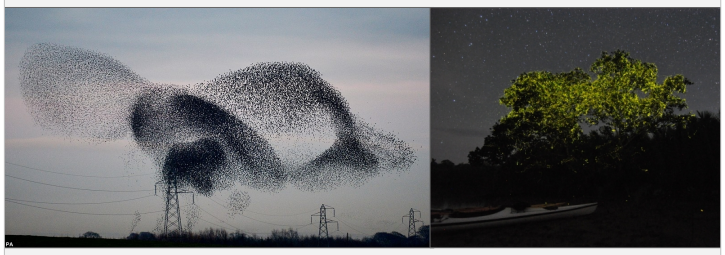
Behavioral Animation

Complex Group Behavior

https://www.youtube.com/watch?v=V4f_1_r80RY



http://www.dailymail.co.uk/news/article-2514252/Incredible-photos-ofhe-moment-TEN-THOUSAND-starlings-fly-formation-Scottish-Borders.html

http://www.firefly.org/synchronous-fireflies.html

Emergent Behavior

Emergent behavior is complex behavior which arises from

- the application of simple rules, and
- the interaction of (only) nearby individuals

→ *global patterns from local behavior*
→ *organization without a leader*

Many natural systems display emergent behavior.





https://en.wikipedia.org/wiki/Emergence
http://www.pbs.org/wgbh/nova/nature/emergence-examples.html

Boids

- a model for coordinated motion in groups of animals
  - demonstrates emergent behaviors
- due to Craig Reynolds
  - published at SIGGRAPH 1987
- a major advance in computer animation in movies
  - Reynolds won a Scientific & Engineering Academy Award in 1998

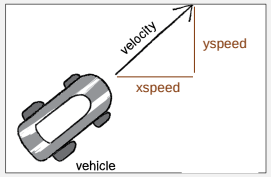
demonstrated in *Stanley and Stella in: Breaking the Ice*, 1987

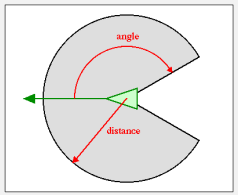
http://www.youtube.com/watch?v=3bTqWsVqyzE

first feature film use in *Batman Returns*, 1992

Boids – Elements

- a basic boid has
  - position
  - velocity (captures both speed and direction of movement)
- additional properties
  - maximum acceleration
  - maximum speed
- in groups, a boid only reacts to its neighbors
  - behaviors are local
  - *neighborhood* is defined by a distance (radius) and an angle

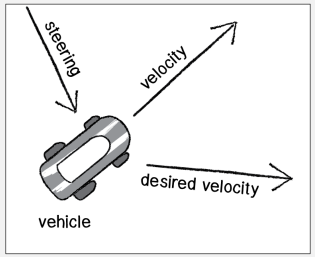


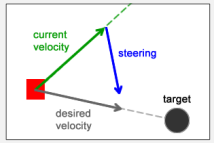


http://natureofcode.com/book/chapter-6-autonomous-agents/
http://www.red3d.com/cwr/boids/

Boids – Steering

- a **steering vector** indicates how the boid wants to turn
  - represents direction and effort
- the steering vector is used to update the boid's velocity
- a **steering behavior** is a method for computing a steering vector



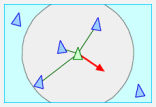


seek behavior

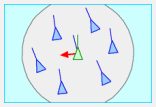
http://natureofcode.com/book/chapter-6-autonomous-agents/
https://gamedevelopment.tutsplus.com/tutorials/understanding-steering-behaviors-flee-and-arrival--gamedev-1303

Boids – Flocking/Schooling

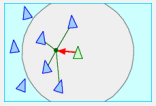
- flocking/schooling arises from three simple steering behaviors



separation: steer away from your neighbors so you don't get too close



alignment: steer to match velocity of neighbors

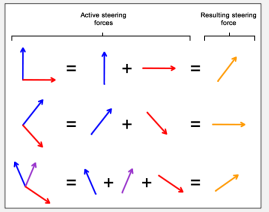


cohesion: steer to move towards the center of your neighbors so you don't get too far apart

http://www.red3d.com/cwr/boids/

Net Steering Force

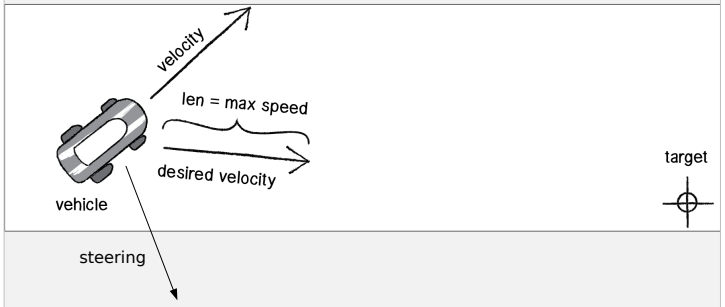
- steering vectors for multiple behaviors can be combined into the **net steering vector** (or **force**)
  - net steering force is used to update the boid's velocity



https://gamedevelopment.tutsplus.com/tutorials/understanding-steering-behaviors-movement-manager--gamedev-4278

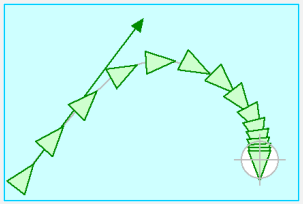
Seek

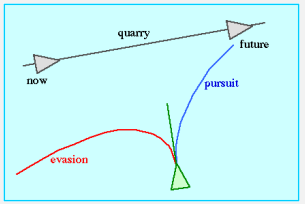
- **seek** steers the boid to head towards the target at its maximum speed



Arrive and Pursue

- **arrive** is similar to seek, but the boid slows once it is within the *stopping radius* near the target
- **pursue** is similar to seek, but uses an estimate of the interception point instead target's current position



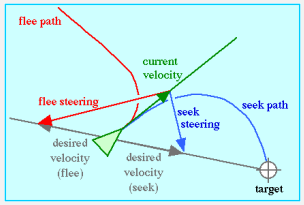


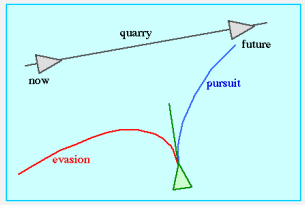
http://www.red3d.com/cwr/steer/gdc99

Flee and Evade

- **flee** is the opposite of seek
- **evade** is the opposite of pursue

In both cases, the goal is to steer so as to be heading away from the target as fast as possible.

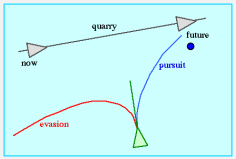




http://www.red3d.com/cwr/steer/gdc99

Variations on the Theme

- **offset pursuit**
  - seek target is some distance *d* to the side of the predicted future position of the target



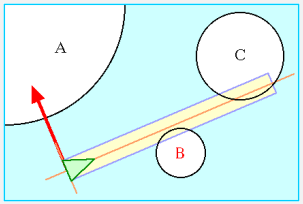
- **interpose**
  - seek a target point between two other boids
- **hide**
  - seek a target on the opposite side of an obstacle from another boid

Forward and Wander

- **forward** steers in the same direction but accelerates as needed to reach maximum speed
- **wander** steers randomly, but not *too* randomly
  - implemented by seeking a point ahead of the boid and not too far to the side

Obstacle Avoidance

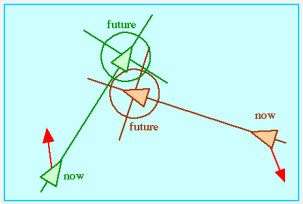
Avoid hitting obstacles.



- identify the most threatening obstacle
  - project region ahead of the boid
    - length depends on boid's speed and agility (longer for faster / less agile)
  - of the obstacles intersecting the region, find the obstacle closest to the boid
- steer away from it

Unaligned Collision Avoidance

Avoid collisions between boids.

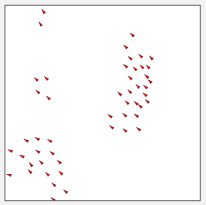


- identify potential collisions
  - determine the point of nearest approach for each other boid
    - assume boids continue with their current velocities
- steer to avoid the location of the nearest such potential collision

Combining Multiple Behaviors

Can have multiple behaviors in effect at once –

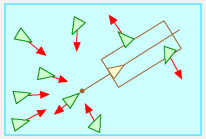
- **flocking** combines separation, alignment, and cohesion
  - also forward or wander or another movement element



Combining Multiple Behaviors

Can choose between different behaviors at different times –

- **shadow**
  - if close to target, use alignment to match velocity
  - otherwise arrive at target
- **leader following** – followers want to stay near the leader without crowding the leader or each other or getting in the leader's way
  - if follower is in a region in front of the leader, it steers away from the leader's path
  - otherwise, followers arrive at a point just behind the leader
  - separation is used to keep followers out of each other's way
  -



http://www.red3d.com/cwr/steer/gdc99

Behavioral Animation

- combines steering and other actions with a higher-level "brain" to determine what action(s) to do in what situations

The Lion King (1994)

https://www.youtube.com/watch?v=NofrY8eB3u4

The Lord of the Rings (2001-2003)

MASSIVE SOFTWARE

https://www.youtube.com/watch?v=rCZ3SN65kIs

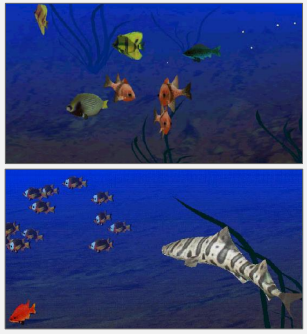
Shrek 2 (2004)

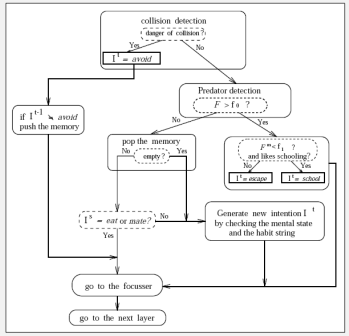
Artificial Fish

https://www.youtube.com/watch?v=VpZ93n5QQuQ

- Tu and Terzopoulos, 1994

  "Imagine a virtual marine world inhabited by a variety of realistic fishes..."

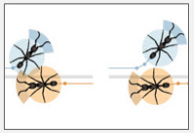


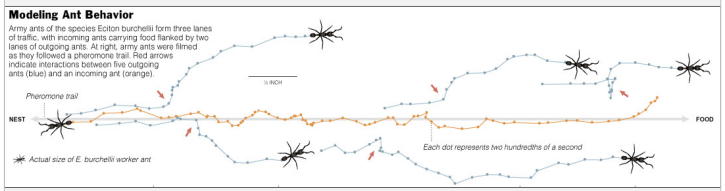


Tu, Terzopoulos, "Artificial Fishes: Physics, Locomotion, Perception, Behavior", SIGGRAPH '94
http://www.siggraph.org/education/materials/HyperGraph/animation/art_life/fish.htm

Ants

- basic behaviors
  - follow pheromone trail
  - if another ant is met, slow down and turn aside

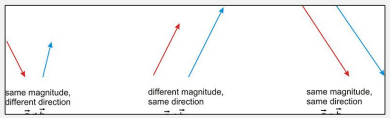


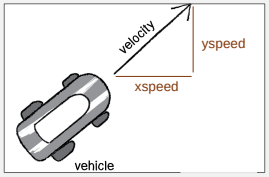


http://www.nytimes.com/imagepages/2007/11/12/science/2007113_TRAFFIC_GRAPHIC.html

Boids in Processing

- velocity is represented by a *vector*
  - has both direction and magnitude





- boid's position and velocity are variables of type `PVector`

we'll use

```
PVector position;
PVector velocity;
```

instead of

```
float x, y;
float xspeed, yspeed;
```

  - `.x` and `.y` to access the x and y components

```
position.x   // x          velocity.x   // xspeed
position.y   // y          velocity.y   // yspeed
```


http://natureofcode.com/book/chapter-6-autonomous-agents/
https://www.ck12.org/book/ck-12-trigonometry-second-edition/section/5.6/

Boids in Processing

- `draw()` contains four steps –
  - draw the scene
  - compute the net steering force –
    - get the steering force for each behavior
    - combine the forces
    - limit the size of the force
  - update boid's velocity
    - add the net steering force to the velocity
    - limit the max speed
  - update boid's position
    - add the velocity to the position
    - wrap at the edges of the window

three patterns for combining forces, depending on whether there is just one behavior, multiple behaviors active at once, or action selection to choose active behavior(s)

```
position = position + speed;
speed = speed + acceleration;
```

⇨

```
position = position + velocity;
velocity = velocity + net steering force;
```



```
void draw () {

  // 1 - draw scene
  background(255);
  drawBoid(pos, vel, 255, 0, 0);

  // 2 - compute net steering force
  //  a - compute steering force for each behavior
  PVector wander = computeWander(pos,vel);
  //  b - combine forces (one behavior)
  PVector steer = new PVector(0,0);
  steer.add(wander);
  //  c - limit the size of the force that can be applied
  steer.limit(maxforce);

  // 3 - update boid's velocity
  //  a - add net steering force
  vel.add(steer);
  //  b - limit boid's max speed
  vel.limit(maxspeed);

  // 4 - update boid's position
  //  a - update position by adding velocity
  pos.add(vel);
  //  b - wrap at edges of window
  if ( pos.x > width ) {
    pos.x = 0;
  } else if ( pos.x < 0 ) {
    pos.x = width;
  }
  if ( pos.y > height ) {
    pos.y = 0;
  } else if ( pos.y < 0 ) {
    pos.y = height;
  }
}
```

we'll use a library of behaviors (provided code) to get the steering force for each desired behavior – won't need to compute them directly

the topics exercises involve only step 2 – the rest of the sketch is provided code that you can use as-is

Combining Forces

```
// 2 - compute net steering force
//  a - compute steering force for each behavior
PVector wander = computeWander(pos,vel);
//  b - combine forces (one behavior)
PVector steer = new PVector(0,0);
steer.add(wander);
//  c - limit the size of the force that can be applied
steer.limit(maxforce);
```

one behavior – get the steering force and add to `steer`

```
// 2 - compute net steering force
//  a - compute steering force for each behavior
PVector wander = computeWander(pos,vel);
PVector seek = computeSeek(pos,vel,new PVector(mouseX,mouseY));
//  b - combine forces (weighted sum)
PVector steer = new PVector(0,0);
wander.setMag(1.2);
seek.setMag(1);
steer.add(wander);
steer.add(seek);
//  c - limit the size of the force that can be applied
steer.limit(maxforce);
```

multiple behaviors active at the same time – get the steering force for each behavior, adjust the weight of each behavior, (`setMag`), and add the scaled force to `steer`

`setMag` sets the length of the vector – defines the weight of that behavior in the net steering force relative to the others
(all same magnitude = all contribute equally, regards of the actual value of the magnitude)

```
// 2 - compute net steering force
//  a - compute steering force for each behavior
PVector wander = computeWander(pos, vel);
PVector seek = computeSeek(pos, vel, new PVector(mouseX, mouseY));
//  b - combine forces (action selection)
PVector steer = new PVector(0, 0);
if ( pos.x < width/2 ) {  // wander in the left side of the window
  steer.add(wander);
} else {  // seek in the right side of the window
  steer.add(seek);
}
//  c - limit the size of the force that can be applied
steer.limit(maxforce);
```

choosing between behaviors – get the steering force for each behavior, then conditionally add steering forces to `steer`