robot 1, because it has only two pucks in its C-shaped gripper, continues to move in a straight line. Robot 2, with three pucks in its gripper, is encountering a cluster: its microswitch is activated by the gripper, so that the robot leaves the pucks next to the cluster.

- C-shaped gripper passively collects pucks
- Infrared sensors detect obstacles (walls, other robots)
  - Behaviour: Triggers random turn away from obstacle
- Force sensor detects that gripper is pushing against three or more pucks
  - Behaviour: Triggers backup, then a random turn, resulting in the pucks being left behind (i.e. deposited)
- If no behaviour is triggered, the robot just moves straight

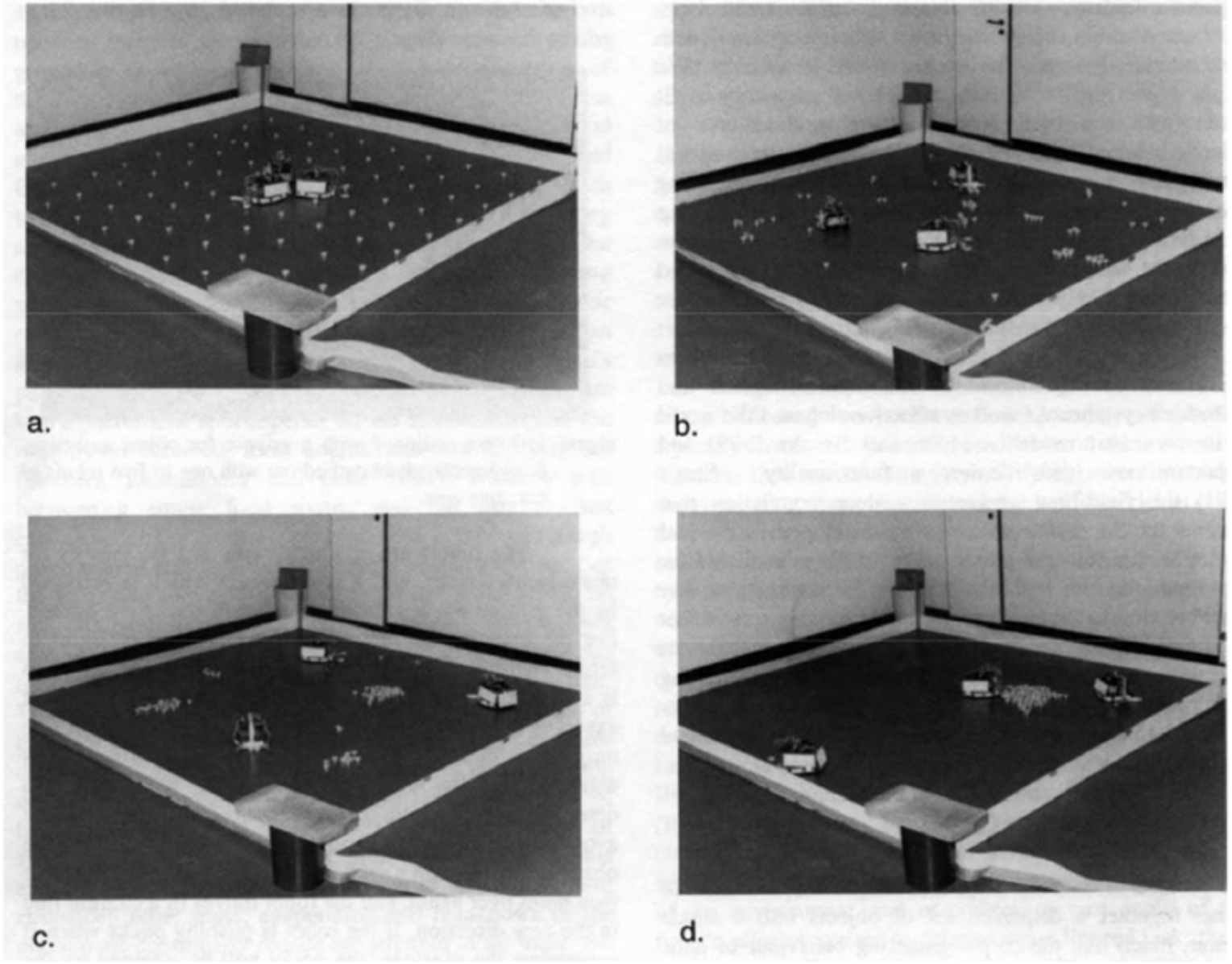


Figure 2. The initial setup (a) and time evolution of a typical experiment involving a group of three robots. Phase 1 (b), occurring after approximately 10 min, is characterised by a large number of small clusters containing from 1 to 10 pucks. In Phase 2 (c), some clusters grow rapidly and the environment becomes more heterogeneous. Finally, Phase 3 (d) is characterised by the competition between a small number (2 – 3) of large clusters and evolves towards the clustering of all objects in one pile.

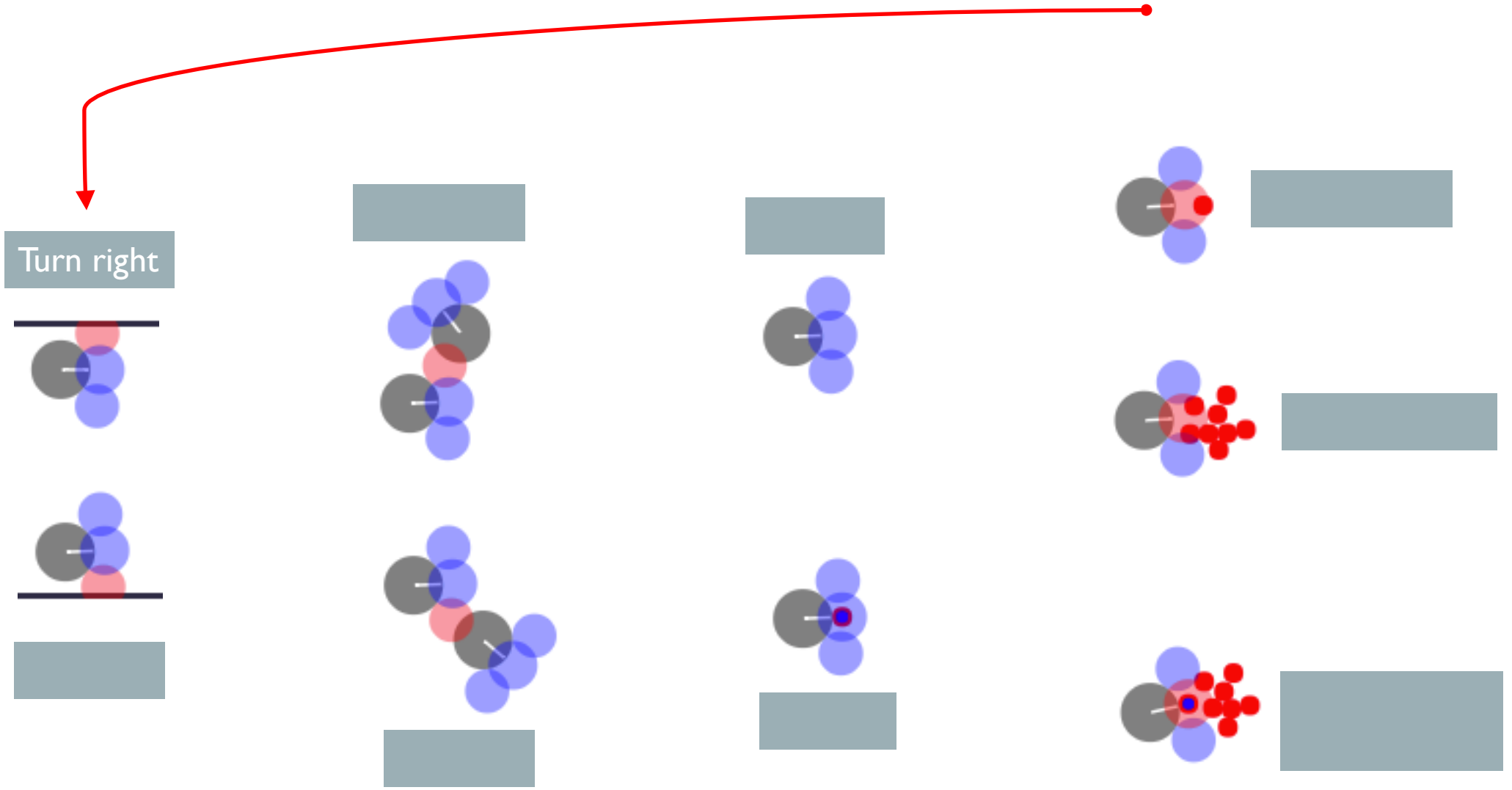
- Unlike Deneubourg et al. there is no explicit sensing of local object density
- Yet when a high density area is encountered, the robot tends to further increase density by backing up and leaving its pucks behind
  - POSITIVE FEEDBACK:
    - Larger clusters are encountered more often, triggering further growth
  - NEGATIVE FEEDBACK:
    - When smaller clusters are encountered their pucks are taken away

CLUSTERING IN WAGGLE

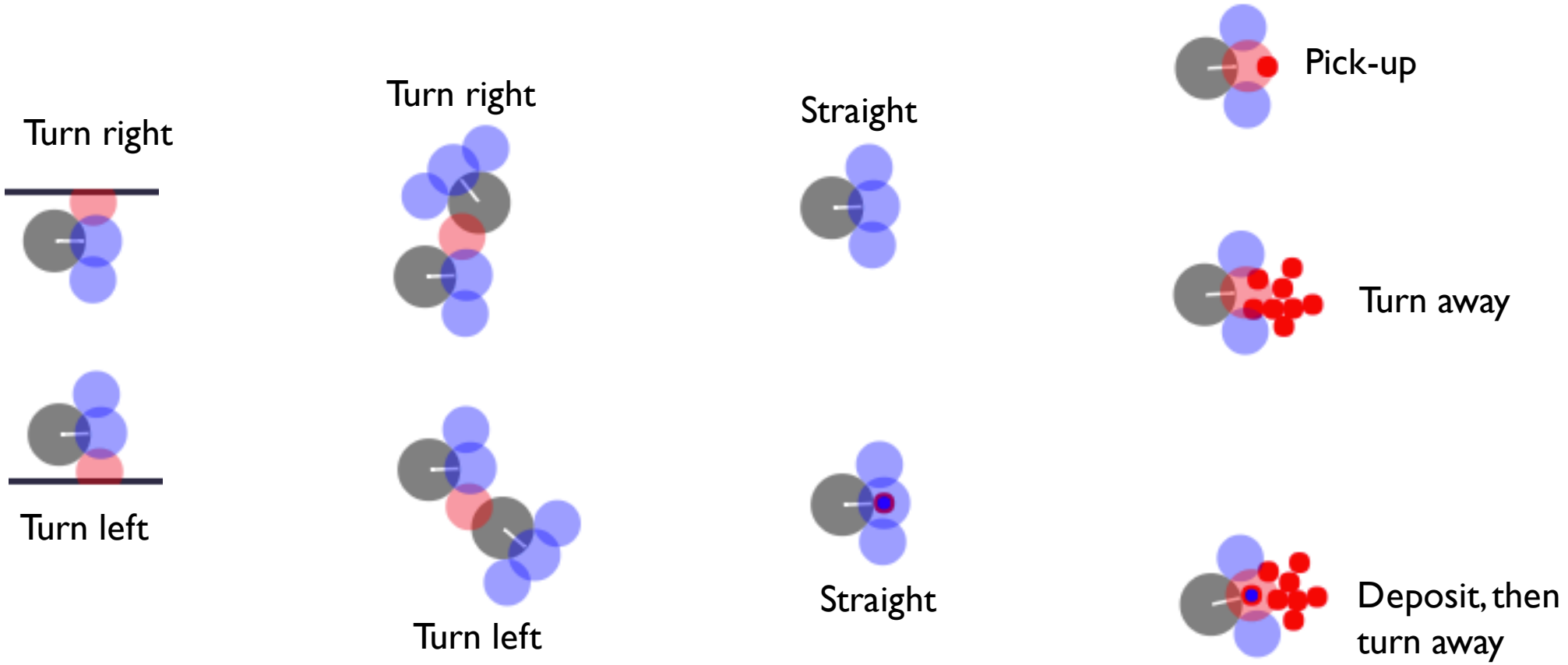
- The next task will be to consider a number of different possible sensor states and decide on reactions for each one
- We'll have to consider different numbers of pucks detected by the robot and decide whether they represent low or high density

- For simplicity, lets say that a single puck is low-density and two or more pucks is high-density

- Fill in the blanks with the choices the robot should make; e.g.



- The answers are shown here:



- Notes:
  - To turn away by a fixed angle, use both "Set speeds" and "Hold speed"
  - There are conditions not shown such as combinations of obstacles and pucks

EXERCISE #6:

OBJECT CLUSTERING

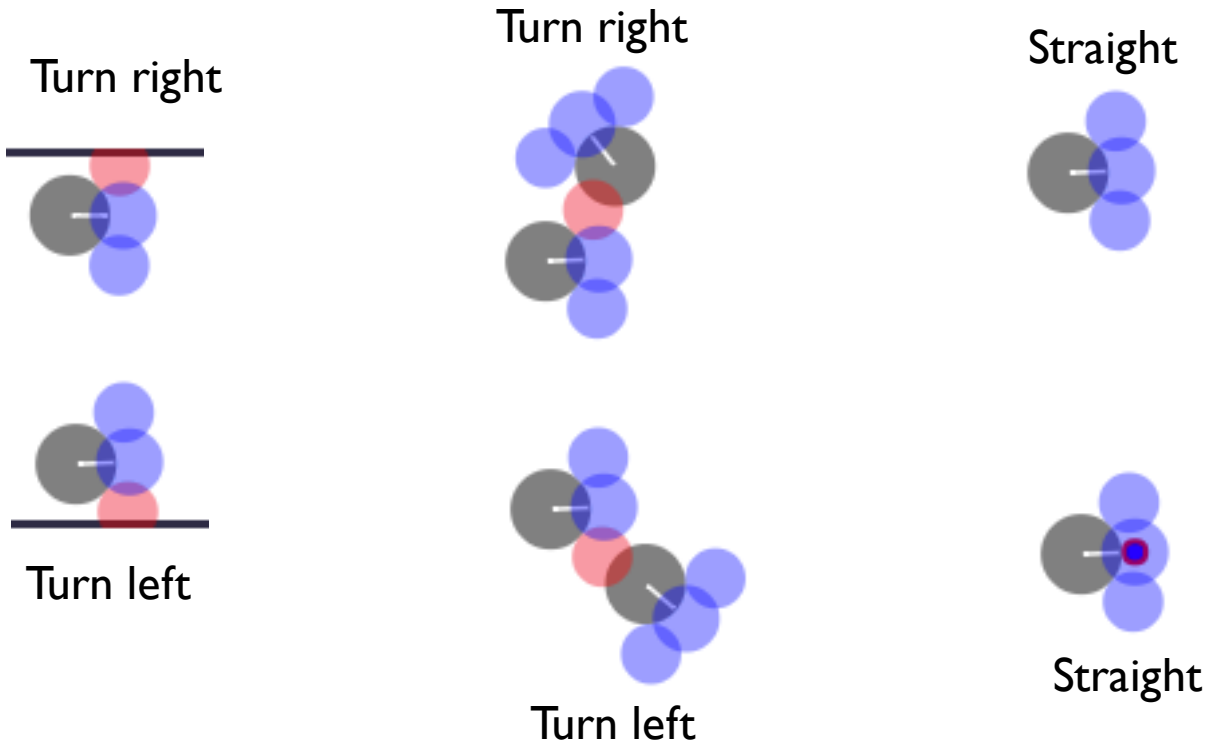
20 MINUTES

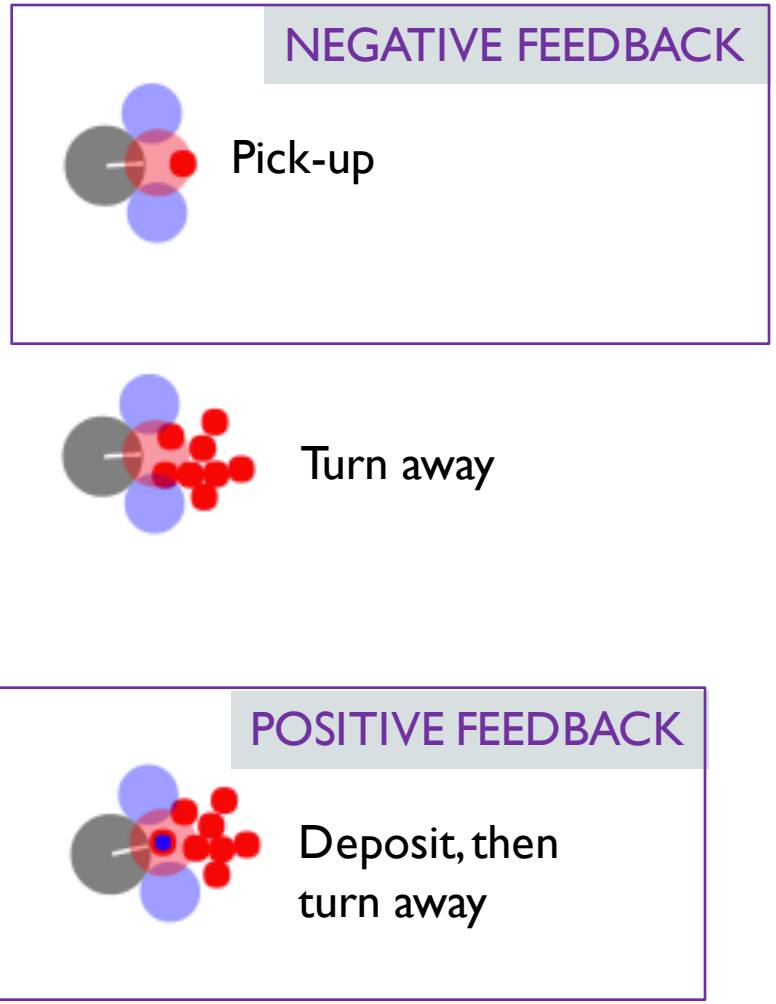
Conditions:

Any number of robots and pucks

- Create a new controller that has the appropriate response for all of the conditions just described
- Beneath the simulation controls is a new plot called "Percentage Completion"
  - Details:
    - Computes the size of the largest cluster. Let this size be L
    - PC = 100 * L / (number of pucks)
- CHALLENGE:
  - With 10 robots and 30 pucks, reach 60% completion within 200 seconds

- Why does this work?





- The other behaviours are necessary but only these two have a direct impact
- Larger clusters attract more deposits than smaller clusters, leading to further growth and the gradual absorption of smaller clusters:
  - Kazadi, S., Abdul-Khaliq, A., & Goodman, R. (2002). On the convergence of puck clustering systems. *Robotics and Autonomous Systems*, *38*(2), 93-117.

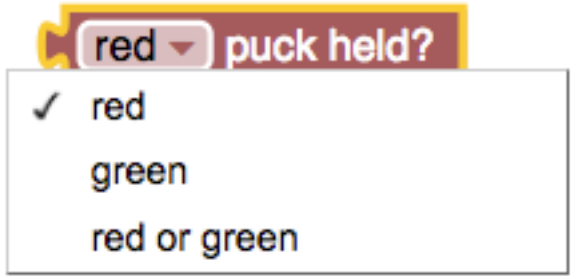
OBJECT SORTING

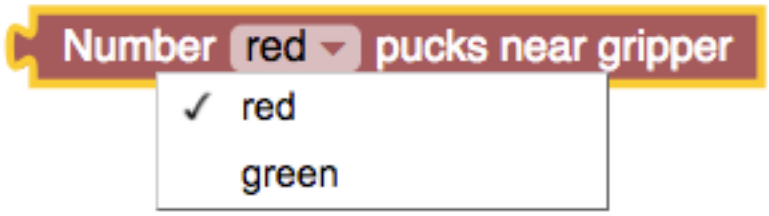
OBJECT SORTING

- Sorting objects is a natural extension of clustering
  - Clustering: One object type
  - Sorting: Two object types
- It has potential applications in recycling, mining, and warehousing
- Vardy, A., Vorobyev, G., & Banzhaf, W. (2014). Cache consensus: rapid object sorting by a robotic swarm. *Swarm Intelligence*, *8*(1), 61-87.

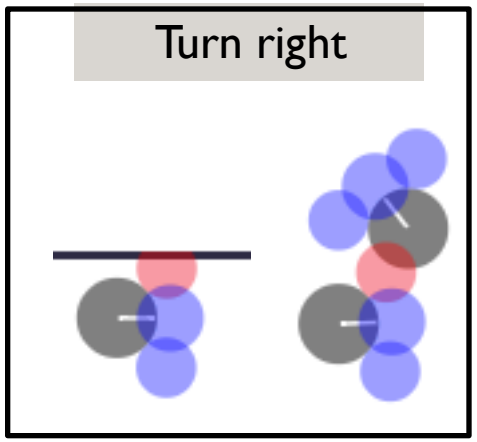
SORTING IN WAGGLE

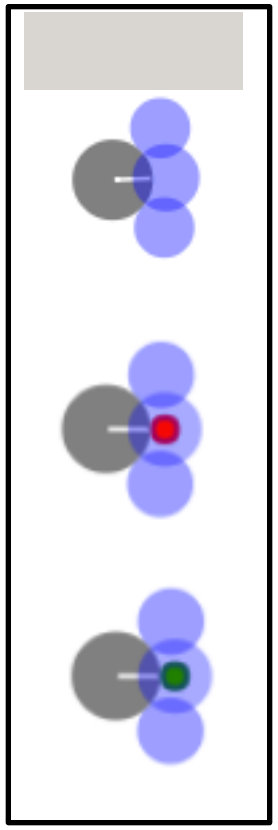
- Go to the main Waggle page (or just hit your browser's back button):
  - http://bots.cs.mun.ca/waggle/
- Select the **Sorting** level. Note that there are now two different colours of pucks available: red and green
- We will use the same sensor and action blocks as before, but will need to customize the "___ puck held" and "Number _____ pucks" blocks:

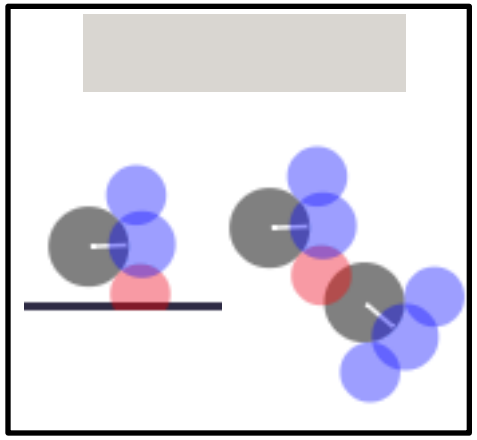


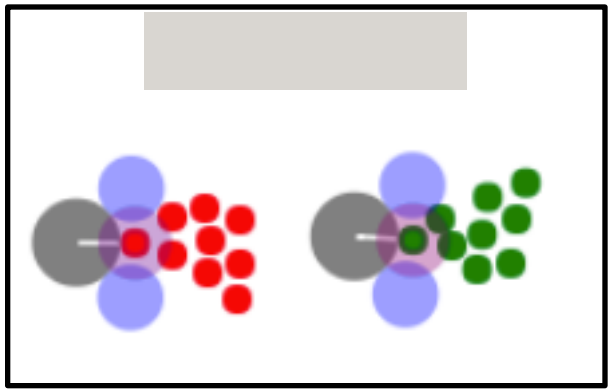


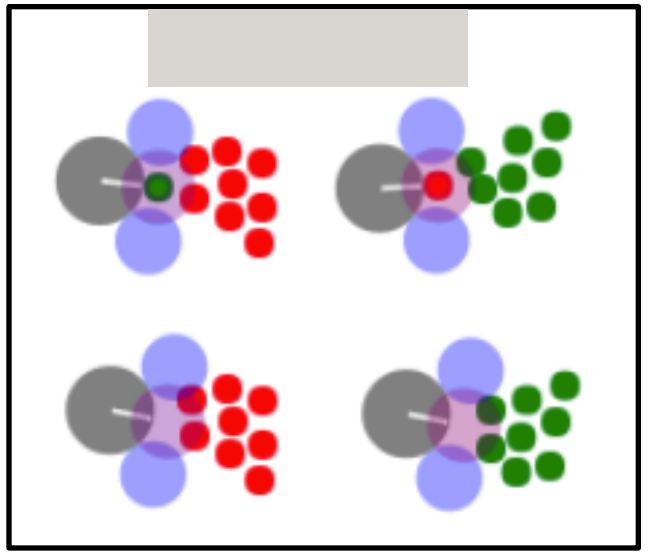
- When sorting two colors, we have more choices to make
- Can you fill in the blanks?

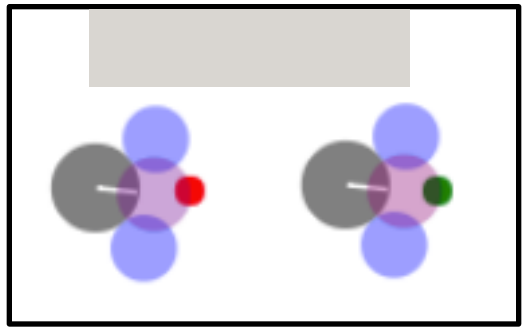


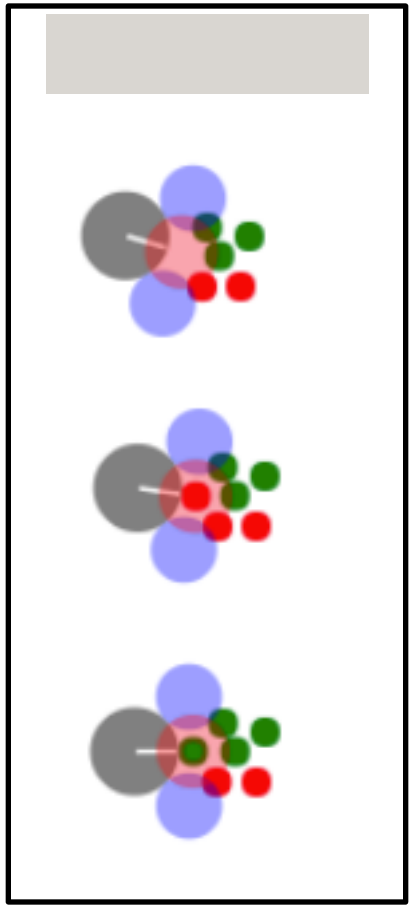




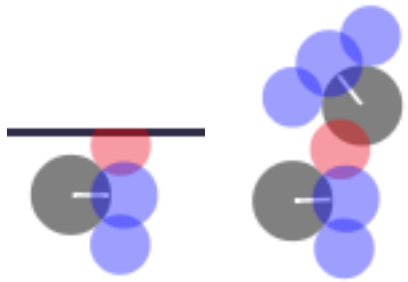






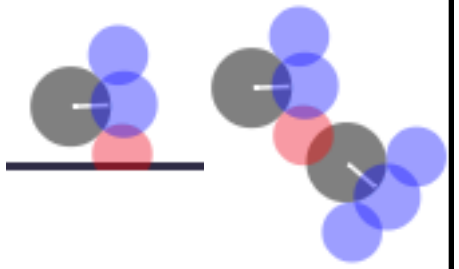


Turn right



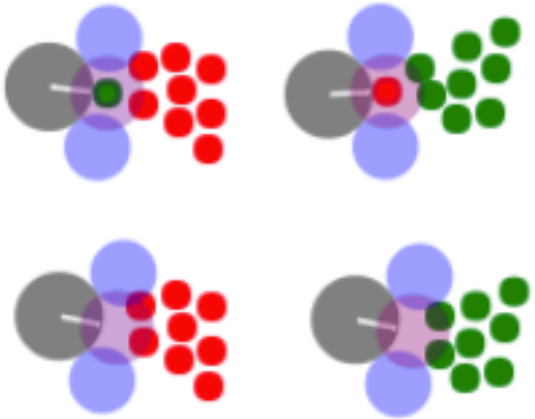
Straight

Turn left



Deposit, then turn away

Turn away



Pick-up

Straight (breaking up mixed cluster)

EXERCISE #7:

OBJECT SORTING

20 MINUTES

Conditions:

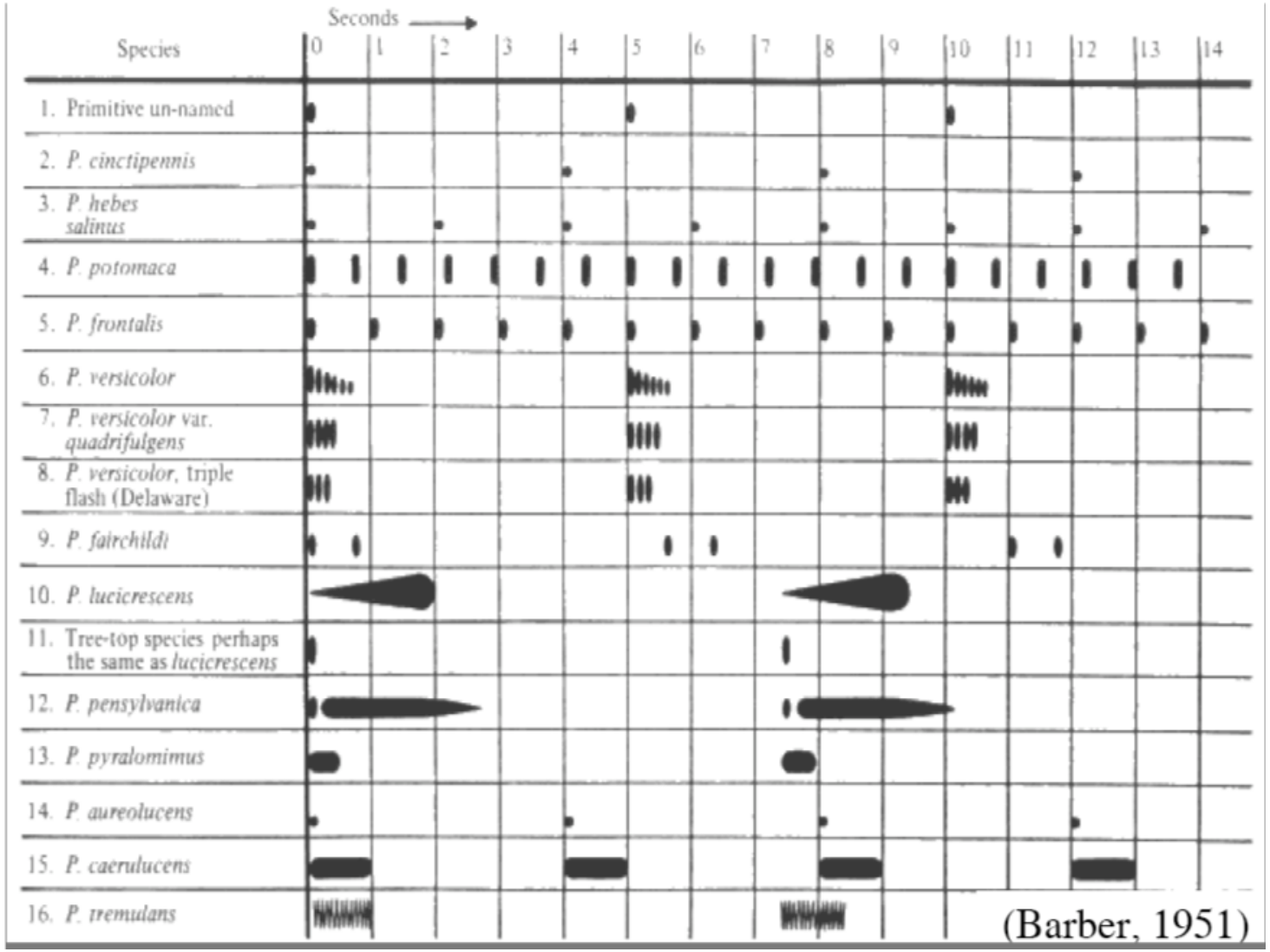
Any number of robots and pucks

- Create a new controller that has the appropriate response for all of the conditions just described
- CHALLENGE:
  - With 10 robots, 30 red pucks, and 30 green pucks, reach 50% completion within 200 seconds

SYNCHRONIZATION

FIREFLY SYNCHRONIZATION

- Fireflies exhibit a remarkable behaviour where they flash in synchrony, apparently for reproduction
- Since different species have different characteristic patterns, when the males flash appropriately the females can tell they have found an appropriate mate



A SIMPLIFIED VIEW

- The patterns of flashing in fireflies can be complex
- We take the simplified view of fireflies flashing in a single pulse
- Exemplified by this video from Thailand:

WHY IS THIS OF INTEREST TO ROBOTICS?

- Having a swarm of robots flash in synchrony has been proposed as a way to detect failures in other robots:
  - That one's not flashing right; It must be broken!
  - Christensen, A. L., OGrady, R., & Dorigo, M. (2009). From fireflies to fault-tolerant swarms of robots. *IEEE Transactions on Evolutionary Computation*, *13*(4), 754-766.
- In general, we're interested in how distributed agents can agree globally when each of them only has a very limited view of the world
  - e.g. Agree on a set of locations for coloured pucks

FLASHING IN WAGGLE

- Go to the main Waggle page (or just hit your browser's back button):
  - http://bots.cs.mun.ca/waggle/
- Select the **Fireflies** level. There are no pucks and the robot has lost its puck sensor. But it has a large sensor that can detect flashes from other robots.
- In "Sensors": 

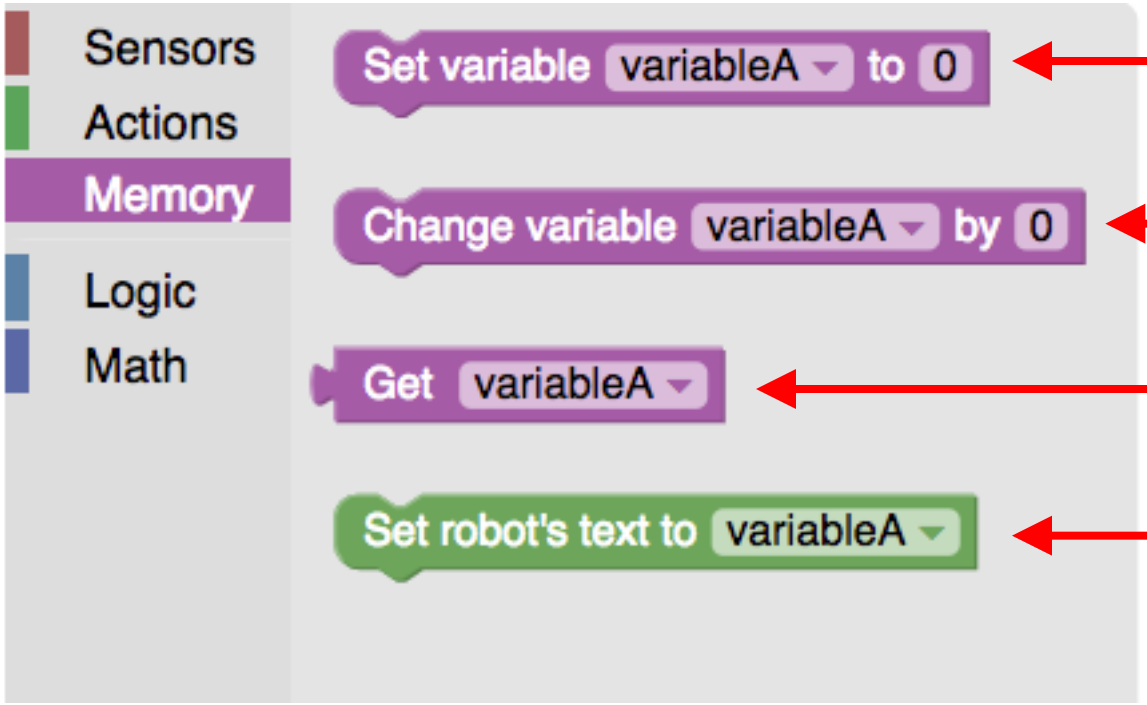
This gives the number of robots within the sensor's area that were flashing during the last time step
- In "Actions": 

These illuminate and de-illuminate the robot

VARIABLES

- All of our controllers so far have been completely **reactive---**behaviour depends only on the present input
- But to make regular pulses of light the robots need to store something about the time since the last flash
- Variables are a fundamental element in programming, but their use in Waggle is pretty limited:
  - Only three variables available: variableA, variableB, and variableC
    - For our purposes we will only need one
  - They are integers, initialized to start at 0

"MEMORY" CATEGORY



Set the variable to the given integer

Modify the variable by adding the given integer

Get the variable's value as a number

Display the variable's value on top of the robot (useful for debugging)

EXERCISE #8:

SIMPLE FLASHING

5 MINUTES

Conditions:

Any number of robots (not trying to synchronize yet)

- Create a controller like this:
  - Add 1 to variableA
  - If variableA = 10
    - Activate flash
    - Set variableA to 0
  - Else
    - Deactivate flash
  - Execute

- See what happens when you increase the number of robots
- Play with "Timescale" slider to see things in slow motion
- Incorporate 

- Note that robots are "born" at different times, and therefore will not be synchronized by default (although those with the same "birthday" will automatically synchronize)

EXERCISE #9:

SYNCHRONIZATION

15 MINUTES

Conditions:

Any number of robots

- Start with the "fixed flasher" just created
- If a neighbouring robot flashes, how should you respond so that you flash at the same time during the next cycle?
  - **Devise your own strategy based on this idea**

- Drag your robots around so that they form groups
- You may wish to deselect "Show Sensors"
- You may not get good performance unless the robots "mingle"
  - Incorporate obstacle avoidance and keep the robots moving while they attempt to sync

PHEROMONES

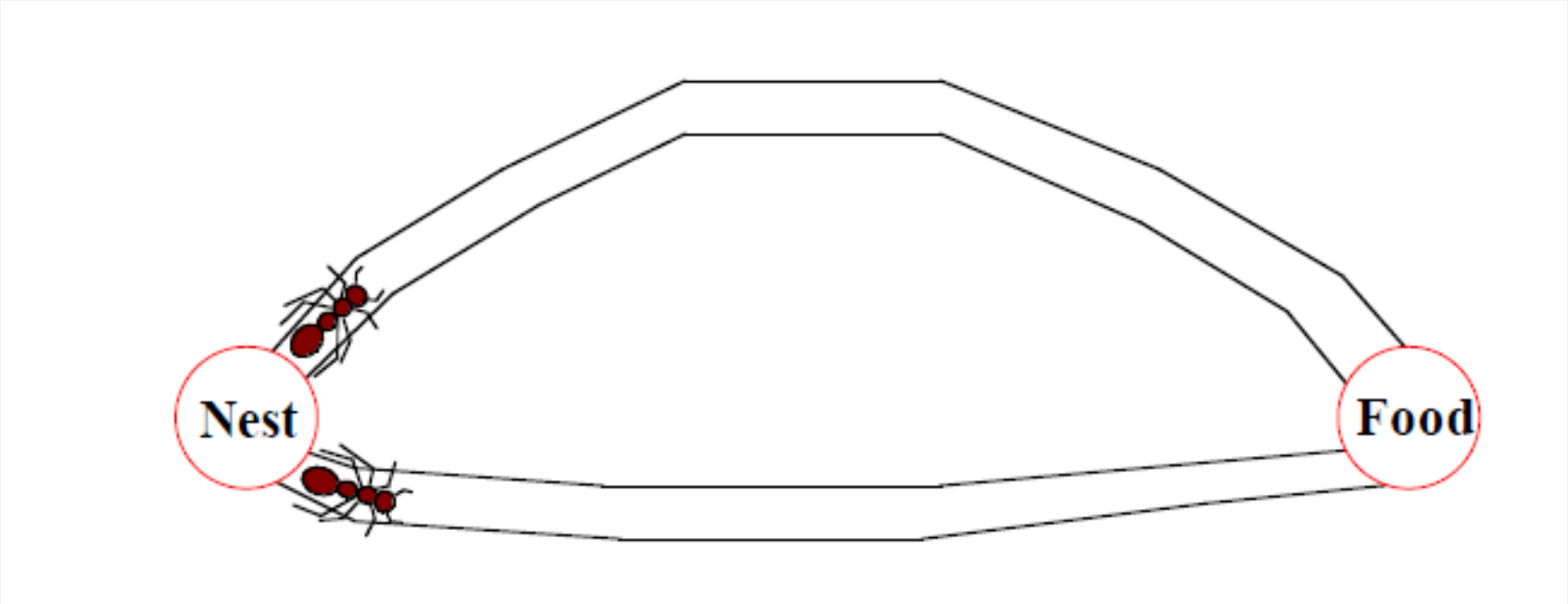
PHEROMONES: INSECT COMMUNICATION

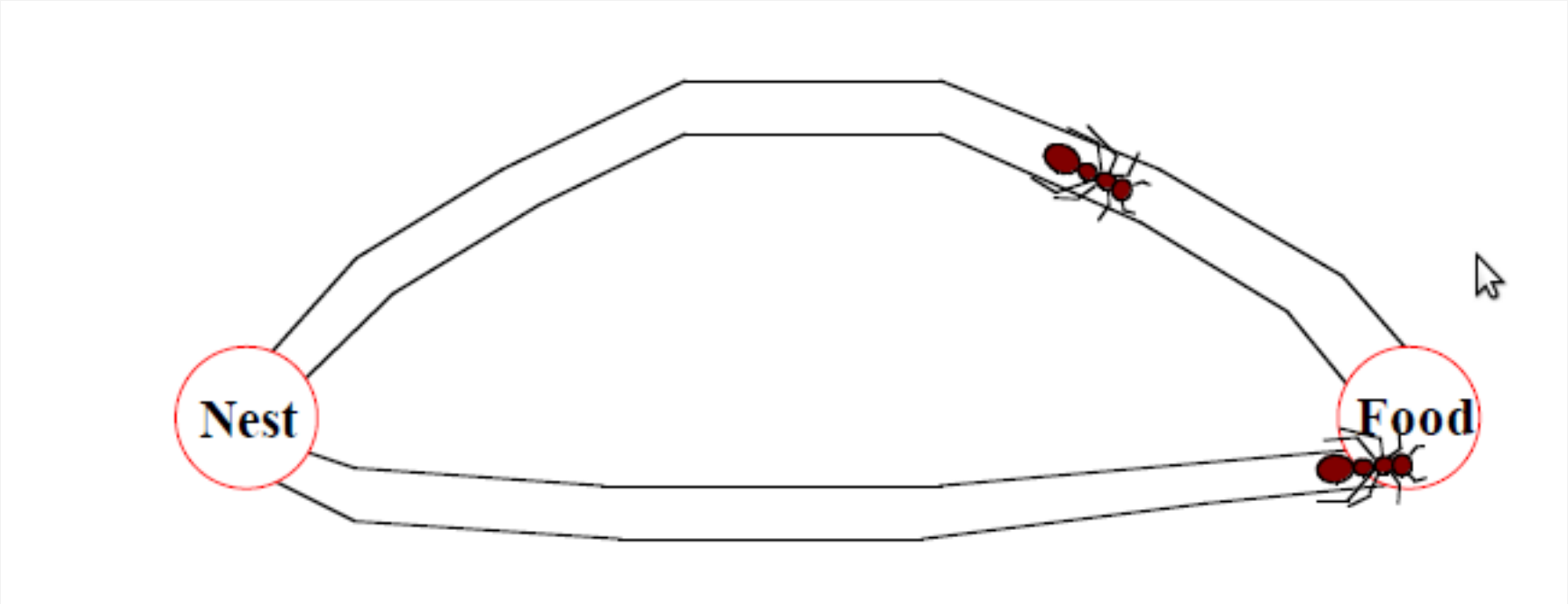
- In many cases, insects deposit chemical signals called pheromones which can indicate something about a certain place or thing
- The classic example of pheromone use is in ant foraging

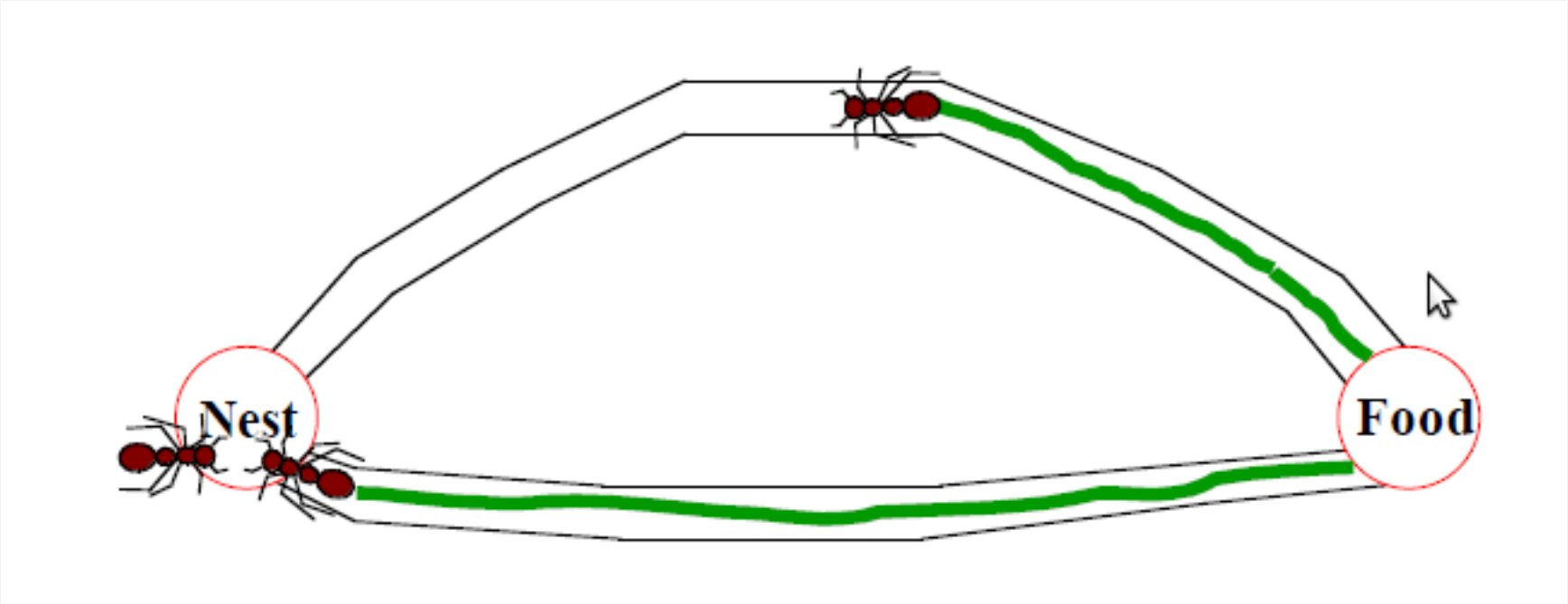
FINDING THE SHORTEST PATH

- When foraging for food, some ants leave behind pheromone trails on the ground:
  - The pheromone is deposited by ants on their way back to the nest
  - After the food source is exhausted the pheromone trail will dissipate
  - Generally ants find the shortest path between food sources and the nest

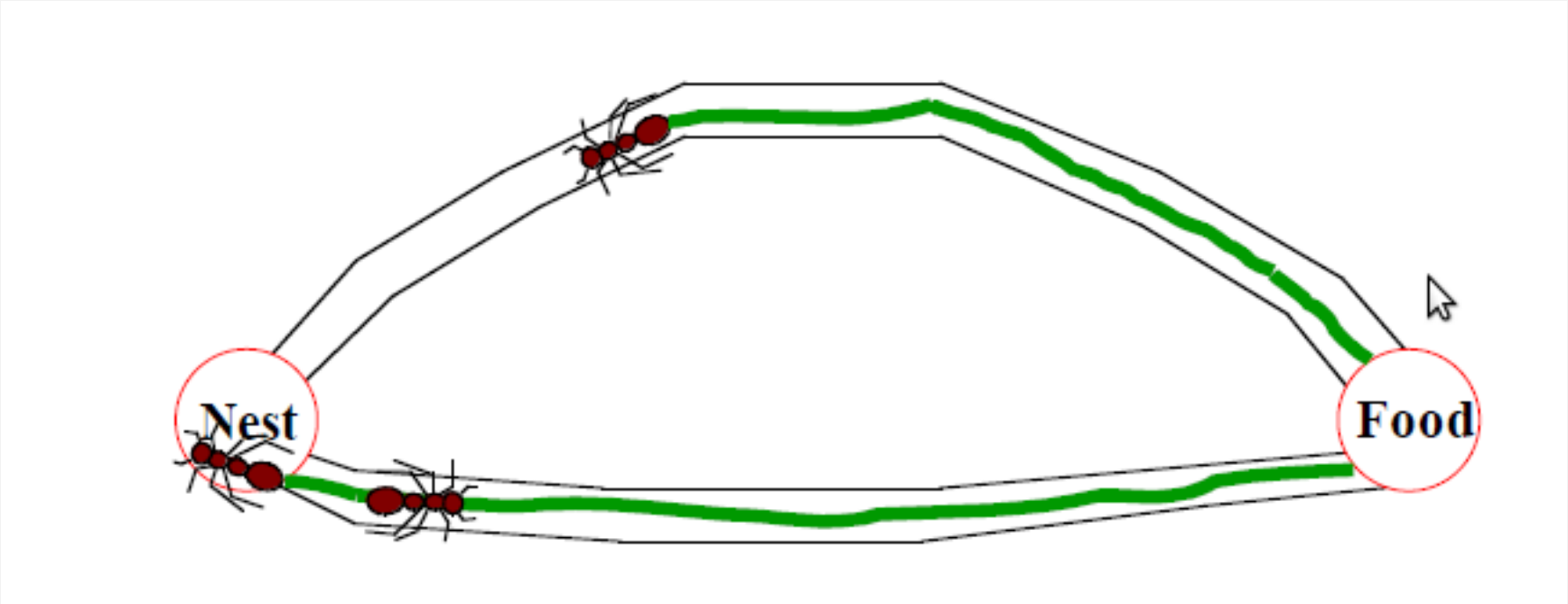
  - That's a big deal! How can these tiny insects solve the problem of finding the shortest path?

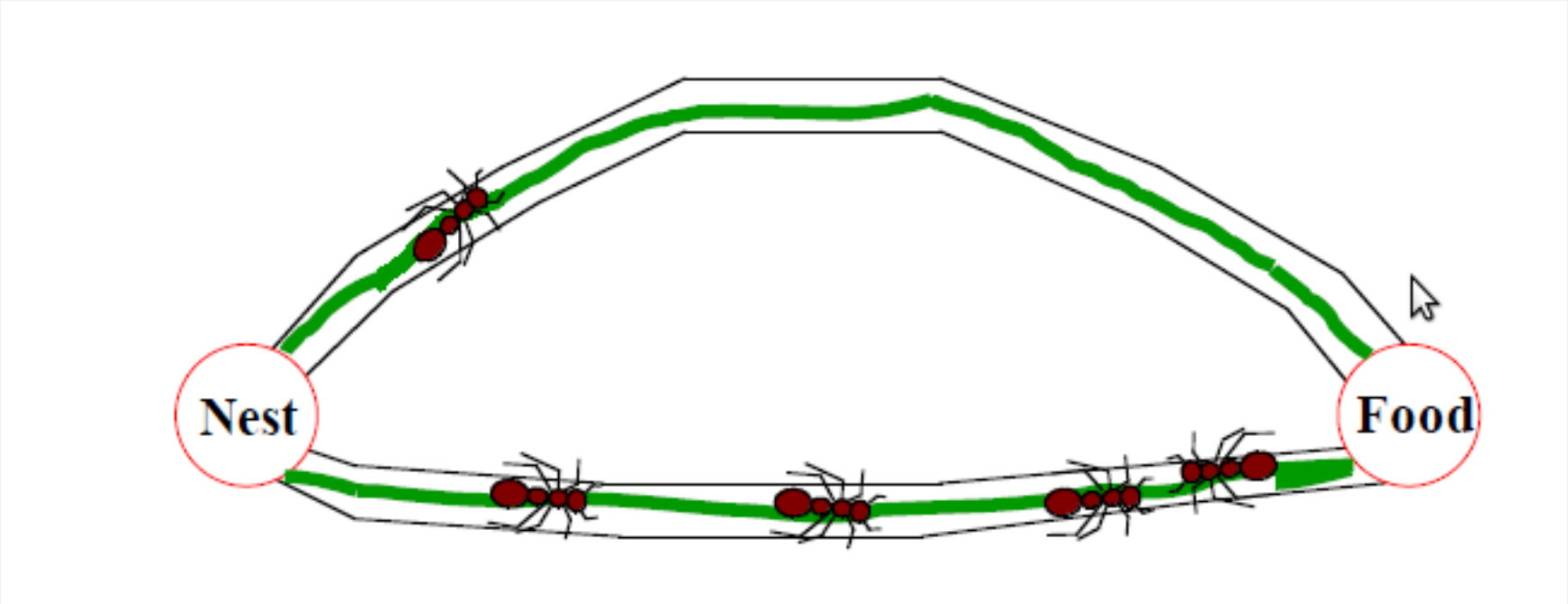

Nest
Food


Nest
Food

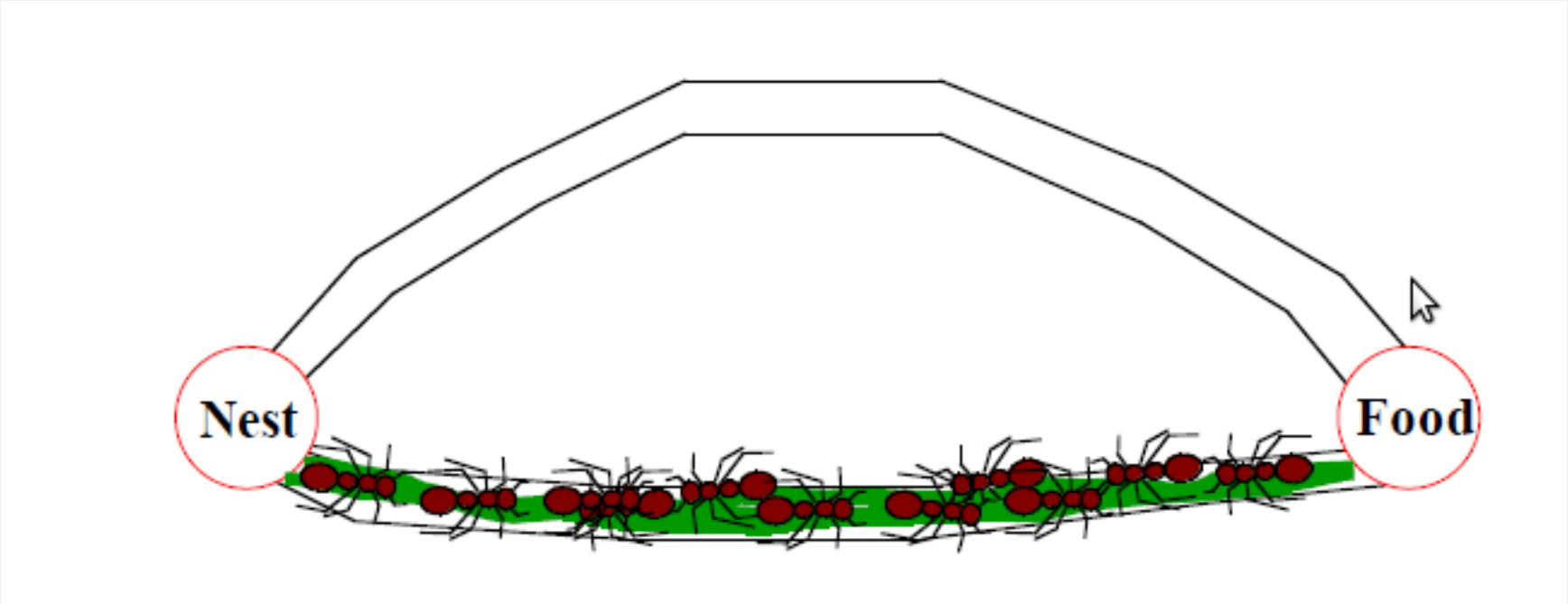

Nest
Food

This is an example of positive feedback! The shorter branch has more pheromone deposited and therefore attracts the new ant. The new ant will also deposit more pheromone, amplifying the shorter branch even more!




Nest
Food

The pheromone deposited on the longer branch evaporates—an example of implicit negative feedback. If the ants forcibly removed the pheromone from that branch then we would say the negative feedback was explicit.



- It is important to note that the ants will usually reach a state of consensus, with the majority adopting one of the two branches. This is true even if both branches are the same length.

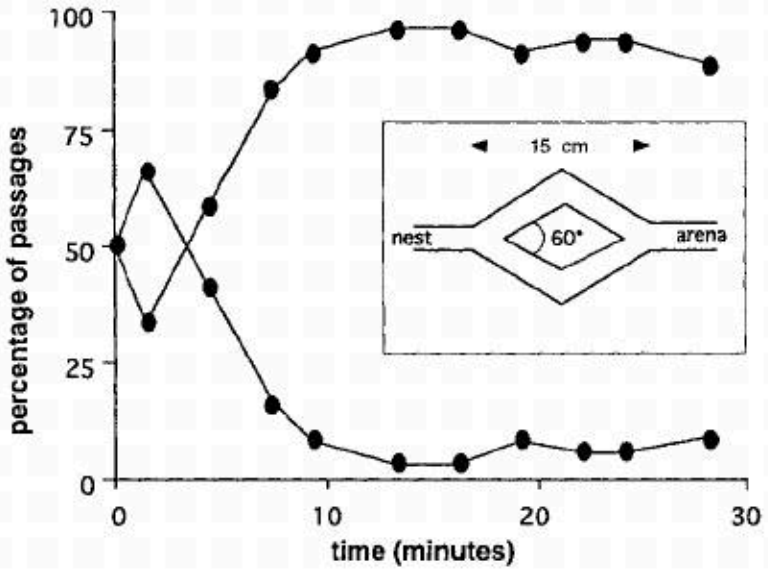
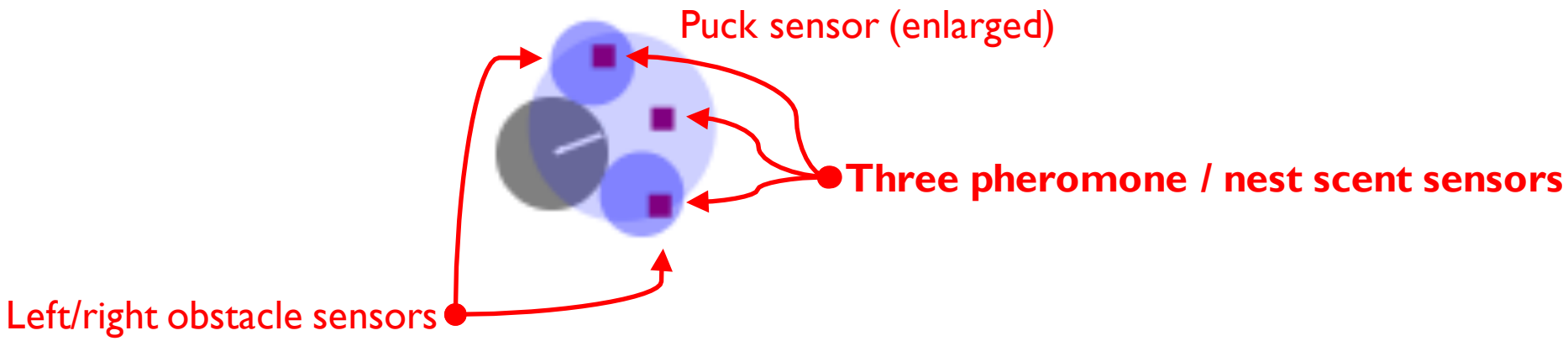


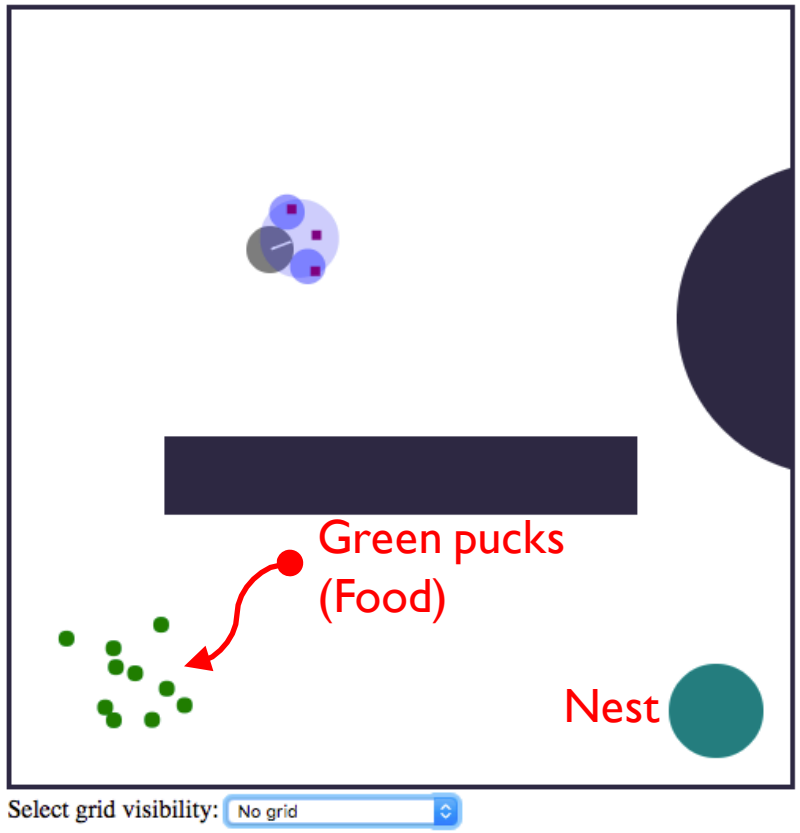
FIGURE 2.1 Percentage of all passages per unit time on each of the two branches as a function of time: one of the branches is eventually used most of the time. Note that the winning branch was not favored by the initial fluctuations, which indicates that these fluctuations were not strong enough to promote exploitation of the other branch. The inset is a schematic representaton of the experimental setup. After Deneubourg et al. [87]. Reprinted by permission © *Plenum Publishing.*

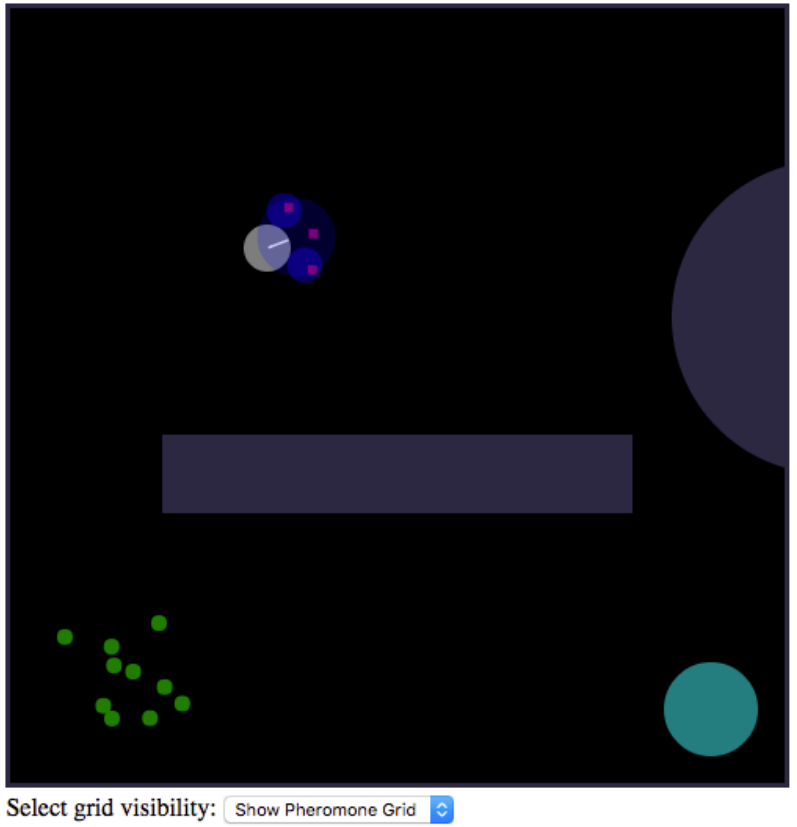
PHEROMONES IN WAGGLE

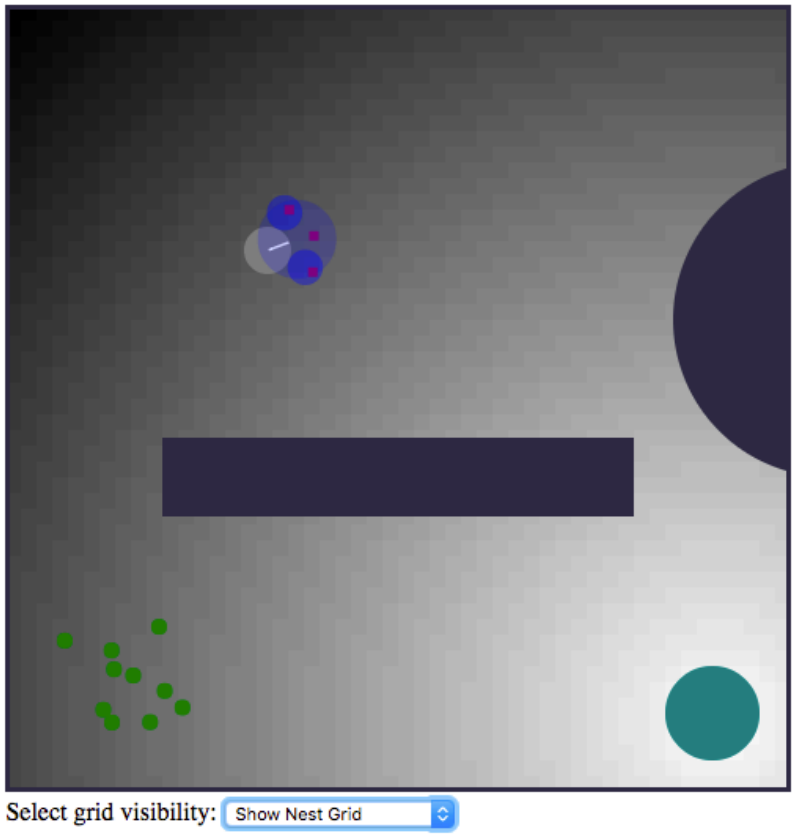
- Go to the main Waggle page (or just hit your browser's back button):
  - http://bots.cs.mun.ca/waggle/
- Select the **Pheromones** level
- The robot has a new layout of sensors:



- The "Select grid visibility" dropdown offers three choices:







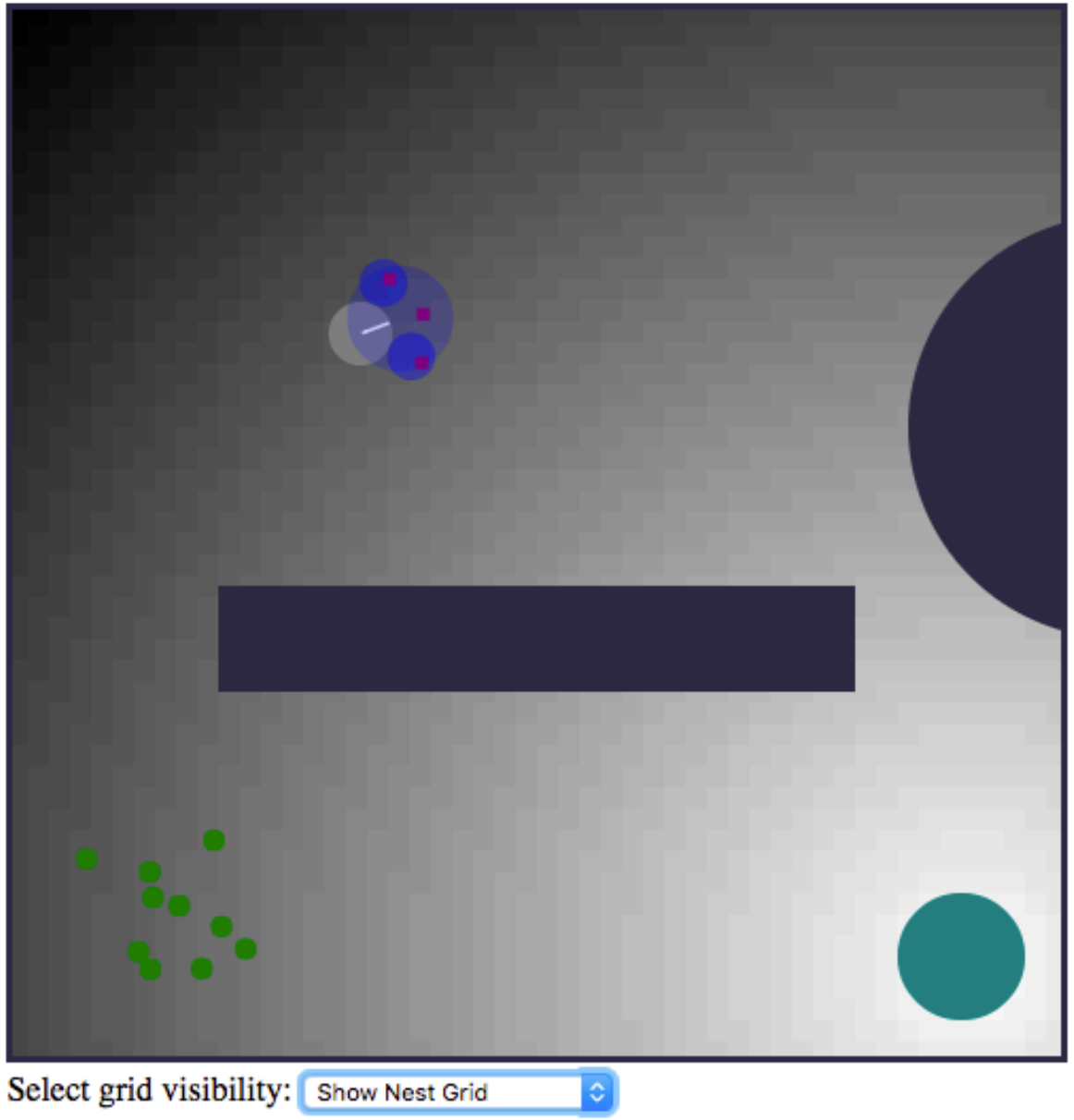
No grid:
- White background just as before

Show Pheromone Grid:
- Displays the quantity of pheromone:
  - 0: Black
  - 1: White
- Appears black initially because no pheromone has been deposited

Show Nest Grid:
- Displays the quantity of nest scent:
  - 0: Black
  - 1: White

NEST SCENT?

- Once grabbing a puck (food) the robot needs to get it back to the nest
- Real ants have a variety of navigation strategies:
  - Visual homing by comparing the current image with one captured at the nest
  - Learning their own visual path
  - Path integration: Integrating velocity over time
- We'll simplify and assume they can somehow smell the nest over a wide area



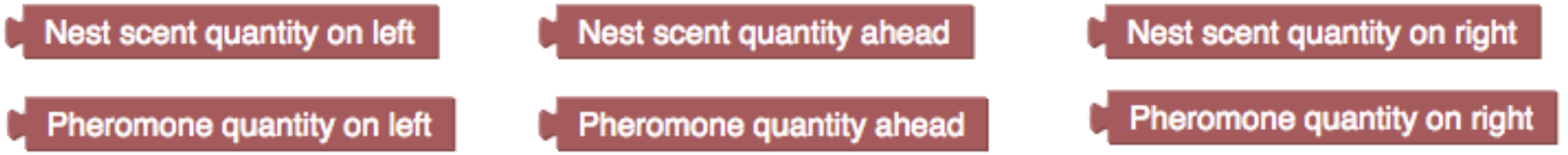
- Each cell has a value in the range [0, 1] corresponding to its distance to the nest:
  - 0 (Black): Cell is at maximum distance to the nest
  - 1 (White): Cell is at the centre of the nest (distance 0)
- So the nest scent grid forms a hill with the nest on top
- This hill can be climbed by always moving in the direction of the highest sensor value

NEW BLOCKS

- In the "Actions" Category:

  - Emit this quantity of pheromones (10 by default)
  - Value is divided by 10 internally

- In the "Sensors" Category:



  - Gives the quantity of either nest scent or pheromone underneath the left, ahead, or right sensor

EXERCISE #10:

HILL CLIMBING

15 MINUTES

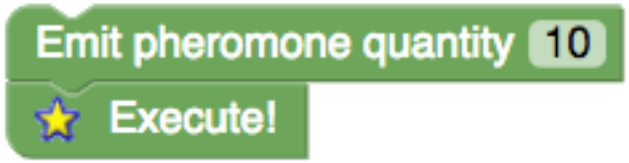
Conditions:

1 robot

- Consider only nest scent, create a controller that moves towards the direction with the greatest nest scent
- Once working, incorporate obstacle avoidance

- CHALLENGE:
  - The robot should be able to reach the nest from anywhere

- Now experiment with this very simple controller by dragging the robot around:



- Our goal is to develop a foraging controller which will leave pheromones behind whenever the robot has collected a green puck and is on its way home
- Try this controller:

- Like the one above, a robot using this controller needs the human user to drag it around. Try using it to collect all of the pucks in each pile:
  - Notice how a new pile is generated

EXERCISE #11:

PHEROMONE-AIDED FORAGING

30 MINUTES

Conditions:

Any number of robots

- Combine the following ideas to develop a controller that continually searches for green pucks using pheromones and returns with pucks to the nest:
  - Controller from previous slide
  - Hill climbing controller (ex. 10) for returning to the nest
  - Obstacle avoidance

- CHALLENGE:
  - 10 robots, 10 pucks: Once the controller is working, determine how long it takes to cycle through all three food piles
  - Contrast this with the same controller which deposits no pheromones

CLOSING THOUGHTS

CONCLUSIONS

- We've approached swarm robotics through biologically-inspired examples
- Waggle's version of Blockly is quite limited---yet interesting and complex collective behaviour still emerges
- Hopefully you've seen that swarm robotics is an engaging and accessible field!
- Please let me know of any suggestions for improving Waggle or the experience of newcomers to swarm robotics:
  - av@mun.ca
  - bots.cs.mun.ca

WAGGLE: KNOWN ISSUES / BUGS

- ”Slots” blocks not working
- When the same controller is re-loaded, the load fails
- Blockly area sizing
- Arrow keys not the best choice for manual control
- Pheromones: Robot gets trapped at nest while holding a puck (occurred with 10 robots, 10 pucks)