Overview

Assignment 6
Project 5

Programming concepts
- Braitenberg vehicle concept
- Look at some vehicles
- Look through major code sections
- Implement a sensory field
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).
Braitenberg Vehicles

Valentino Braitenberg (http://www.kyb.mpg.de/~braitenb)

Professor and former director of the Max Planck Institute for Biological Cybernetics in Tübingen, Germany

Book: "Vehicles: Experiments in Synthetic Psychology"

Neuro-psychologist interested in how primitive neural structures can give rise to complex behavior

He developed a simple model of robots with sensors and motors to show how complex behavior can arise from simple mechanisms

We're interested in his vehicles as a simple autonomous agent framework to play with

Build ecosystems of interacting agents and sensory sources
Vehicle 1

One sensor (light sensor) connected directly to one motor

What will happen in light areas?

What will happen in dark areas?
Vehicle 1: simple movement

Straight movement

When in light areas it speeds up, in dark areas it slows down - could be described as being "restless" in light places, but "liking" dark places.
Vehicle 2a

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

What will happen when light is to one side?

What will happen when light is directly in front?
Vehicle 2a: coward

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

Steers away from source

Charges source directly in front

Rests in darkness
Vehicle 2b

Sensors connected directly to motor on opposite side

What will happen when light is to one side?

What will happen when light is directly in front?
Vehicle 2b: aggressive

Sensors connected directly to motor on opposite side

Turns towards source and charges

Charges source directly in front

Rests in darkness
Vehicle 3a

Sensors connected through an inverter to motor on same side

What will happen when light is to one side?

What will happen when light is directly in front?
Vehicle 3a: love

Sensors connected through an inverter to motor on same side

Turns towards source and rests

Moves toward source and rests

Moves in darkness
Vehicle 3b

Sensors connected through an inverter to motor on opposite side

What will happen when light is to one side?

What will happen when light is directly in front?
Vehicle 3b: explorer

Sensors connected through an inverter to motor on opposite side

- Turns away from source
- Moves in darkness
- Rotates away (unless exactly on target)
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?

This week:

We'll begin by creating our sensory field of light sources
Write code to implement:

- Source.java
- SensoryField.java

Next week:

We'll create the vehicles to move around the sensory field
Write code to implement:

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

The sensory field is a 2D array of pixels that corresponds to the size of the display. Different light sources change the pixels of the field.

Provides a mechanism for summing all the different sources together

This is the mechanism for summing different sources together

update() sums the light sources to fill the 2D array of background pixels
updateGround() turns the 2D array into a PImage that can be displayed, which is faster than drawing out each pixel individually

Sensing in the vehicles can thus take place by asking the ground for the summed sense value (rather than by directly asking the sources)
Sources

We'll implement a light source

The influence falls off non-linearly farther from the source

**maxrange** determines how far out influence extends

Sources only draw the little circle in the middle

The light gradient around them is actually in the sensory field

**drawrange** toggles drawing of the maxrange circle around sources

**getValue()** is used by the sensory field to sample a light source (to determine how much a light source affects a given pixel)

Let's build a class to represent a Source...
Source class

What do we need to store for a Source?
What should a Source be able to do?

Let's build this class...
Source variables

What do we need to store for a Source?

float x, y;  // the position
float maxrange;  // the radius of influence of this source
boolean grabbed;  // so we can tell if it's being dragged
boolean drawrange;  // so we know whether to draw the range circle
Source methods

What should a Source be able to do?

- `drawMe()` // should know how to draw itself
- `setLocation()` // so that we can move it around after initialization
- `hitTest()` // check if this source has been clicked
- `getValue()` // get a sensory reading at a specific point
- `getInfluence()` // find influence of source at given a distance

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

Check if the source has been clicked at (xhit,yhit)
The source is located at coordinates (x,y)

```java
boolean hitTest(float xhit, float yhit) {
    if ((xhit < x+RAD) && (xhit > x-RAD) && (yhit < y+RAD) && (yhit > y-RAD)) {
        grabbed = true;
    } else {
        grabbed = false;
    }
    return grabbed;
}
```
getValue

Get the sensory value of this source at location (tx,ty)
The inverse influence is returned when positiveInfluence flag is false

```java
float getValue(float tx, float ty, boolean positiveInfluence) {
    float d = distance(tx, ty, x, y);
    if (d >= maxrange) {
        // point is outside the radius of influence
        return ((positiveInfluence) ? 0 : 1);
    }
    // non-linear influence based on proximity to source
    float f = getInfluence(d, maxrange); // non-linear
    return ((positiveInfluence) ? f : 1-f);
}
```
getInfluence

Get the influence of this source at a radius of \( r \) from the source

\( r_{\text{max}} \) is the maximum radius of influence

```java
float getInfluence(float r, float rmax) {
    // get the ratio between 0-1 of how close \( r \) is to the source
    // (0 when \( r \) at range perimeter, 1 when \( r \) is at source center)
    float ratio = (rmax - (float)Math.min(r, rmax)) / rmax;

    // compute the (non-linear) strength (0-1) based on how close we are.
    // -- Note: \( \cos(t) \) ranges from 1 to -1 as \( t \) goes from 0 to PI.
    // -- Want: number from 0-1, so need to scale by 0.5 and shift by 0.5
    // -- Want: increasing strength closer to source, so need to flip
    return (float)(0.5 - 0.5*(Math.cos(ratio*Math.PI)));
}
```

A demonstration of this function in Processing is [here](#)
SensoryField class

What do we need to store for a SensoryField?
What should a SensoryField be able to do?

Let's build this class...
SensoryField variables

What do we need to store for a SensoryField?

```java
float w, h; // its width and height
int blocksize; // the resolution of our sensory field
float[][] field; // 2D array of sensory values
ArrayList sources; // light sources in this sensory field
boolean visible = false; // we want to draw the ground
boolean drawranges = false; // draw the range of each source
PImage ground; // the image corresponding to the field
```
SensoryField methods

What should a SensoryField be able to do?

- `getValue()` // return the sense value at a given pixel
- `addSource()` // add a new source
- `removeSource()` // remove source at a given index
- `getSource()` // get source at a given index
- `numSources()` // return the number of sources in this field
- `getVisible()` // is the ground image being shown?
- `setVisible()` // set the ground image to be shown/hidden
- `drawMe` // draw the ground (if visible) and all the sources
- `update()` // update the sensory field values if something has changed
  // (e.g. source added/removed, source moved)
- `updateGround()` // create a new ground image if something has changed
  // and it is currently visible
- `updateAll()` // update the sensory field values and the ground image
  // if it is currently visible

Let's fill in some of these methods...
Draw the ground image (if visible) and all the sources in this field

Since we're creating a separate Java class, we need to take in the parent PApplet as a handle on the drawing canvas

```java
drawMe(PApplet pa) {
    // if the ground is visible then draw the ground image
    if (visible && ground != null) {
        pa.image(ground, 0, 0);
    }
    // draw all the sources
    for (int i=0; i<sources.size(); i++) {
        Source s = (Source)sources.get(i);
        s.drawMe(pa);
    }
}
```
Update all the values in the sensory field if something has changed

```java
void update() {
    float sum;

    // compute the sensory value for each block, and for each pixel within each block
    for (int i=0; i<w; i+=blocksize) {
        for (int j=0; j<h; j+=blocksize) {
            sum = 0;
            for (int s=0; s<sources.size(); s++) {
                // first sum the sensory values from each source at this spot
                sum += this.getSource(s).getValue(i,j,true);
                // now use that value to fill in all the pixels of this block (up to 255)
                for (int p=0; p<blocksize; p++) {
                    for (int q=0; q<blocksize; q++) {
                        field[i+p][j+q] = (float)Math.min(sum*255,255);  // don't exceed 255
                    }
                }
            }
        }
    }
}
```
Create a PImage to display the sensory values (need the PApplet to do this). Using a PImage is faster than drawing out the pixels every time, since we can draw the same image to screen until the sources change in some way (e.g. light source added/removed, light source moved).

```java
void updateGround(PApplet pa) {
    if (sources.size() == 0) {
        ground = null;
    } else {
        ground = new PImage(w,h);
        for (int i=0; i<w; i++) {
            for (int j=0; j<h; j++) {
                int v = (int)(getValue(i,j));
                // darker yellow where sense value is stronger
                ground.set(i,j,pa.color(v, (int)Math.min(200,v), v/8));
            }
        }
    }
}
```
Putting it all together

Let's see how all these pieces come together in Processing...
Sensory field extensions

In the long run, the sensory field would ideally be modified to handle multiple types of sources.

There are different ways to do this:

- Separate by color (r, g, b), but then you can only have three.
- Better approach: lay multiple sensory fields on top of each other.
  - Transparency will make them all visible.
  - Different sensor types will look at different grounds (sound, light sources, etc.).
The field of artificial intelligence (AI) builds computational systems that model the intelligent behavior of people and animals. AI architectures can be extremely generative, able to produce complex responses to environmental changes, including user interaction. In computational art, AI approaches have been used to build work ranging from robotic sculpture, to drawing and painting generators, from generative interactive stories to music composition. In the popular art form of computer games, AI approaches are used extensively to build tactical and strategic opponents, non-player characters and player modeling systems. 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

Examples:

* Emergence
* Autopoiesis
* Blockies
* A-Volve
Remember...

For **Thursday** this week: Theory Readings

- Two students: present one reading each
- Everyone else: prepare one discussion question for each reading

From *Plans and Situated Actions* - Lucy Suchman (NMR pp.599-612)

*Expressive AI: A hybrid art and science practice* - Michael Mateas *(Online)*