

LCC 6310

The Computer as an Expressive Medium

Lecture 23

Overview

Programming concepts

- Braitenberg vehicle reminder

- Implement vehicles

- Possible extensions

Free time to work on P5 and/or A6 at the end of class

Project 5

Due: Friday November 16

In this project, build a collection of simple AI agents that interact with the user, each other, and their ecosystem, to give the illusion of life. You can build upon the provided framework of Braitenberg vehicles, which can produce complex agent behaviors, or code your own simulation.

Provided: Braitenberg starter code

Assignment 6

Posted online, **not graded**

A6-01: Modify the Braitenberg vehicle so that it has a different visual appearance.

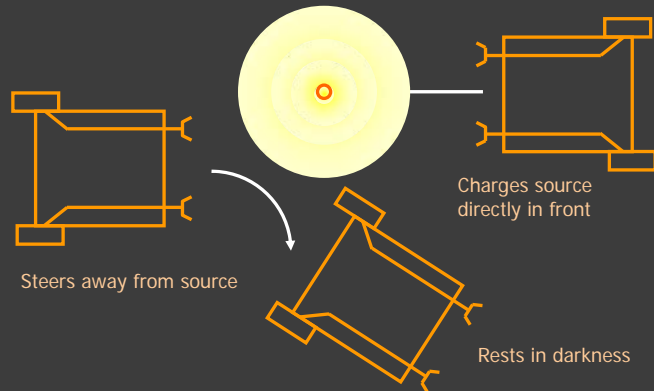
A6-02: Create a vehicle that responds to multiple sense modalities. The best way to do this is to overlay multiple sensory fields, each with its own source types (e.g. sound, light, heat, etc.).

A6-03: Make vehicles also be sources, so that vehicles respond to each other. One way to do this is to place a moving light source on each vehicle. You can choose to make the light source visible, or sum the moving sources into an invisible sensory field (if you don't want glowing circles or some such appearing around vehicles).

A6-04: Have your vehicles interact with the environment in some way. For example, when a vehicle runs into a source, perhaps it destroys the source. Other vehicles could create sources. Vehicles could lay trails that other vehicles respond to. A vehicle could have multiple ways of moving (flying under certain conditions, moving on the ground under other conditions, etc).

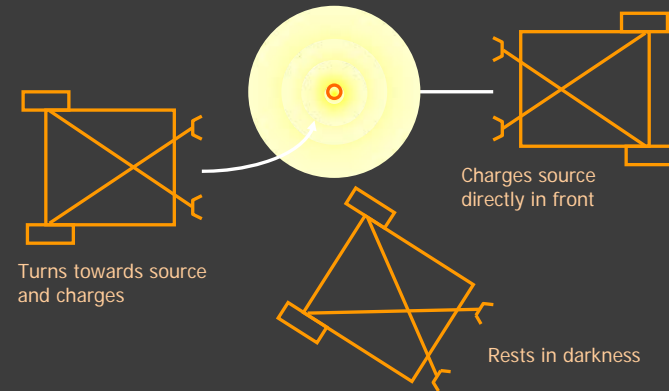
Vehicle 2a: coward

Sensors (light sensors) connected directly to motor on same side



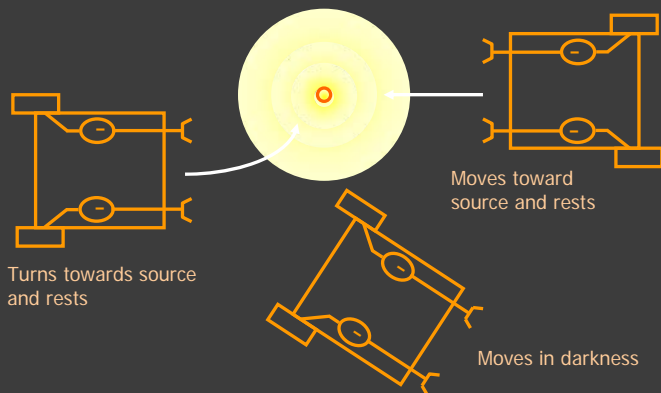
Vehicle 2b: aggressive

Sensors connected directly to motor on opposite side



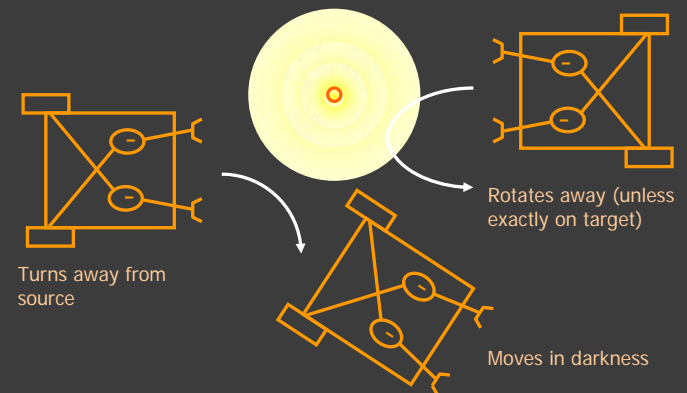
Vehicle 3a: love

Sensors connected through an inverter to motor on same side



Vehicle 3b: explorer

Sensors connected through an inverter to motor on opposite side



Classes in code

Vehicle (1)

Abstract class, provides template for movement logic
Subclassed by vehicles with specific behavior logics
Draws itself and the wheels and the sensors

Wheel (2)

The flapping things on the vehicle

Sensor (3)

The "eyes" on the vehicle – glow indicates activation
Can act as sensors with inverters as well

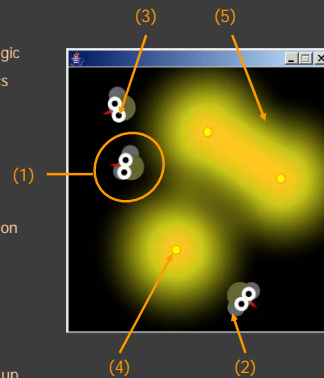
Source (4)

A light source - has a range of influence

SensoryField (5)

A collection of light sources whose influence adds up
Draws itself and all the light sources in the field

Let's see this run in [Processing...](#)



How do we implement all this?

Last week:

We implemented a sensory field of light sources

Wrote code to implement:

Source.java
SensoryField.java

This week:

We'll create the vehicles to move around the sensory field

Write code to implement:

Sensor.java
Wheel.java
Vehicle.java (and VehicleCoward.java, VehicleAggressive.java, etc.)

Concept: abstract classes

Abstract classes let you define some behaviors but force your subclasses to provide others.

This is useful if your class knows that it needs to do certain things, but doesn't know how to do them (because they might be particular to different subclasses for example). The abstract base class can declare these methods to be abstract, and subclasses will need to fill them in.

```
abstract class MyAbstractBaseClass {
  ...
  abstract void myMethod();
  ...
}
class MySubclass extends MyAbstractBaseClass {
  ...
  void myMethod() {
    // fill me in here
  }
  ...
}
```

Vehicles

Vehicles have two sensors and two wheels each

Vehicle is an abstract class

It knows how to do almost everything a vehicle should do: initialize itself, draw itself (body, wheels and sensors), etc.

But it doesn't know how to adjust the wheel velocities based on the sensor input: this is done differently for different kinds of vehicles (coward, aggressive, lover, explorer).

Vehicle provides an abstract `doSenseLogic()` method

Should describe the relationship between sensors and wheels. Different types of vehicles (subclasses) can implement this method in different ways to display different behaviors.

`doSenseLogic()` will set the wheel velocities based on sensor input.

`doSenseLogic()` called by `moveMe()` to actually move the vehicle.

Sensors and Wheels

Sensors know how read values from the surrounding sensory field. They also know how to draw themselves.

Wheels have a radius and angular velocity which determines their displacement (and hence the displacement of their corresponding vehicle).

Let's try to build some of the Vehicle-related classes...

Sensor class

What do we need to store for a Sensor?

What should a Sensor be able to do?

Let's build this class...

Sensor variables

What do we need to store for a Sensor?

```
float x, y; // the sensor position
float sense; // the sensor reading
```

Sensor methods

What should a Sensor be able to do?

```
drawMe() // should know how to draw itself
setLocation() // so that we can move
getSense() // get a sensory reading at the current location
```

Let's fill in some of these methods...

getSense

Get a sensor reading (between 0-1) at the current (x,y) coordinate

Need to pass in the sensory field

Can return an inverse reading if the flag is set to true

```
float getSense(SensoryField sfield, boolean inverse) {  
    float val = sfield.getValue((int)x,(int)y) / 255.0f;  
    sense = (inverse) ? (1-val) : val;  
    return sense;  
}
```

Wheel class

What do we need to store for a Wheel?

What should a Wheel be able to do?

Let's build this class...

Wheel variables

What do we need to store for a Wheel?

```
float x, y; // the wheel position  
float angvel, linvel; // angular and linear velocities  
float radius; // wheel radius  
float ang; // individual wheel angle used for wheel wiggle
```

Wheel methods

What should a Wheel be able to do?

```
drawMe() // should know how to draw itself  
setVelocity() // set the velocity of this wheel
```

Let's fill in some of these methods...

setVelocity

Set angular velocity for this wheel and update linear velocity accordingly

```
void setVelocity(float angvel) {
    this.angvel = angvel;

    // linear velocity updated based on wheel radius and angular velocity
    linvel = angvel * radius;
}
```

Vehicle class

What do we need to store for a Vehicle?

What should a Vehicle be able to do?

Let's build this class...

Vehicle variables

What do we need to store for a Vehicle?

```
float x, y; // vehicle position
float axle, halfaxle, axlesq; // axle dimensions
float angle; // direction of vehicle
float wheeldiff, wheelavg; // difference and average wheel velocities
Wheel wL, wR; // two wheels, left and right
Sensor sL, sR; // two sensors, left and right
```

Vehicle methods

What should a Wheel be able to do?

```
drawMe() // should know how to draw itself with sensors/wheels
moveMe() // do the vehicle movement based on behavior logic
checkCollision() // check for collision with other vehicles
checkBounds() // make sure we're in bounds
setVelocityL() // set the left wheel velocity
setVelocityR() // set the right wheel velocity

doSenseLogic() // abstract method to implement vehicle logic
```

Let's fill in some of these methods...

checkCollision

If the vehicle collides with other vehicles we want to push past them

```
void checkCollision(ArrayList vehicles) {
    float dx, dy, da;
    for (int i=0; i<vehicles.size(); i++) {
        Vehicle bv = (Vehicle)vehicles.get(i);
        if (!bv.equals(this)) { // if it's not us then find the distance to vehicle
            dx = x-bv.x;
            dy = y-bv.y;
            if (dx*dx + dy*dy < axlesq) { // if the other vehicle is within our axle range
                da = (float)Math.atan2(dy,dx); // find the angle between us
                x = x + (float)Math.cos(da) * halfaxle; // shove past the other vehicle
                y = y + (float)Math.cos(da) * halfaxle;
                bv.x = bv.x + (float)Math.cos(da+(float)Math.PI) * halfaxle;
                bv.y = bv.y + (float)Math.sin(da+(float)Math.PI) * halfaxle;
            }
        }
    }
}
```

moveMe

Move the vehicle based on its sense logic. Needs to know the sensor field and dimensions, as well as the other vehicles to check for collisions.

```
void moveMe(ArrayList vehicles, SensoryField sfield, int w, int h) {
    float ang;
    checkBounds(w,h); // make sure we're in bounds
    doSenseLogic(sfield); // update the wheel velocities based on the sense logic

    // move vehicle
    wheeldiff = wL.linvel - wR.linvel; // left and right wheel velocity difference
    angle += wheeldiff / axle; // update angle based on difference of wheel velocities
    wheelavg = (wL.linvel + wR.linvel) / 2; // average overall vehicle velocity
    x += Math.cos(angle) * wheelavg; // update x based on angle and vehicle velocity
    y += Math.sin(angle) * wheelavg; // update y based on angle and vehicle velocity

    checkCollision(vehicles); // check if we've run into another vehicle

    // don't forget to move the wheels and sensors here too!
    ...
}
```

VehicleCoward subclass

Remember that in a coward vehicle the sensor inputs are wired directly to the wheels on the same side...

```
class VehicleCoward extends Vehicle {

    VehicleCoward(PApplet pa, float x, float y, float angle, float axle) {
        super(pa,x,y,angle,axle);
    }

    void doSenseLogic(SensoryField sfield) {
        setVelocityL(sL.getSense(sfield,false));
        setVelocityR(sR.getSense(sfield,false));
    }
}
```

Putting it all together

Let's see how all these pieces come together in [Processing...](#)

Possible extensions

Make vehicles also be sources

Vehicles start following each other, are repelled by each other, etc.

Implement other sources

E.g. sound, smell...

Implement other objects in the world

Add more complex sensor response curves

Add more interactions between different types of vehicles

Remember...

For **Thursday** this week: Theory Readings

Three students: present one reading each

Everyone else: prepare one discussion question for each reading

Video Games and Computer Holding Power - Sherry Turkle (NMR pp.499-514)

The Six Elements and the Causal Relations Among Them & Star Raiders: Dramatic Interaction in a Small World - Brenda Laurel (NMR pp.563-573)

From *Theater of the Oppressed* - Augusto Boal (NMR pp.339-352)