Exam I is now scheduled for **Oct. 14**.
I will post a practice exam this weekend.

Labs: You currently have a lab due before the Fall Break (Oct. 7-8).
Due to the Fall Break there will be no labs the week of Oct. 4-8.
Your next lab will be assigned Oct. 13/15.

There is **no class Oct. 7** (despite what the schedule said).

For Tuesday: TCS 9.

Robot Control

The are various approaches to controlling robot behavior. Three principal methods, from simpler to more complex:

(1) Reflexive control
(2) Behavior-based control
(3) Hierarchical control

All of these can be mixed in various ways.

You have already seen an example of reflexive control: Braitenberg vehicles.
In reflexive control, perception is connected directly to some action. For example the sensors directly control the motors.
Complex behavior may emerge as a result of these simple robots acting in a complex environment. Observing them, you might infer more intelligence or purposes than the robots actually have.
Very few animals are purely reflexive, but all animals have some reflexive behaviors.

At the other extreme is hierarchical control, which typically involves "thinking" in some form (e.g., memory, knowledge, plans, cognition, reasoning, and internal representations such as maps, images, etc.).
Hierarchical control is difficult, and it tends to be slow. This is not very good for "low level" sensory-motor activities.

For the middle level we have behavior-based control.

In BBC, a behavior is defined as a tightly coupled perception-action pair. And then the total behavior of the robot is defined as a combination of multiple individual behaviors.

Examples of behaviors: for robots: move forward, avoid obstacles, seek light,…
for animals: seek food, flee predator, ….
These behaviors typically have one well-defined purpose.

We distinguish two types of perceptions: releasers and guides.

A releaser releases or activates a behavior.
For example, the sight of a seed might activate pecking behavior.
The sight of a predator might activate fleeing behavior.
It turns on the behavioral program.

A guide guides (directs, constrains) the behavior once it has been released.
For example, once the flee behavior has been released, then other perceptions will guide the fleeing behavior, such as escape routes, hiding places, etc.

Often the same perception functions as both releaser and guide. For example, the

sight of a seed might release the bird's pecking behavior, but also guide it so it pecks in the correct place.

One advantage of thinking about behavior in this way, is that implementing behavior can often be simplified by using affordances. An affordance is a simple perception that's relevant to releasing or guiding a behavior. For example, when you want to get out, doors and door knobs are affordances that guide your behavior.

So with BBC we break down complex behaviors into simpler ones, but how do we combine them again into the agent's total behavior?
For example, the sight of a seed might be releasing the feeding behavior, at the same time that the sight of a cat is releasing the fleeing behavior. How do you deal with these two conflicting behaviors?

(1) One way of combining is with a priority structure. Fleeing probably has a higher priority than feeding.
(2) Sometimes behaviors can be carried on concurrently (e.g., walking and chewing gum).
(3) Sometimes behaviors can be blended (e.g., turning and braking at the same time).
These are some of the principal ways we can take individual behaviors and combine them.

One idea, from Rodney Brooks (MIT): subsumption architectures.
Lower level behaviors are subsumed (controlled) by higher level ones.
Insect-like robot example. Working upward:
1) Design single leg motion.
2) Coordinated leg motion. Control the individual legs (activating and blocking their behavior) so that at least three legs are bearing weight at any given time.

3) Next level, coordinate leg motion to move forward.
4) Next level, avoid obstacles.
5) Next level, goal seeking or exploration.

General theme: breaking complex systems down into simpler systems, and being aware of how the parts combine into more complex ones.
This allows parts to be tested separately (unit testing). In particular, individual behaviors can be tested on their own, and when they work individually, they can be combined into the compete system.

Loan Program:

This is a nice example of top-down program development.

Pay especial attention to the print() function because this is the way you can control the format of output.

print (<format string>, <expression 1>, <expression 2>, … , <expression n>);

Prints <expression 1> through <expression n> according to the format string.
In the format string, characters are printed as themselves, except for "format codes," which begin with "%". A format code tells the system to output the value of the next expression in accord with the format code.

%W.Df = format a floating-point number in a field of width at least W and with D digits after the decimal point. For example %4.2f will print a number in a field of width at least 4 with 2 digits after the decimal. (Note that the decimal point takes up on the 4 places.) I say "at least" because if the number needs more that W spaces, it will use what it needs.

%Wd = format an integer in a field of width W.

%Ws = format a string in a field of width W.

In all of these, if the characters needed are less than W, then the output is right-justified in a field of width W. To left justify them, use %- instead (i.e., %-W.Df, %-Wd, %-Ws).