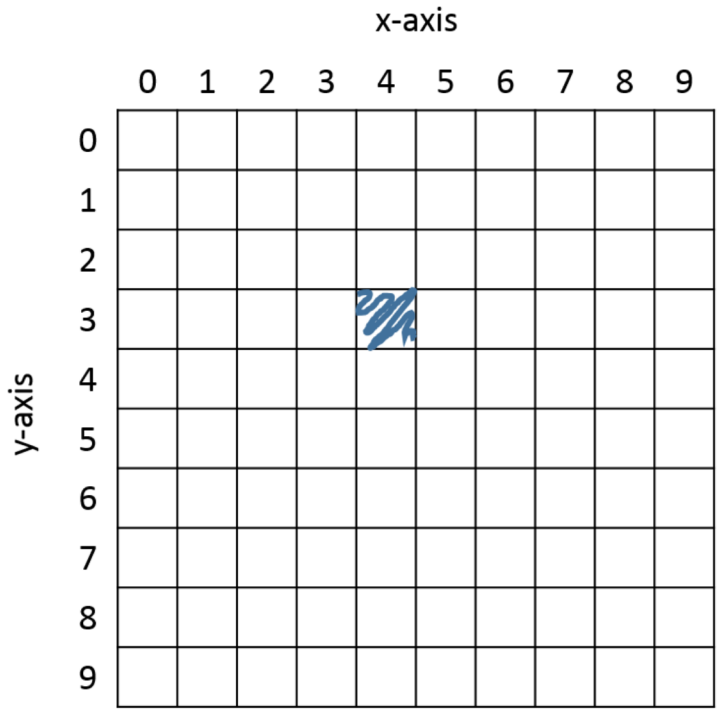
COMP 202 Review with robots on Processing

Pixel coordinate system

Going back to Assignment 2

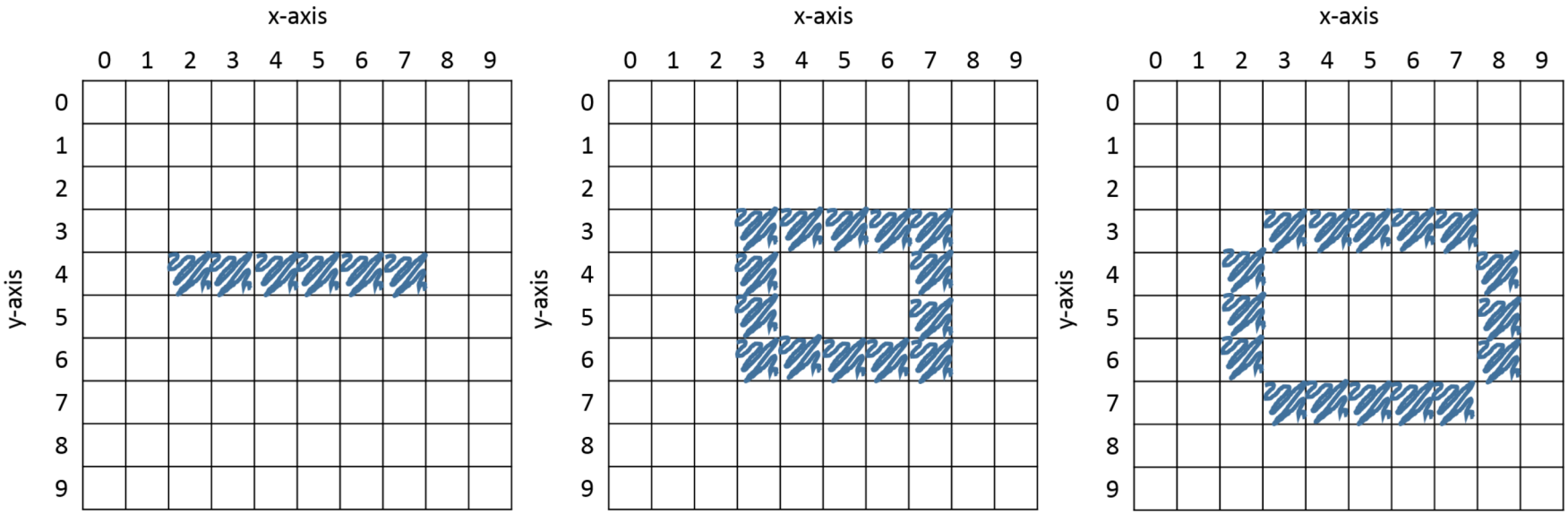


```
1    // setting the size of our canvas
2    size(10, 10);
3
4    // drawing a single point
5    point(4,3);
6
```

`point(x,y)` draws a point centered at pixel coordinates (x,y)

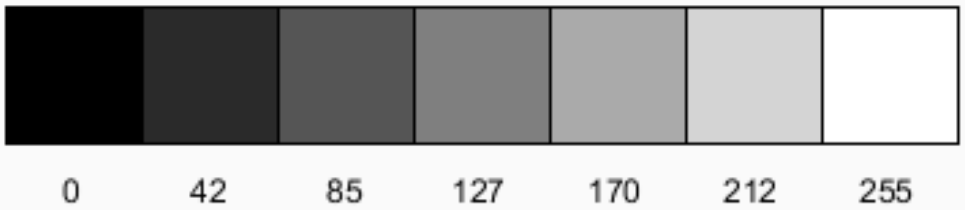
`strokeWeight(pixels)` lets you set the size of the point

Shapes



- `line(x1,y1,x2,y2)` draws a line segment from (x1,y1) to (x2,y2)
- `rect(x,y,w,h)` draws a rectangle whose top left corner is at (x,y) and its size is w by h pixels
- `ellipse(x,y,w,h)` draws an ellipse centered at (x,y) with a size of w by h pixels

Color



- `stroke(val)` sets the color of lines to `val` in grayscale
- `fill(val)` sets the color inside a shape to `val` in grayscale
- `background(val)` sets the color of the background to `val` in grayscale
- `stroke(r,g,b)` sets the color of lines to `r,g,b` in the red-green-blue scale
- `fill(r,g,b)` sets the color inside a shape to `r,g,b` in the red-green-blue scale
- `background(r,g,b)` sets the color of the background to `r,g,b` in red-green-blue scale

Interaction

Making things animated and interactive

- The `setup` method specifies what will happen *at the beginning of the program, and will be executed only once*
- The `draw` method specifies what will happen *continuously, during your program's execution*. This is known as the *drawing loop*.

```
    // here we define global variables
    int wid = 600;
    int hei = 600;

    int x = wid;
    int y = hei;

    void setup(){
        // this only gets executed once, at the beginning of your program
        size(wid,hei);
    }

    void draw(){
        // this sets the background color
        background(100);

        // this updates the position of the ellipse for the next frame
        x+=10;
        y=x*x/600;
        if (x >= 650){
            x = 0;
        }

        // this gets executed continuously
        ellipse(x,y,100,100);
    }
```

The speed of the drawing loop can be set with the `frameRate(fps)`, where `fps` is the desired number of *frames per second*

Interaction

Making things animated and interactive

Mouse input:

- The `mousePressed` global variable is a boolean that is set to `true` when the user clicks a mouse button. Before every call to the `draw` method it is set to `false` by default.
- The global variables `mouseX` and `mouseY` give you the *pixel coordintates of the mouse pointer* in the canvas. These are of type `int`.

Keyboard input (an interactive to what the Scanner class does):

- The `keyPressed` method gets executed whenever the user presses a key
- You can get the ASCII value of the key that was pressed using the global variable `key`, of type `char`

Pausing the program execution:

- The `delay` method takes a `double` respresenting a time in seconds to pause the program

Using Objects and Classes in Processing

Processing is based on Java, so we can use all of the things we learned during the COMP 202. Let's create a program that draws robots with the following rules:

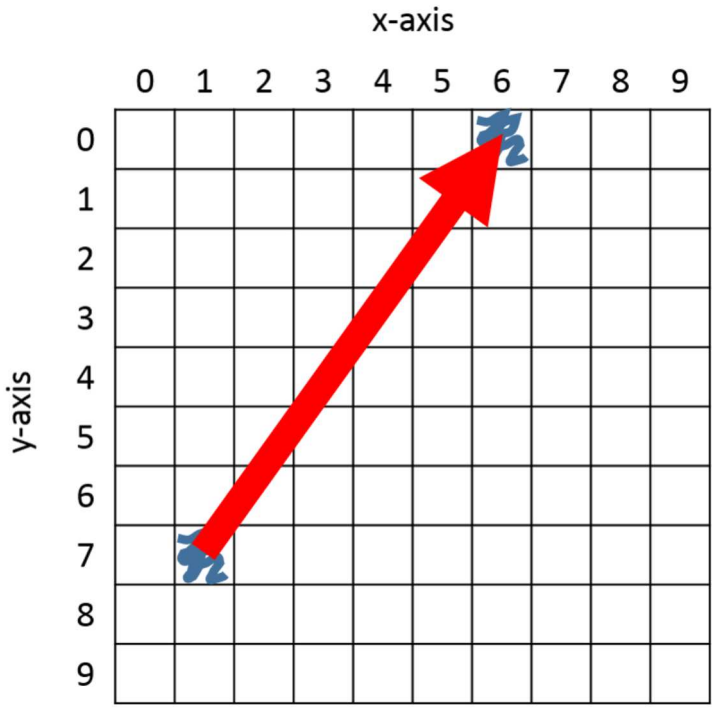
- A `Robot` should be drawn as an ellipse. It has a position on the screen in pixel coordinates (`x`,`y`), and a `diameter` attribute. It will also have a `speed` attribute (in pixels per second).
- Every time the user clicks on the window, a new `Robot` will be drawn at the location of the click.
- Every time the user presses x, a `Robot` will be selected at random, and killed.
- `Robot` instances are scared of the last rule, so they should be *shaking* around their current location

Some additional rules

- All the `Robot` instances are `attracted` towards the location of the mouse pointer.
- `Robot` instances hate each other: add a `repelling` force between `Robot` instances that keeps them apart.

Robot velocities, attraction and repulsion forces

All these have a magnitude and direction



- Let's modify the Point2D class from Assignment 3, to create a `Vector2D` class
- A vector has `x` and `y` components
- Since we want to accumulate forces, and velocities, the vector should have an `add` method.

- We will find useful to compute the `angle` and `magnitude` of a vector.
- We will find useful to `multiply` a vector by a number to change it's magnitude.

Robot velocities, attraction and repulsion forces

Let's modify the Robot class by adding the following attributes:

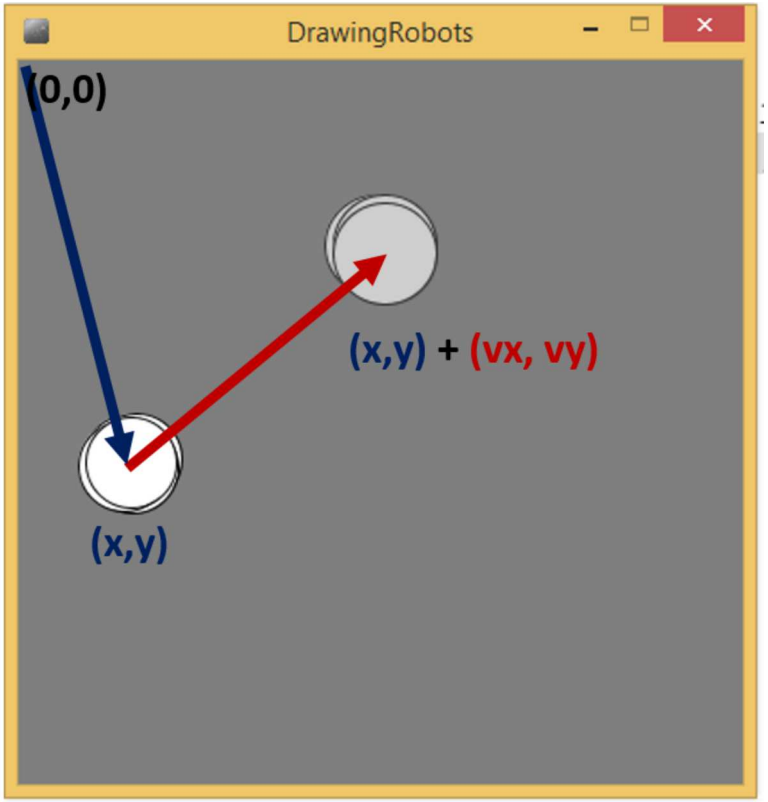
- The `location` vector will represent the Robot's position
- The `velocity` vector will control the direction in which the robot will move in the next frame.
- The `acceleration` vector will control how the velocity changes, depending on external forces.

And the following methods:

- The `update` method will change the Robot's location, using its current velocity.
- The `applyForce` method will accumulate external forces that we apply to each robot. This will change the Robot's acceleration.

Robot velocities, attraction and repulsion forces

The kinematics of our robot world (the update method)

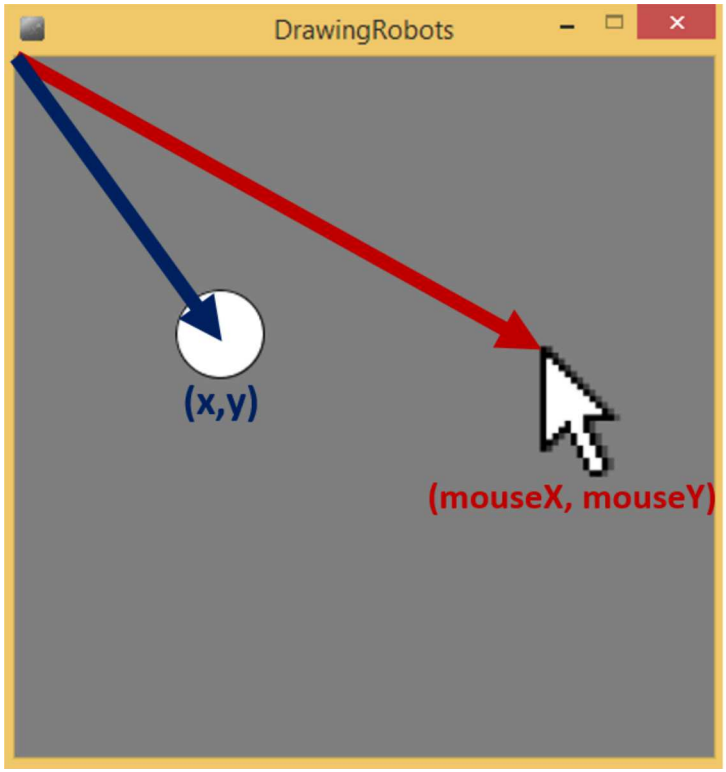


Using the vectors for the Robot and mouse pointer position, we will modify the Robot's velocity

- The `update` method will change the Robot's location, using its current velocity. It will also update it's current velocity using using its current acceleration

Robot velocities, attraction and repulsion forces

The dynamics of our robot world (the applyForce method)

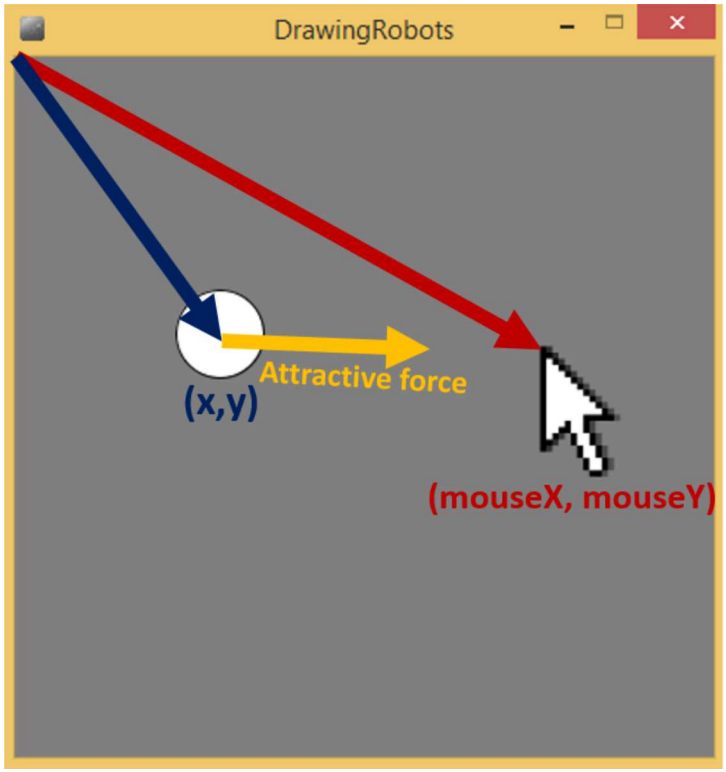


Using the vectors for the Robot and mouse pointer position, we will modify the Robot's velocity

- The `update` method will change the Robot's location, using its current velocity. It will also update it's current velocity using using its current acceleration
- The `applyForce` method will accumulate external forces that we apply to each robot. This will change the Robot's acceleration.

Robot velocities, attraction and repulsion forces

The dynamics of our robot world (the applyForce method)

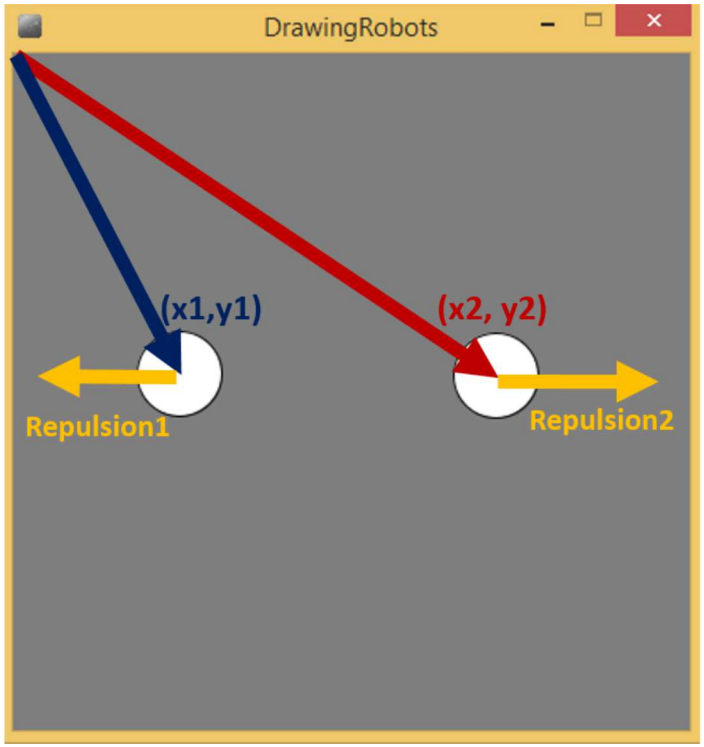


Using the vectors for the Robot and mouse pointer position, we will modify the Robot's velocity

- The `update` method will change the Robot's location, using its current velocity. It will also update it's current velocity using using its current acceleration
- The `applyForce` method will accumulate external forces that we apply to each robot. This will change the Robot's acceleration.

Robot velocities, attraction and repulsion forces

The dynamics of our robot world (the applyForce method)



Using the vectors for the positions of every Robot, we will modify each Robot's velocity

- The `update` method will change the Robot's location, using its current velocity. It will also update it's current velocity using using its current acceleration
- The `applyForce` method will accumulate external forces that we apply to each robot. This will change the Robot's acceleration.

Saving a file of our current robot world

We want to store the `location` and `velocity` of every robot, so that we can reload it when we start the program:

- We need to implement the `toString` method of the `Vector2D` and `Robot` classes
- We need to implement a `loadFromString` method for the `Robot` class
- We will implement a method `loadState` that will try opening a file to load the state of the robot world
- We will implement a method `saveState` that will open a file with the robot world state, and recreate it.
- We will use the `keyPressed` method to save the world state, when the user presses the 'S' key.

Next class: Final Exam Review

Resources

- Processing:
  http://processing.org
- **Very** simple Processing Tutorial:
  http://hello.processing.org
- A book with **far more advanced** programming concepts on Processing:
  http://natureofcode.com/book/

Drawing Tools