POLITECNICO
MILANO 1863

IMAGE AND SOUND
ISPG
PROCESSING GROUP

CREATIVE PROGRAMMING AND COMPUTING

Lab: MultiAgent systems

MULTIAGENT SYSTEM

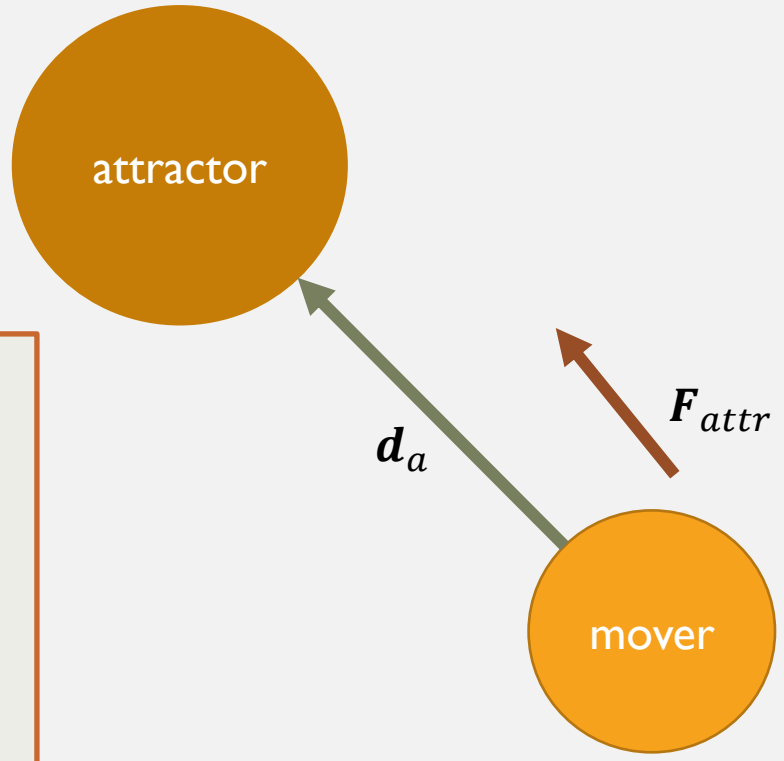
- We will see different multiagent systems based on their type of interaction
  - No Interaction
  - Personal Goal
  - Collective Goal

- First we will adapt some of the agents we have seen in the previous lesson to the multiagent system (without interaction).

GRAVITY AND ATTRACTION

GRAVITY AND ATTRACTION

- We will start from moving_ball_attractor → moving_balls_attractor
  - Remove the OSC communication
- Modify the script to create several movers
- Suggestions:




```
# moving_balls_attractor.pde
int N_AGENTS=20; AgentMover[] movers;
void setup(){
  movers=new AgentMover[N_AGENTS];
  for(int i=0; i<N_AGENTS; i++){
     movers[i]=new AgentMover(/*place a random mass*/);
  }
  /*...*/}
void draw(){
  /* your code */  }}
```

GRAVITY AND ATTRACTION

- Let's sonify this script using Processing Audio Library
- Idea: each Agent is connected with a sound
  - map the sound volume to the Agent's distance from the attractor

Preparation:

1. Install Sound Library in Processing
2. Create a folder «sounds» inside «moving_balls_attr_sounds»
3. Download the audio files from https://tinyurl.com/cpac2021sounds and place them in «sounds»
4. Keep the documentation for SoundFile open https://processing.org/reference/libraries/sound/SoundFile.html

GRAVITY AND ATTRACTION

- We map the sound volume of a mover to its distance from the attractor

$$A_i = \left\lfloor \frac{1}{1 + \alpha\, d_i} \right\rfloor_{A_{min}}$$

$\lfloor x \rfloor_y$ = `max(x,y)`

- A_i the amplitude of the i-th samples,
- d_i the distance from the i-th mover to the attractor,
- α a constant to rescale (I choose 0.01)
- A_{min} the minimum allowed amplitude, so that samples are always audible (I choose 0.02)
- What happens when $d_i = 0$? And what when $d_i \to \infty$?

GRAVITY AND ATTRACTION

Import audio library and utils

```
# moving_balls_attr_sounds.pde
import processing.sound.*;
import java.util.Date;
int N_AGENTS;
AgentMover[] movers;
SoundFile[] samples;
void setup(){
  String P=sketchPath()+"/sounds";
  File dir = new File(P);
  String fns[] = dir.list();
  N_AGENTS=filenames.length;
  movers=new AgentMover[N_AGENTS];
  samples=new SoundFile[N_AGENTS];
  for(int i=0; i<N_AGENTS; i++){
     movers[i]=new AgentMover(random(100,200));
     samples[i]=new SoundFile(this, P+"/"+fns[i]);
     samples[i].amp(0); samples[i].loop();
  }}
```

Get all the namefiles in the directory "sounds"

Create a mover for each audiofile, setting the volume to zero and play it on loop

```
void changeAmp(i){
  /*your code here*/
}

void draw(){
  /* draw attractor in the
  middle of the screen */
  PVector force_a;
  for(int i=0; i<N_AGENTS; i++){
    force_a =
           computeGravityForce(movers[i]);
    movers[i].applyForce(force_a);
    changeAmp(i);
    movers[i].update();
    movers[i].draw();
  }
}
```

GRAVITY AND ATTRACTION

Challenges / Extensions

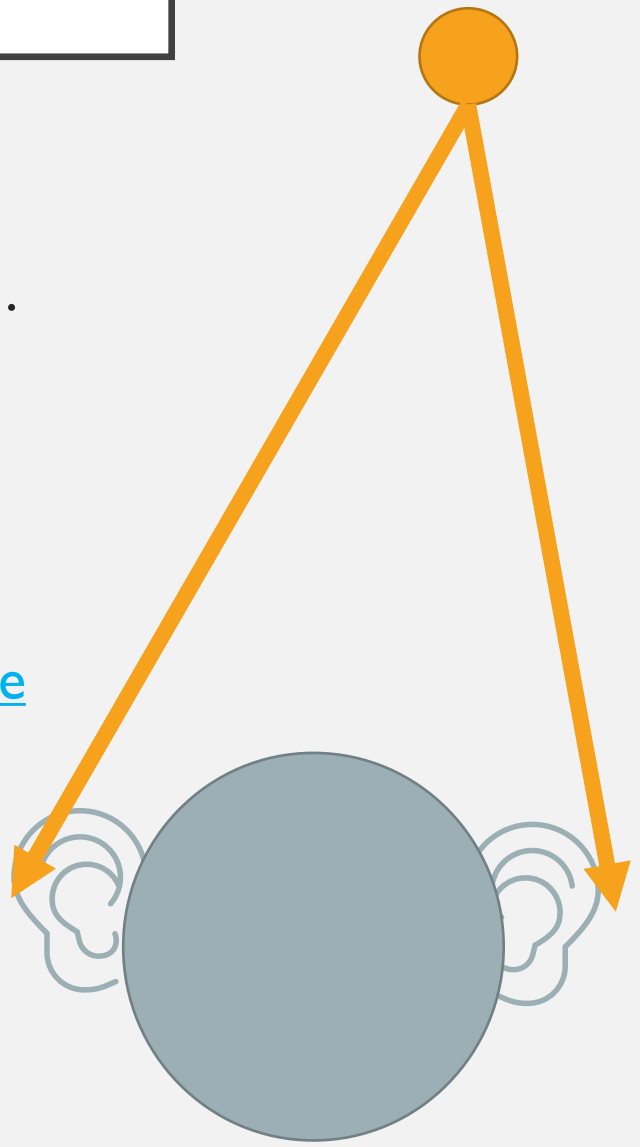
1. Slow down the system: how? (which parameters affect the speed?)
2. Play with the animation: change colors, shapes, background dynamically following the movement of the AgentMover
3. Sound changes with Movers movement; change Mover's shape with sound's features

GRAVITY AND ATTRACTION

Personal project

- Suppose the attractor is our head, and the 2D plan is the space around us…
- we can make amplitude change with the distances from our «ears»
- Use amplitude and pan to create a 3D effect
- That was my personal project for working in an open space

[mbuccoli/binaural_soundscape: A small project to dynamically generate binaural landscape (github.com)]



PARTICLE SYSTEMS

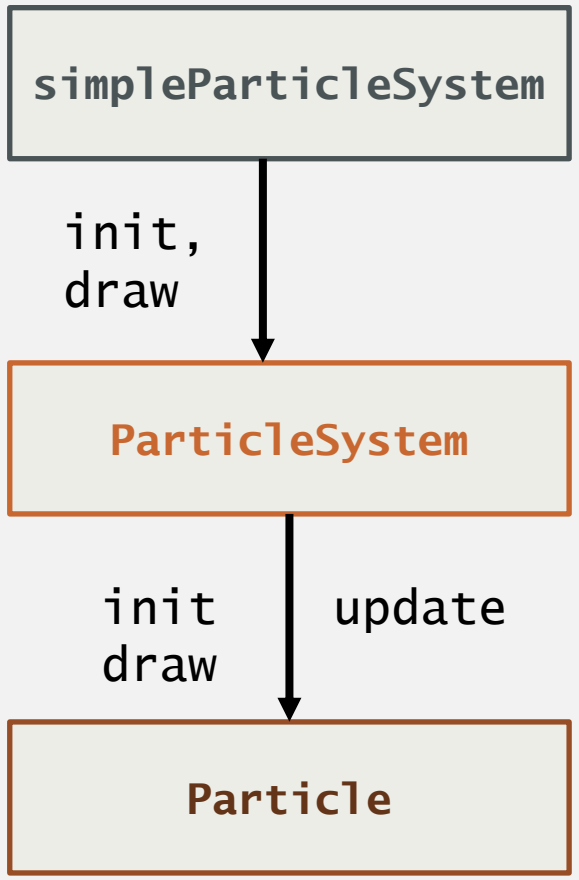
PARTICLE SYSTEM

- *"A particle system is a collection of many many minute particles that together represent a fuzzy object. Over a period of time, particles are generated into a system, move and change from within the system, and die from the system."*
  —William Reeves, "Particle Systems—A Technique for Modeling a Class of Fuzzy Objects," ACM Transactions on Graphics *2:2 (April 1983), 92.*
- We define a Particle System as a… system of particles
  - Each particle move with a certain behavior, both rule-based and randomic
  - The system is the Agent that collects and acts on all of them
- Let's first create a single Particle, not very differently from our mover

PARTICLE SYSTEM

Our architecture is composed by:

- The main script simpleParticleSystem.pde, which creates a Particle System
- A class ParticleSystem that creates, updates and controls an array of Particles
- A class Particle representing each object



PARTICLE SYSTEM

- Particle is like a mover
- Neglect the mass
- add a *lifespan* attribute
  - It starts with a given value and decreases at each update
  - When it gets to 0, it means the particle is dead, and it should be removed
  - Lifespan can be mapped into alpha, to make the particle fade out with time

```
# Particle.pde
class Particle{
  PVector loc, vel, acc;
  float radius, lifespan;
  AgentMover(PVector pos, float r,
                          float ls){
    this.pos= pos.copy();
    this.vel = new PVector();
    this.acc = new PVector();
    this.radius=r;
    this.lifespan=ls}
  void update(){
    this.vel.add(this.acc);
    this.loc.add(this.vel);
    this.acc.mult(0);}
  void applyForce(PVector force){
    this.acceleration.add(force);}
  void draw(){
    /* draw */ }
}
```

PARTICLE SYSTEM

- Now we create a class for ParticleSystem that organizes particles
  - We will make use of ArrayList and function overloading
- ArrayList are an advanced type in Java that support easy adding, removing and iteration
  - particles.add(), particles.get(int i), particles.remove(int i), particles.size()
- Function overloading means to define a function several times with different parameters
  - In the example, either an origin is specified, or it is automatically defined as the middle of the screen

```
# ParticleSystem.pde
class ParticleSystem{
  ArrayList<Particle> particles;
  PVector origin;
  ParticleSystem(){
    this.particles = new ArrayList<Particle>();
    this.origin=new PVector(width/2, height/2);
  }
  ParticleSystem(PVector origin){
    this.particles = new ArrayList<Particle>();
    this.origin=origin.copy();
  }
```

PARTICLE SYSTEM

```
# ParticleSystem.pde
class ParticleSystem{
  ArrayList<Particle> particles; PVector origin;
  ParticleSystem(){
    this.particles = new ArrayList<Particle>(); // here we store the particles
    this.origin=new PVector(width/2, height/2); // this is the origin of the system
  }
  ParticleSystem(PVector origin){ /*see prev. slide*/}
  void addParticle(){
    this.particles.add(new Particle(this.origin, 10, random(0,255)));}
  void draw(){
    for(int i=this.particles.size()-1; i>=0; i--){
      Particle p=this.particles.get(i);
      /* your code */
      p.draw(); p.lifespan-=0.3;
      if(p.isDead()){particles.remove(i); this.addParticle();}
    }
  }
}
```

PARTICLE SYSTEM

```
# ParticleSystem.pde
class ParticleSystem{
  ArrayList<Particle> particles; PVector or
  ParticleSystem(){
    this.particles = new ArrayList<Particle
    this.origin=new PVector(width/2, heigh
  }
  ParticleSystem(PVector origin){ /*see
  void addParticle(){
    this.particles.add(new Particle(th
  void draw(){
    for(int i=this.particles.size()-1; i>=0; i--){
      Particle p=this.particles.get(i);
      /* your code */
      p.draw(); p.lifespan-=0.3;
      if(p.isDead()){particles.remove(i); this.addParticle();}
    }
  }
}
```

Why are we loop from the end to the start?

We are dynamically changing the size of the ArrayList:

- removing particles when their lifespan has expired
- adding new particles **at the end of the array**

Reading from the end ensures we read:

- everything
- just once.

PARTICLE SYSTEM

```
# ParticleSystem.pde
class ParticleSystem{
  ArrayList<Particle> particles; PVector
  ParticleSystem(){
    this.particles = new ArrayList<Partic
    this.origin=new PVector(width/2, heig
  }
  ParticleSystem(PVector origin){ /*see p
  void addParticle(){
    this.particles.add(new Particle(this.
  void draw(){
    for(int i=this.particles.size()-1;
      Particle p=this.particles.get(i
      /* your code */
      p.draw(); p.lifespan-=0.3;
      if(p.isDead()){particles.remove(i);
    }
  }
}
```

How to control lifespan?

- 255 is the maximum value → map to alpha
- We remove a value **z** at each iteration → 0.3

Each particles stays alive for (at maximum):

- 255/0.3 steps ~ 850 steps
- Framerate: 60 Hz
- Steps / framerate: 255/60 =14.17 seconds

Each particle lives at most for 14.17 seconds

By controlling **z** you can control the maximum stepsize and the fadeway speed

PARTICLE SYSTEM

```
# ParticleSystem.pde
class ParticleSystem{
  ArrayList<Particle> particles; PVector
  ParticleSystem(){
    this.particles = new ArrayList<Partic
    this.origin=new PVector(width/2, heig
  }
  ParticleSystem(PVector origin){ /*see p
  void addParticle(){
    this.particles.add(new Particle(th
  void draw(){
    for(int i=this.particles.si
      Particle p=this.partic      et(i);
      /* your code */
      p.draw(); p.lifespan-=0.3;
      if(p.isDead()){particles.remove(i); this.addParticle();}
    }
  }
}
```

When is a particle *dead*?
When its lifespan is lower than 0

What do we do when this happen?
Remove the current particle and add a new one

How do we do it?
We must implement a method
boolean isDead()
Returning whether a Particle is dead

What you need to do:

1. Apply a random (small) acceleration to each particle at update
2. Implement isDead() method in the particle class
3. Use lifespan as the alpha value for each particle

```
# ParticleSystem.pde
class ParticleSystem{
  ArrayList<Particle> parti                    o
  ParticleSystem(){
    this.particles = new Ar                   ticl
    this.origin=new PVector                   heigh
  }
  ParticleSystem(PVector or                  /*see pr
  void addParticle(){
    this.particles.add(new                    cle(this.o
  void draw(){
    for(int i=this.particle                   size()-1; i>=
      Particle p=this.parti                   les.get(i);
      /* your code */
      p.draw(); p.lifespan-=0.3;
      if(p.isDead()){particles.remove(i);
    }
  }
}
```

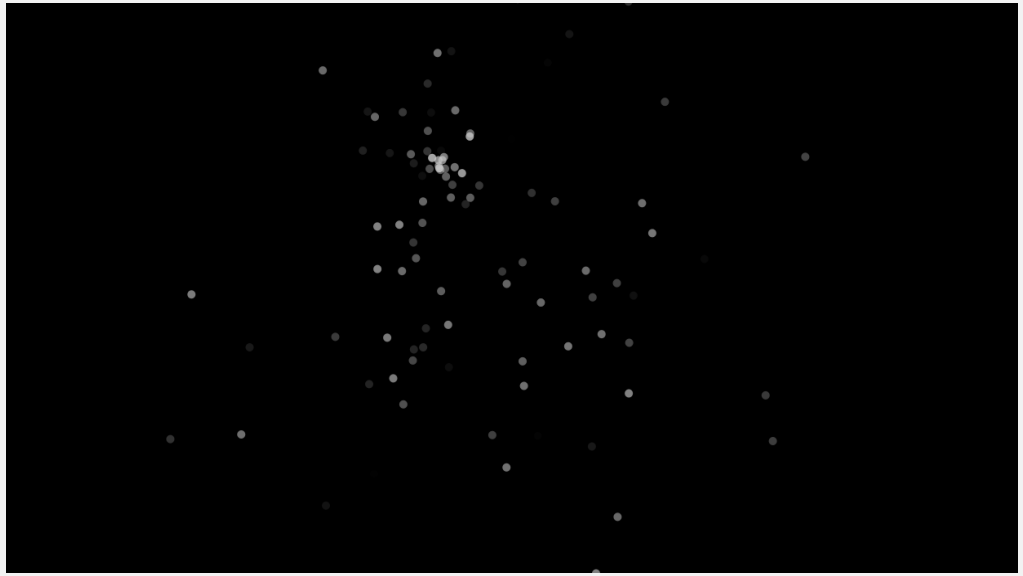
```
# simpleParticleSystem.pde
ParticleSystem ps;
int Nparticles=100;
void setup(){
  size(1280,720);
  ps=new ParticleSystem();
  for(int p=0; p<Nparticles; p++){
    ps.addParticle();
  }
  background(0);
}

void draw(){
  background(0);
  ps.origin=new PVector(mouseX, mouseY);

  ps.draw();
}
```

The origin of the ParticleSystem follows the mouse

PARTICLE SYSTEM

What if you change color?

Play with it changing parameters



PARTICLE SYSTEM + TEXTURES

- Particle systems are a useful way to handle multiple agents under one system
- Plotting circles or other shape is nice, but very limiting
- **Particles are extremely effective when combined with textures**

- Even by just replacing circles with fuzzy-edge circles is a great evolution

- **How do we change our code for using textures?**

PARTICLE SYSTEM + TEXTURES

How do we change our code for using textures?

```
# simpleParticleSystem.pde
//...

void setup(){
  size(1280,720);
  // ...
}
void draw(){
  background(0);
   // ...}
```

- In main script:
  - We add the texture in an image
  - Use additive Blender to add layers with each other → glowing effect

```
# textureParticleSystem.pde
// ...
PImage img;
void setup(){
  size(1280,720, P2D);
  img=loadImage("texture.png");
   // ...
}
void draw(){
  blendMode(ADD);
   // ...}
```

```
# Particle.pde
class Particle{
  // ...
  void action(){
    this.planning();
    fill(255, this.ls);
    ellipse(/*...*/); }
}
```

- In Particle:
  - We just render an image instead of drawing a circle

```
# Particle.pde
class Particle{
  // ...
  void action(){
    this.planning();
    imageMode(CENTER);
    tint(255, this.lifespan);
    image(img, this.location.x,
              this.location.y); }
}
```

PARTICLE SYSTEM + TEXTURES

- Let's create a smokey effect
- Use the provided texture + the previous particle system
- Requirements:
  - Origin is at 0.75*width, height
  - Velocity of each particle is set at $\mathcal{N}(\boldsymbol{\mu}, \sigma)$, i.e., a value from a normal distribution with mean $\boldsymbol{\mu} = [0, -1]$ and standard deviation $\sigma = 0.3$.
- At each step, apply to every particles a horizontal wind force, i.e., a Pvector with y =0
- For the x, we use a Microphone input (for macOS, a recording of the wind), and we extract the energy as we did in the first lab
  - `Pvector wind= new PVector(-audio.getEnergy(), 0);`
- Use 1000 particles

Processing provides a function randomGaussian() that outputs random values drawn from $\mathcal{N}(0,1)$.

Remember that
$\mathcal{N}(\boldsymbol{\mu}, \sigma) = \mathcal{N}(0,1)\sigma + \mu$

PARTICLE SYSTEM + TEXTURES

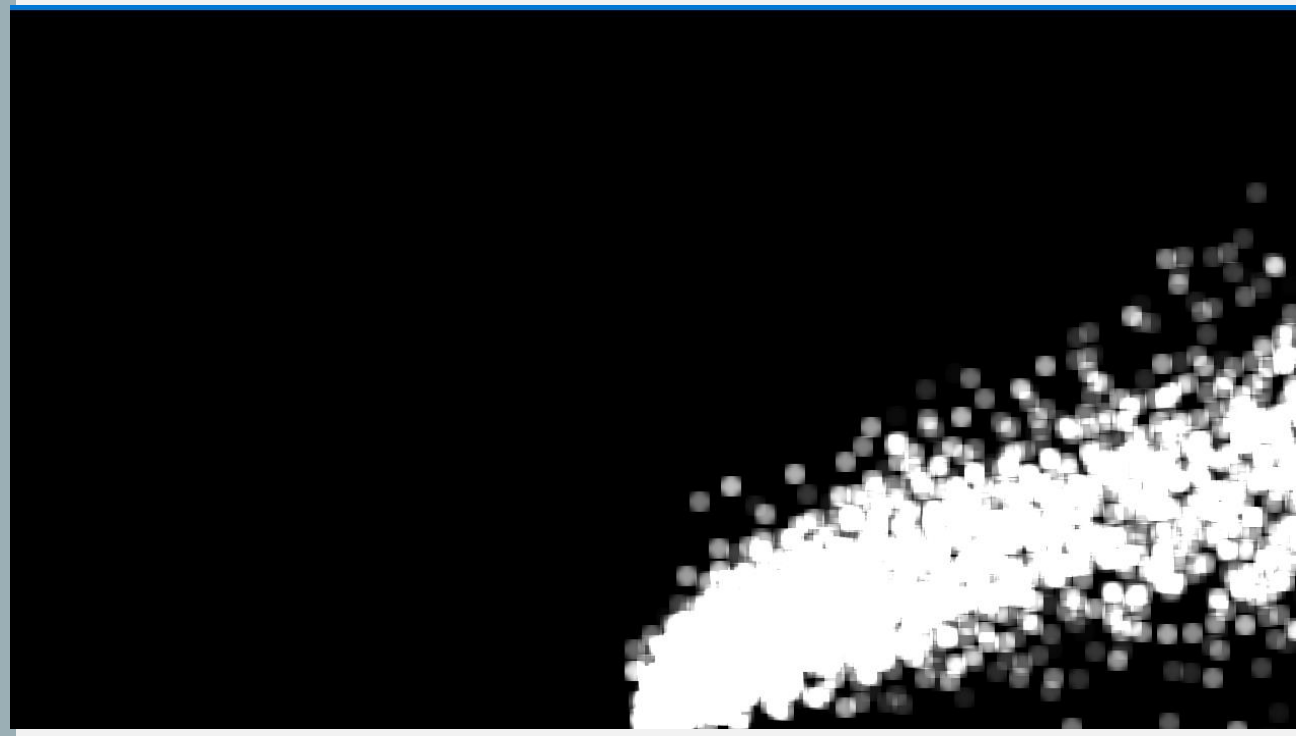
```
# textureParticleSystem.pde
AudioIn audio;
void setup(){
  /* ... */
  audio=new AudioIn(this);
}

PVector computeWind(){
  float energy= audio.getEnergy();
  return new PVector(-energy,0);
}

void draw(){
  blendMode(ADD);
  background(0);
ps.action(computeWind());
}
```

```
# AudioIn.pde
import ddf.minim.*; import ddf.minim.analysis.*;
int frameLength = 1024;
class AudioIn{ // variables...
   AudioIn(textureParticleSystem app){
     this.minim= new Minim(app);
     // use mic input}
  float getEnergy(){
    this.fft.forward(this.mic.mix);
    float energy = 0;
    /* your code */
    energy=map(energy,0, this.fft.specSize(), 0, 1);
    this.energy= this.energy*0.1+energy*0.9;
    return this.energy;
  }
}
```

PARTICLE SYSTEM + TEXTURE

Can you control the wind with some musical features?

Can you combine wind + sinusoidal motion?



BOIDS & BOX2D

BOIDS & BOX2D

- With *boids* we refer to bird-like objects that act in a multiagent system and they are aware of the surrounding
- Boids' flocking behavior is the result of three rules:
  - Avoid the other boids, i.e., do not collide with them
  - Align their direction with nearby boids
  - Approach to distant boids
  - More rules can be defined to, for example, avoid predators

- Online we can find several examples of boids that implement such rules
- Let's instead join the concept of boid with another one: a **physical engine**

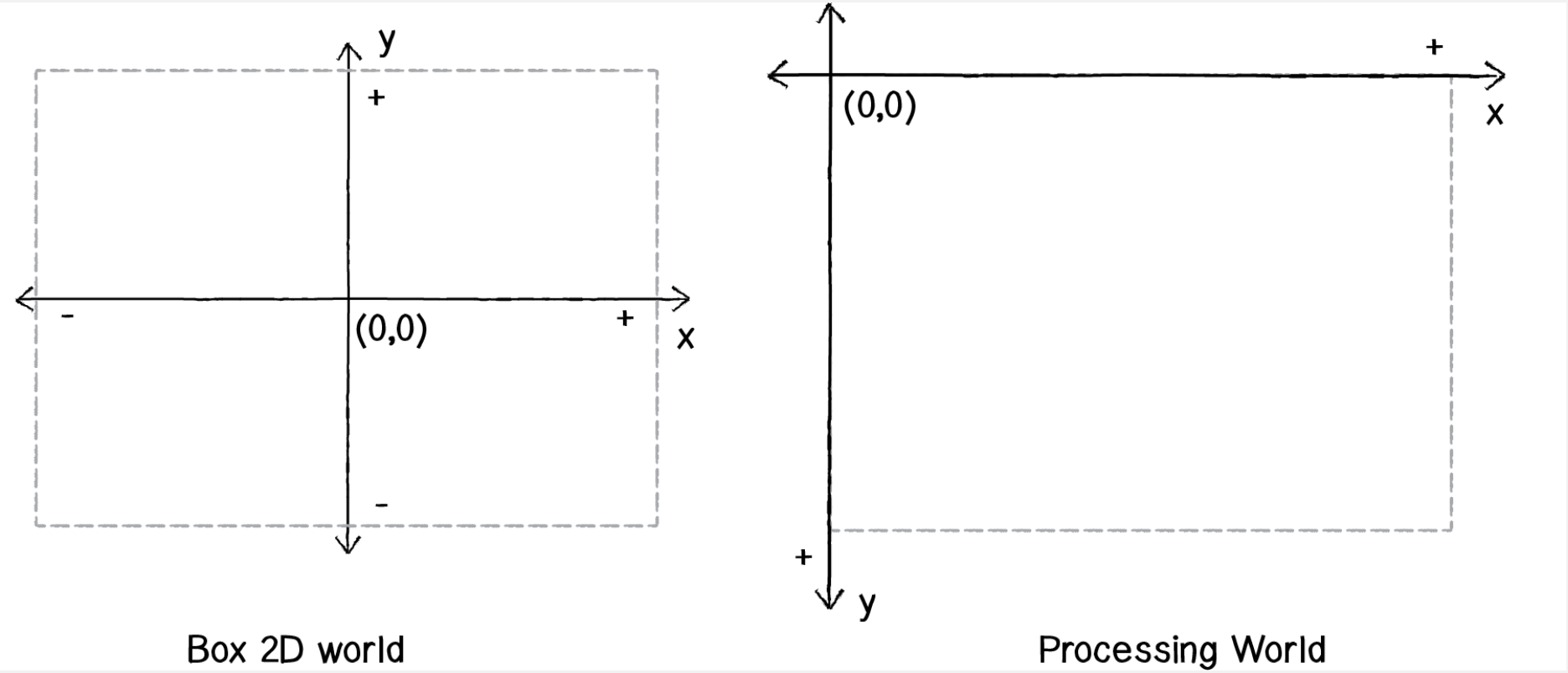
BOIDS & BOX2D

- A **physical engine** is a set of rules which emulate physics
- In the second lab we have seen a small amount of a physical engine by looking at how the object can follow rule of physics
- However, each agent was moving on their own and they may even overlap
- We want to implement **collisions** using the box2d physical engine

- Install **box2d on Processing**
- https://pub.dev/documentation/box2d_flame/latest/box2d/box2d-library.html

BOIDS & BOX2D

- There is a main difference on how we define the world in Processing and in Box2D
  - (there are function to make the convertion for us)



BOIDS & BOX2D

- Instead of Pvector, in Box2D we use Vec2D,
  - The syntax is slightly different as well

PVector	Vec2
PVector a = new PVector(1,-1); PVector b = new PVector(3,4); a.add(b);	Vec2 a = new Vec2(1,-1); Vec2 b = new Vec2(3,4); a.addLocal(b);
PVector a = new PVector(1,-1); PVector b = new PVector(3,4); PVector c = PVector.add(a,b);	Vec2 a = new Vec2(1,-1); Vec2 b = new Vec2(3,4); Vec2 c = a.add(b);

PVector a = new PVector(1,-1);
float m = a.mag();
a.normalize();
	Vec2 a = new Vec2(1,-1);
	float m = a.length();
	a.normalize();

BOIDS & BOX2D

- When using box2d, we need to create a box2d world

```
# example.pde
import org.jbox2d.collision.shapes.*;
import org.jbox2d.common.*;
import org.jbox2d.dynamics.*;
import org.jbox2d.dynamics.contacts.*;
Box2DProcessing box2d;
void setup(){
  box2d = new Box2DProcessing(this);
  box2d.createWorld(); // create world
  box2d.setGravity(0, 0); // no gravity
  ...
}
```

BOIDS & BOX2D

- When using box2d, we need to create a box2d world
- Then create a body with a certain body definition and shape as its *fixture*
  - A DYNAMIC body will move, while a STATIC body is used to draw boundaries or terrain
  - The body automatically implements a function *applyForce*, so we don't have to write it

```
# example.pde
import org.jbox2d.collision.shapes.*;
import org.jbox2d.common.*;
import org.jbox2d.dynamics.*;
import org.jbox2d.dynamics.contacts.*;
Box2DProcessing box2d;
void setup(){
  box2d = new Box2DProcessing(this);
  box2d.createWorld(); // create world
  box2d.setGravity(0, 0); // no gravity
  bd= new BodyDef(); // body definition
  bd.type= BodyType.DYNAMIC;
  cs  = new CircleShape();
  cs.m_radius = P2W(RADIUS_CIRCLE/2);
  bd.linearDamping=0;
  Vec2 position=P2W(new Pvector(witdh/2,
                                height/2));
  bd.position.set(position);
  body = box2d.createBody(bd);
  body.m_mass=1;
  body.createFixture(ps, 1);
}
```

BOIDS & BOX2D

- When using box2d, we need to create a box2d world
- Then create a body with a certain body definition and shape as its *fixture*
  - A DYNAMIC body will move, while a STATIC body is used to draw boundaries or terrain
  - The body automatically implements a function *applyForce*, so we don't have to write it
- we need to convert positions and dimensions from the Pixel domain to the box domain

```
# example.pde
import org.jbox2d.collision.shapes.*;
import org.jbox2d.common.*;
import org.jbox2d.dynamics.*;
import org.jbox2d.dynamics.contacts.*;
Box2DProcessing box2d;
void setup(){
  box2d = new Box2DProcessing(this);
  box2d.createWorld(); // create world
  box2d.setGravity(0, 0); // no gravity
  bd= new BodyDef(); // body definition
  bd.type= BodyType.DYNAMIC;
  cs  = new CircleShape();
  cs.m_radius = P2W(RADIUS_CIRCLE/2);
  bd.linearDamping=0;
  Vec2 position=P2W(new Pvector(witdh/2,
                                height/2));
  bd.position.set(position);
  body = box2d.createBody(bd);
  body.m_mass=1;
  body.createFixture(ps, 1);
}
```

BOIDS & BOX2D

- We need to convert positions and dimensions from the Pixel domain to the box domain
- I defined two functions for you, using function overloading
  - P2W: from Pixels to World
  - W2P: from World to Pixels

```
# example.pde
Vec2 P2W(Vec2 in_value){
  return box2d.coordPixelsToWorld(in_value);}

Vec2 P2W(float pixelX, float pixelY){
  return box2d.coordPixelsToWorld(pixelX, pixelY);}

Vec2 W2P(Vec2 in_value){
  return box2d.coordWorldToPixels(in_value);}

Vec2 W2P(float worldX, float worldY){
  return box2d.coordWorldToPixels(worldX, worldY);}

float P2W(float in_value){
  return box2d.scalarPixelsToWorld(in_value);}

float W2P(float in_value){
  return box2d.scalarWorldToPixels(in_value);}
```

BOIDS & BOX2D

- We need to convert positions and dimensions from the Pixel domain to the box domain
- I defined two functions for you, using function overloading
  - P2W: from Pixels to World
  - W2P: from World to Pix

These functions refer to the Vector worlds, i.e., they convert coordinates from Pixel to World and viceversa

```
# example.pde
Vec2 P2W(Vec2 in_value){
  return box2d.coordPixelsToWorld(in_value);}

Vec2 P2W(float pixelX, float pixelY){
  return box2d.coordPixelsToWorld(pixelX, pixelY);}

Vec2 W2P(Vec2 in_value){
  return box2d.coordWorldToPixels(in_value);}

Vec2 W2P(float worldX, float worldY){
  return box2d.coordWorldToPixels(worldX, worldY);}

float P2W(float in_value){
  return box2d.scalarPixelsToWorld(in_value);}

float W2P(float in_value){
  return box2d.scalarWorldToPixels(in_value);}
```

BOIDS & BOX2D

- We need to convert positions and dimensions from the Pixel domain to the box domain
- I defined two functions for you, using function overloading
  - P2W: from Pixels to World
  - W2P: from World to Pixels

These functions refer to the scalar worlds, i.e., they convert sizes from Pixel to World and viceversa

```
# example.pde
Vec2 P2W(Vec2 in_value){
  return box2d.coordPixelsToWorld(in_value);}

Vec2 P2W(float pixelX, float pixelY){
  return box2d.coordPixelsToWorld(pixelX, pixelY);}

Vec2 W2P(Vec2 in_value){
  return box2d.coordWorldToPixels(in_value);}

Vec2 W2P(float worldX, float worldY){
  return box2d.coordWorldToPixels(worldX, worldY);}

float P2W(float in_value){
  return box2d.scalarPixelsToWorld(in_value);}

float W2P(float in_value){
  return box2d.scalarWorldToPixels(in_value);}
```

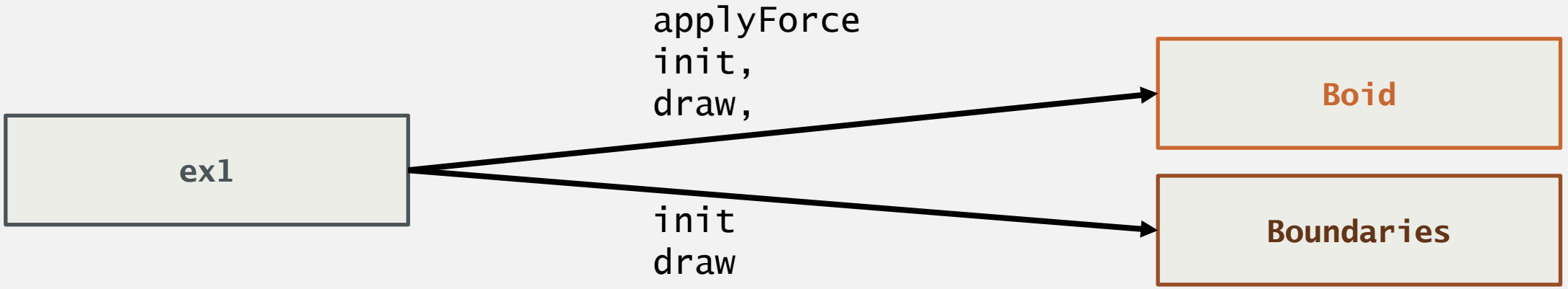
BOIDS & BOX2D

EX 1: Let's start to make **together** our first «stupid» boid

- We start initializing a Box2D world
- Every time we left-click on the screen, a new boid is created
  - we apply a random force to it at the beginning
  - we apply a random force to every boid whenever we press the right-click button
- We include boundaries (STATIC bodies) at the sides of the screen

- The engine is in charge of collisions

BOIDS & BOX2D



Look at the code and let's fill

`ex1->mousePressed()`

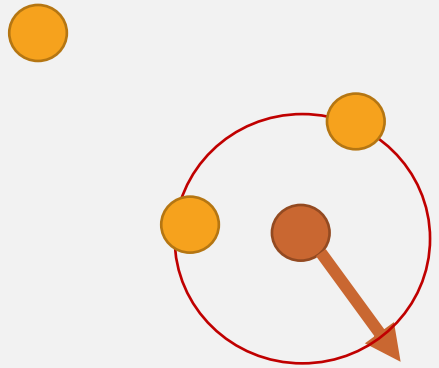
`Boid->draw()`

```
# ex1.pde
void draw() {
  fill(0,50);
  rectMode(CENTER);
  rect(width/2, height/2, width, height);
  box2d.step(); // THIS makes the world update
  boundaries.draw();
  for (Boid b : boids) {
    b.draw();
  }
}
```

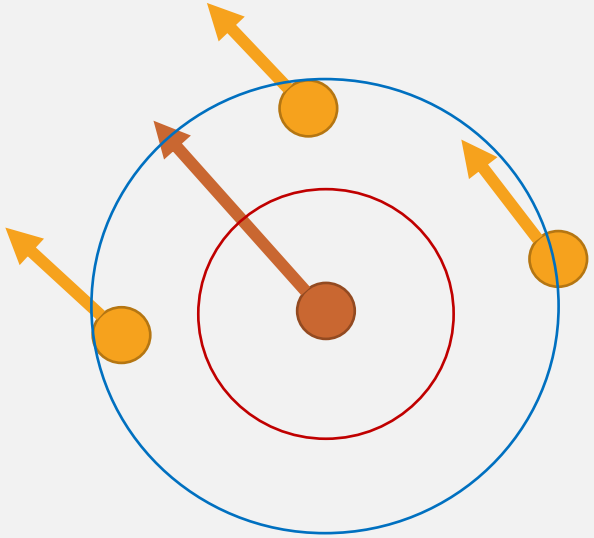
BOIDS & BOX2D

EX 2: Make the boids behave like boids

- **avoid collisions**: whenever other boids are closest than AVOID_DIST, apply a force that is the opposite of the direction toward them, in order to make space



- **align**: steer to align to the direction of boids who are closest than ALIGN_DIST
  - (but further than AVOID_DIST)

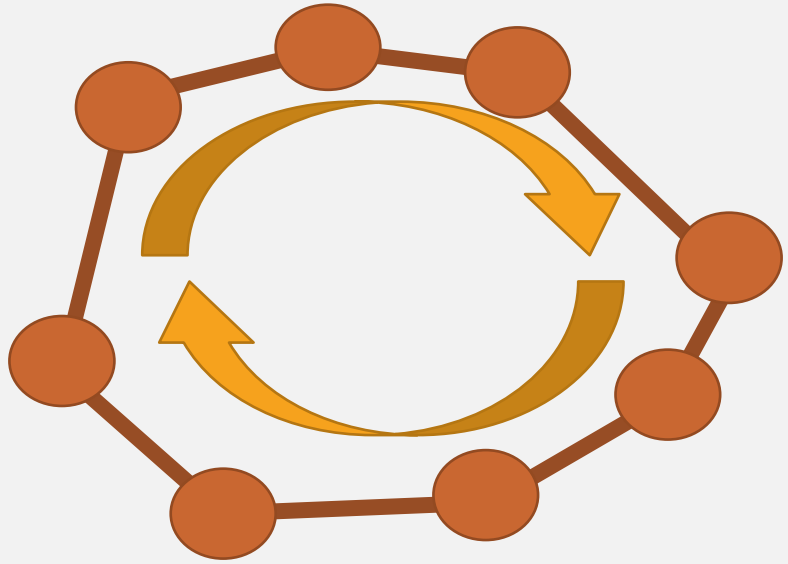


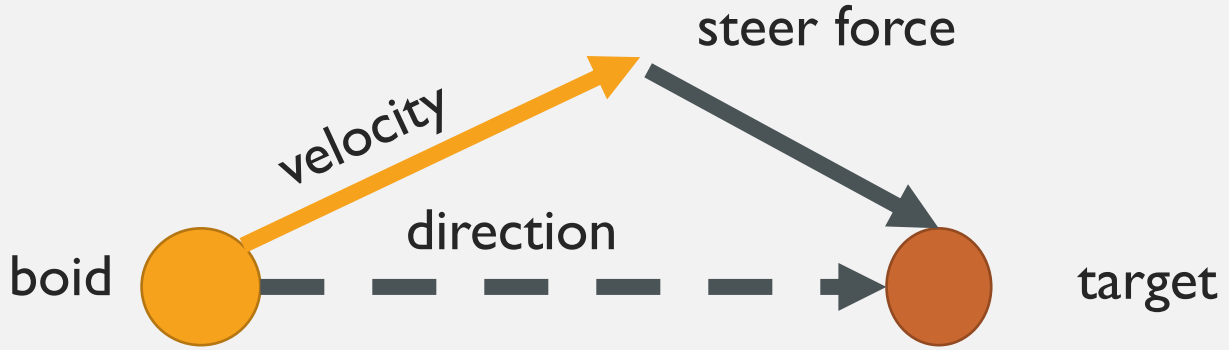
- Implement the method `update(ArrayList<Boid> boids);`

BOIDS & BOX2D

Boids are following a path

- Path is a set of corners (points)
- At every instant, each boid steers to the next point
  - Steering_force = direction – velocity
  - limit the force it in order not to overshoot





BOIDS & BOX2D

- Ex2: Implement the Boid's method `update(ArrayList<Boid> boids);`
  - Compute `avoid_force`: avoid other boids
  - Compute `align_force`: align to other boids

```
# Boid.pde
float AVOID_DIST=6; float ALIGN_DIST=25;
Class Boid{//...
  void update(ArrayList<Boid> boids){
    //...
    Vec2 align_force=new Vec2(0,0); Vec2 avoid_force=new Vec2(
    for(Boid other: boids){
       if(this.body==other.body)// skip
         {continue;}
      // your code }
    // your code
    this.applyForce(avoid_force);
    this.applyForce(align_force);
  }
```

```
# ex2.pde
void draw() {
  //...
  for (Boid b : boids) {
    b.applyForce(computeForce(b));
    b.update(boids);
    b.draw();
  }
}
```

BOIDS & BOX2D

EX 3: Behavior during collisions!

- We want the boid to *react* to the collision by
  - Playing a sound
  - Briefly changing color

BOIDS & BOX2D

EX 3: Behavior during collisions!

- How to make box2d react to collisions?

We ask box2d to listen for Collision

We retrieve the Boid connected with a Body by setting the Boid as "User Data" of the body

Every time a new collision occurs, the function `beginContact` will be called

```
# ex4.pde
void setup() {
  //...
  box2d.createWorld();
  box2d.listenForCollisions();
  //... }

void beginContact(Contact cp) {
  Body body1 = cp.getFixtureA().getBody();
  Body body2 = cp.getFixtureB().getBody();
  Boid b1 = (Boid) body1.getUserData();
  Boid b2 = (Boid) body2.getUserData();
  if (b1!=null) {b1.play(); b1.changeColor();}
  if (b2!=null) {b2.play(); b2.changeColor();}
}
```

```
# Boid.pde
Boid(/*...*/){
  this.box2d = box2d;
  bd.position.set(position);
  /*...*/
  this.body.setUserData(this);
  /* ... */
}
```

BOIDS & BOX2D

EX 3 Behavior during collisions!

- Implement Boid.play and Boid.changeColor()
- we change the constructor of the Boid

```
# ex4.pde
void beginContact(Contact cp) {
  Body body1 = cp.getFixtureA().getBody();
  Body body2 = cp.getFixtureB().getBody();
  Boid b1 = (Boid) body1.getUserData();
  Boid b2 = (Boid) body2.getUserData();
  if (b1!=null) {b1.play(); b1.changeColor();}
  if (b2!=null) {b2.play(); b2.changeColor();}}
```

```
# Boid.pde
class Boid{
  Body body; Box2DProcessing
  box2d; int nextPoint;
  color defColor = color(200, 200, 200); // default color
  color contactColor; // color during contact
  float time_to_color,time_index;
  SoundFile sample;
  Boid(Box2DProcessing  box2d, CircleShape ps, BodyDef bd,
       Vec2 position, SoundFile sample, int nextPoint){
    /*...*/
  }
```

BOIDS & BOX2D

EX 3: Behavior during collisions!

- we change the constructor of the Boid
- You implement Boid.play() , Boid.changeColor() and Boid.draw()

```
# Boid.pde
class Boid{
  Body body; Box2DProcessing
  box2d; int nextPoint;
  color defColor = color(200, 200, 200);
  color contactColor; // color during contact
  float time_to_color,time_index;
  SoundFile sample;
  Boid(Box2DProcessing  box2d, CircleShape ps,
      BodyDef bd, Vec2 position, SoundFile sample,
      int nextPoint){
     /*...*/
  }
```

```
# ex4.pde
void beginContact(Contact cp) {
  Body body1 = cp.getFixtureA().getBody();
  Body body2 = cp.getFixtureB().getBody();
  Boid b1 = (Boid) body1.getUserData();
  Boid b2 = (Boid) body2.getUserData();
  if (b1!=null) {b1.play(); b1.changeColor();}
  if (b2!=null) {b2.play(); b2.changeColor();}}
void setup(){  //...
  String path=sketchPath()+"/sounds";
  File dir = new File(path);
  filenames= dir.list();} // wavfiles
```

BOIDS & BOX2D

EX 4: Behavior during collisions!

- Implement Boid.play and Boid.changeColor()
- we change the constructor of the Boid

```
# Boid.pde
class Boid{
  Body body; Box2DProcessing
  box2d; int nextPoint;
  color defColor = color(200, 200, 200); // de
  color contactColor; // color during contact
  float time_to_color,time_index;
  SoundFile sample;
  Boid(Box2DProcessing  box2d, CircleShape ps,
       Vec2 position, SoundFile sample, int ne
     /*...*/
  }
```

```
# ex4.pde
void beginContact(Contact cp) {
  Body body1 = cp.getFixtureA().getBody();
  Body body2 = cp.getFixtureB().getBody();
  Boid b1 = (Boid) body1.getUserData();
  Boid b2 = (Boid) body2.getUserData();
  if (b1!=null) {b1.play(); b1.changeColor();}
  if (b2!=null) {b2.play(); b2.changeColor();}}
void setup(){  //...
  String path=sketchPath()+"/sounds";
  File dir = new File(path);
  filenames= dir.list(); // wavfiles }
void mousePressed() {
  //insert a new Boid
  Boid b = new Boid(box2d, cs, bd,
                    P2W(mouseX, mouseY),
                    new SoundFile(this,
                       "sounds/"+filenames[i]),
                    p); }
  }
```

BOIDS & BOX2D

```
# create_sounds.py
// use this Python script to generate the collision sounds

#...
env=np.zeros((int(DUR*sr),));
env[:N]=np.sin(np.linspace(0, np.pi, N)) # envelope

if __name__=="__main__":
  for f, freq in enumerate(freqs):
     fn_out="sounds/%.2fHz.wav"%(freq)
     T=int(sr/freq);
     if Osc_type=="square":
        osc=np.zeros((T,))-1;
        osc[int(T/4):int(-T/4)]=1
     elif Osc=="saw":
        osc=np.concatenate([np.linspace(-1,1,int(T/2)),
                            np.linspace(1,-1,T-int(T/2))])
     sample=np.tile(osc, (1+int(DUR/freq),))
     sample=sample[:int(sr*DUR)]
     sample*=env
     sf.write(fn_out, 0.707*sample/np.max(np.abs(sample)), sr)
```

Creating an envelope to avoid abrupt attack

Period in sample corresponding to frequency

Creating the basic shape

Repeat and apply envelope

Write

BOIDS & BOX2D

Hints for `changeColor`

1. Define the color you want to use during a collision in `Boid.contactColor;`
2. Define the duration (in frames) you want the new color to be active in `Boid.time_to_color`
   - `dur_frames=dur_seconds * frameRate`
3. Define an internal variable that is 0 before a collision and `dur_frames` right after a collision
   - at each frame, decrease it by one until is 0
4. Use the function `lerpColor` to move from one color to the other
   - `Color c=lerpColor(color Color0, color Color1, float value);`
   - c is a weighted blending between the two as value*Color1 + (1-value)*Color0