345 Ludic Computing

Lecture 9

Steering Behaviours

Simon Colton & Alison Pease
Computational Creativity Group
Department of Computing
Imperial College London

www.doc.ic.ac.uk/ccg

Lecture Schedule

Week	Thurs 4pm	Thurs 5pm
2	1. Introduction	2. CC
3	3. Evolutionary Design 1	4. Evolutionary Design 2
4	5. Shape Grammars	6. L-Systems
5	7. Image Filters	8. NPR Rendering
6	9. Steering Behaviours	10. Pathfinding
7	11. Behaviour Trees	12. Adaptive Games
8	13. Interactive Play	14. Designing for Play
9	15. Social Networks	16. Monte Carlo Tree Search
10	17. Revision lecture	

Movement in Games

- Game agents typically have to **act** and **move**
- Two approaches to movement
  - **Reactive**: continuously sense and react to surroundings (this lecture)
  - **Planned**: construct and execute navigation plan based on world representation (next lecture)
- Hybrid approaches can work well

Reactive Movement

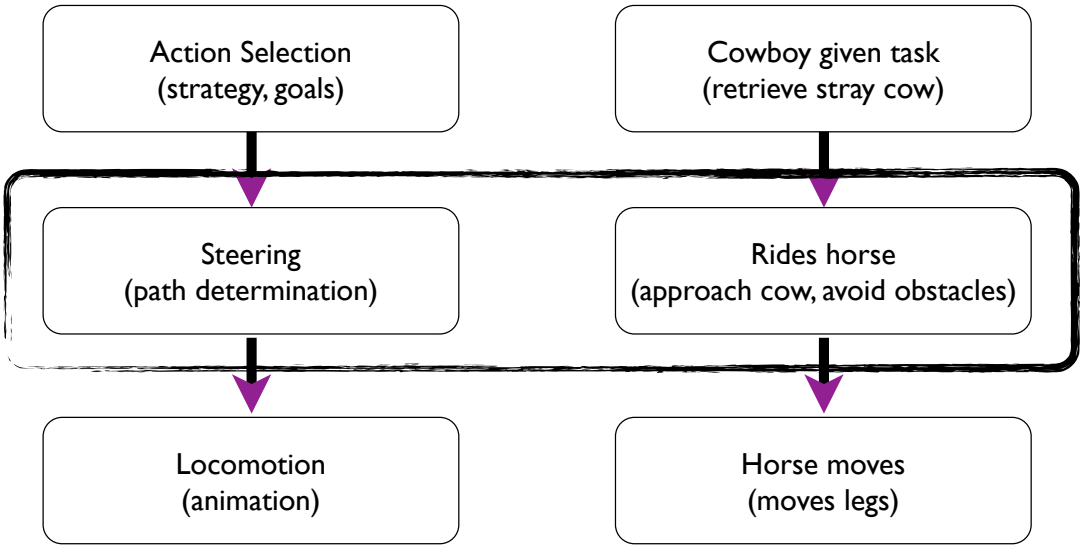
- Agent continuously senses and reacts to its surroundings
- **Easy to comput**e: little time/memory, local information
- **Dynamic**: based on current state of world
- Good: Fast responses to dynamic environment (e.g. dodging moving obstacles); low-intelligence agents (e.g. fish, birds); open environments with simple convex obstacles (e.g. a field with trees)
- Bad: can not handle complex layouts and goals; difficult to tune; prone to unrealistic or unpredictable behaviour

This Lecture

- Steering behaviours as a reactive solution
- The steering model
- Some steering behaviours in detail
- Quick tour of some other behaviours
- Combining behaviours
- Emergent behaviours

Reactive Movement

Action, Steering, Locomotion



Notation

- Column vector $\mathbf{v} = \begin{pmatrix} v_x \\ v_y \end{pmatrix}$
- Multiplication of vector and scalar written . (dot)
- Magnitude: $|\mathbf{v}| = \sqrt{(v_x^2 + v_y^2)}$
- Unit vector: $\text{unit}(\mathbf{v}) = \mathbf{v}/|\mathbf{v}|$
- $\text{trim}(\mathbf{v}, k) = k.\text{unit}(\mathbf{v})$ if $|\mathbf{v}| > k$, else $\mathbf{v}$

A Steering Agent

- Body with mass m
- At time t it has position **p**, velocity **v** and it chooses a *steering force* **f** (all 2D vectors)
- Maximum speed s_{max} and force f_{max} (scalars)
- Orientation vector, i.e. direction agent is facing
    - In this lecture: orientation = unit(**v**)
- Beyond this lecture: orientation and rotation; 3D space (orientation with 'forward' and 'up' vectors); actuation (motor control); smoothing; ....

Steering Simulation

- Step-by-step simulation

$$\mathbf{p'} = \mathbf{p} + \mathbf{v}.\Delta t$$
$$\mathbf{v'} = \mathbf{v} + \mathbf{a}.\Delta t$$
$$\mathbf{f} = m\mathbf{a}$$

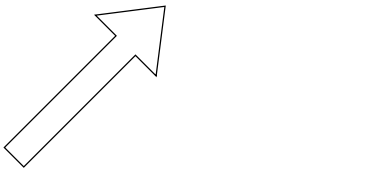
- Agent steers by varying force **f**

$$\mathbf{f} = \text{trim}(\text{steer}(\mathbf{p}, \mathbf{v}), \mathbf{f}_{max})$$

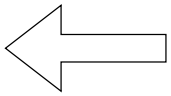
- Aim: movement that looks intentional
  - We don't care if physics is 100% accurate!

Steering Simulation

Calculate steering force
$\mathbf{f} = \text{trim}(\text{steer}(\mathbf{p}, \mathbf{v}), f_{max})$



Update position
$\mathbf{p} = \mathbf{p} + \Delta t.\mathbf{v}$

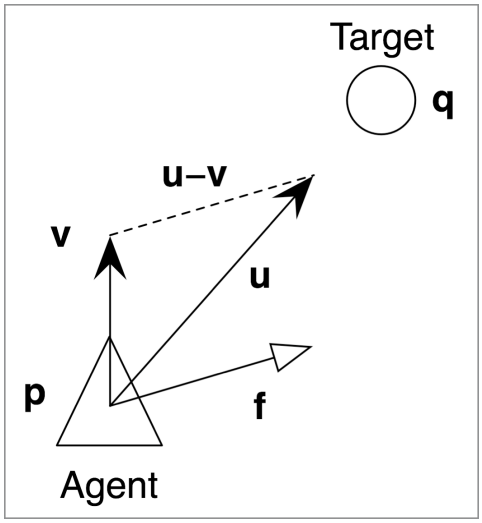


Update velocity
$\mathbf{v} = \text{trim}(\mathbf{v} + \Delta t.\mathbf{f}/m, s_{max})$

Seek

- Moves towards target at position **q** as fast as possible
- Target direction **d** = unit(**q** − **p**)
- Desired velocity **u** = **d**.s_{max}
- Steering force Seek(**q**) = **u** − **v**

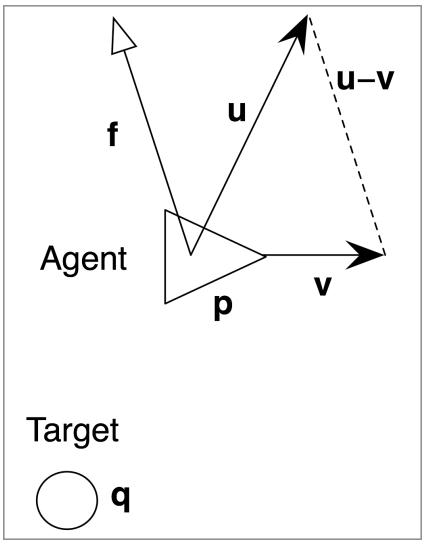
Seek(**q**) = s_{max}.unit(**q** − **p**) − **v**



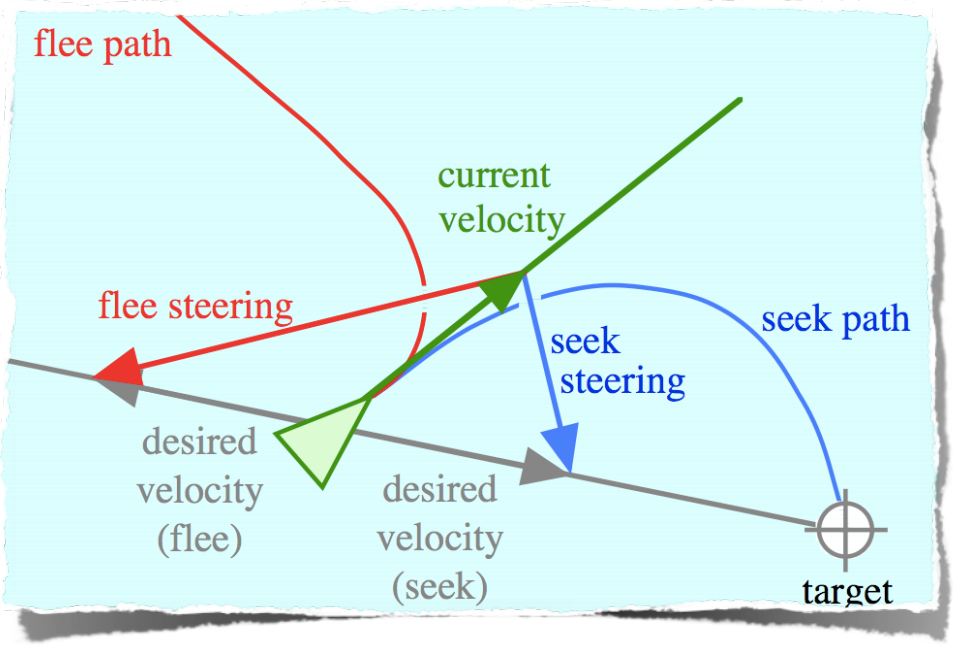
Flee

- Flee moves away from target **q** as fast as possible
- Away from target **d** = unit(**p** − **q**)
- 'Seek' towards **d**

Flee(**q**) = s_{max}.unit(**p** − **q**) − **v**



Seek & Flee



Demo 1

Image: Craig Reynolds

Seek

Worked Example

Agent has m = 1, s_{max} = 2, f_{max} = 4 and $\mathbf{q}$ = (4,3). Δt = 0.5

At t_0 = 0, $\mathbf{p}_0$ = (0,0) and $\mathbf{v}_0$ = (0,0).

Force on agent is $\mathbf{f}_0 = s_{max}.\text{unit}(\mathbf{q} - \mathbf{p}_0) - \mathbf{v}_0$ = (1.6, 1.2)

t_1=0.5 $\mathbf{v}_1 = \text{trim}(\mathbf{v}_0 + \Delta t.\mathbf{f}_0/m, s_{max})$ = (0.8, 0.6)

$\mathbf{p}_1 = \mathbf{p}_0 + \Delta t.\mathbf{v}_1$ = (0.4, 0.3)

$\mathbf{f}_1 = \text{trim}(s_{max}.\text{unit}(\mathbf{q} - \mathbf{p}_1) - \mathbf{v}_1, f_{max})$= (0.8, 0.6)

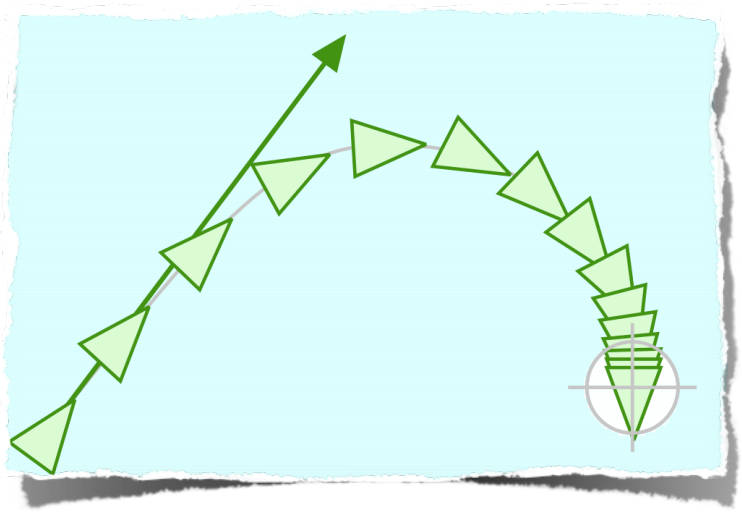
t_2=1.0 $\mathbf{v}_2 = \text{trim}(\mathbf{v}_1 + \Delta t.\mathbf{f}_1/m, s_{max})$ = (1.2, 0.9)

$\mathbf{p}_2 = \mathbf{p}_1 + \Delta t.\mathbf{v}_2$ = (1, 0.75)

$\mathbf{f}_2 = \text{trim}(s_{max}.\text{unit}(\mathbf{q} - \mathbf{p}_2) - \mathbf{v}_2, f_{max})$= (0.4, 0.3)

Arrive

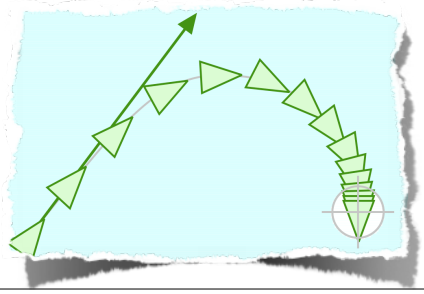
Seek can overshoot the target, so instead decelerate to the target **q** when we get within a certain distance



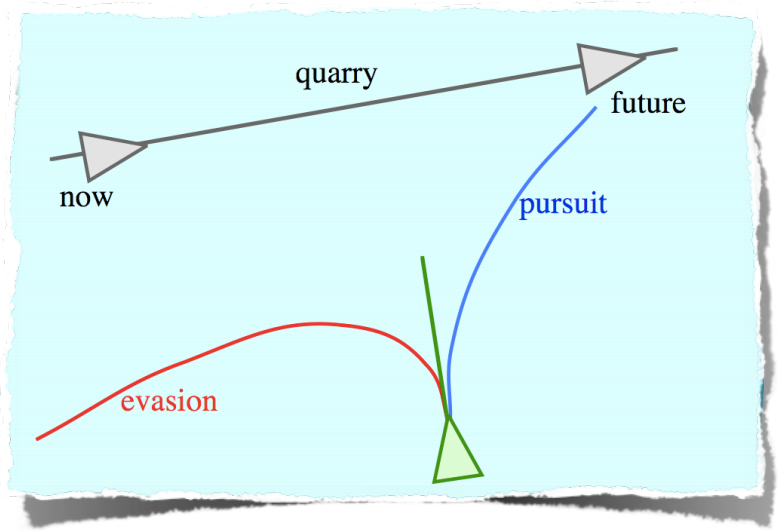
Arrive

- Seek with brakes: reduce speed when within stopping distance d_{stop}
- Target distance $d_{off} = |\mathbf{q} - \mathbf{p}|$
- Ideal speed $\sigma = \begin{cases} s_{max} & \text{if } d_{off} > d_{stop} \\ s_{max}.d_{off} / d_{stop} & \text{otherwise} \end{cases}$
- $\mathbf{u} = \sigma.\text{unit}(\mathbf{q} - \mathbf{p})$

$\text{Arrive}(\mathbf{q}) = \sigma.\text{unit}(\mathbf{q} - \mathbf{p}) - \mathbf{v}$



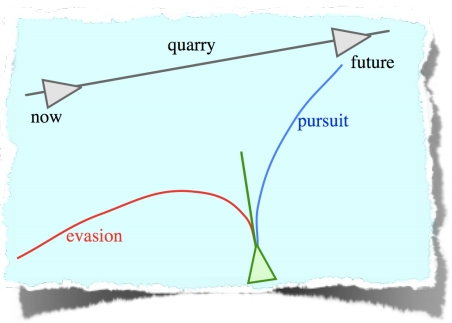
Pursue and Evade



When heading for a moving target **q** it's better to aim for where the target is heading, rather than where it is right now

Pursue and Evade

- Assume target maintains current velocity $\mathbf{v}_q$ and predict its position $\mathbf{q}_{pred}$ at some time $t + t_{pred}$ in the future ($t_{pred} < t_{lim}$)
- $\mathbf{q}_{pred} = \mathbf{q} + \mathbf{v}_q.\min(t_{pred}, t_{lim})$
- Pursue(**q**) = Seek($\mathbf{q}_{pred}$) or Arrive($\mathbf{q}_{pred}$)
- Evade(**q**) = Flee($\mathbf{q}_{pred}$)



Pursue and Evade

How Far To Look Ahead?

Constant $t_{pred} = k$ for some k

Distance $t_{pred} = k. |\mathbf{q} - \mathbf{p}|$

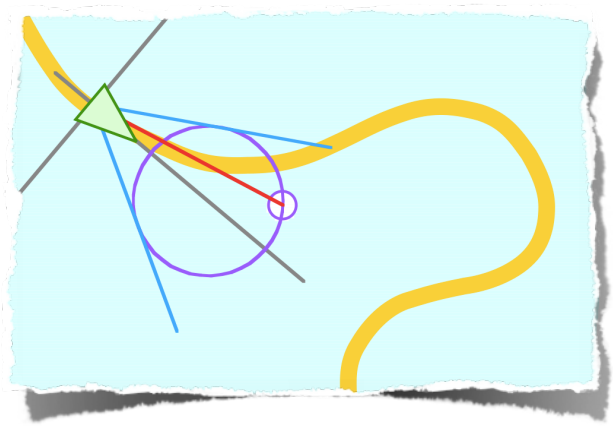
Time to target $t_{pred} = k. |\mathbf{q} - \mathbf{p}| / |\mathbf{v}|$

Reduce for high speeds

$$t_{pred} = k. |\mathbf{q} - \mathbf{p}| / (|\mathbf{v}| + |\mathbf{v}_q|)$$

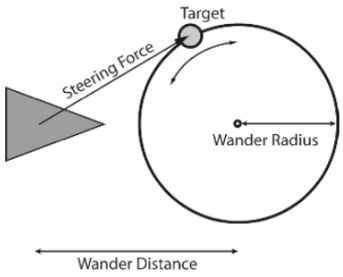
Wander

- Sometimes an agent needs to move along a 'random' path. But random steering forces look unnatural: jerky movements without sustained turns
- Instead a) fix a **wander circle** ahead of the agent b) Seek a target **q** that moves randomly around the circle

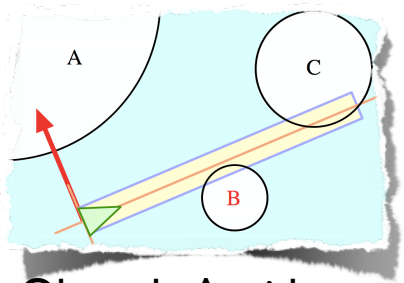


Wander

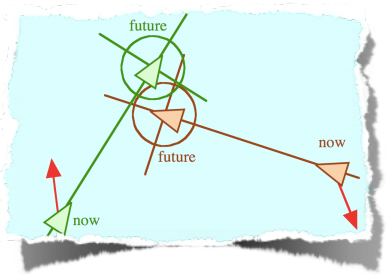
- Parameters: wander radius r_w, wander distance d_w and target jitter α
- Centre of wander circle is $\mathbf{p}_w = \mathbf{p} + d_w.\text{unit}(\mathbf{v})$
- Target $\mathbf{q} = \mathbf{p}_w + \mathbf{c}$ ($\mathbf{c}$ goes from centre to point on circle)
- Each time step we randomly change $\mathbf{c}$
  - Displace to $\mathbf{c} + (\alpha x, \alpha y)$ where random $x, y \in [-1, 1]$
  - Project back onto circle: $\mathbf{c} = r_w.\text{unit}(\mathbf{c} + (\alpha x, \alpha y))$
- Hence: Wander = Seek($\mathbf{p} + d_w.\text{unit}(\mathbf{v}) + \mathbf{c}$)



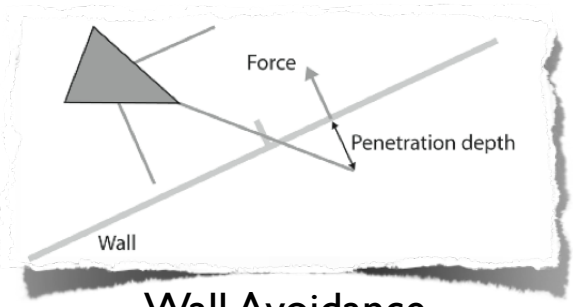
Avoidance Behaviours



Obstacle Avoidance



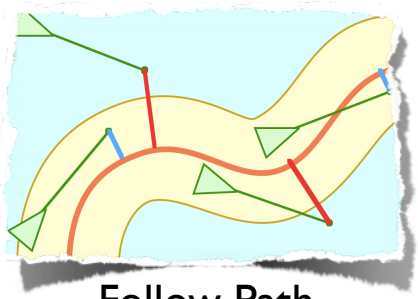
Collision Avoidance



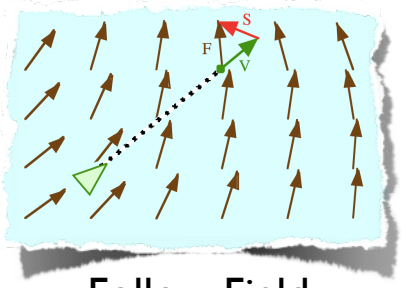
Wall Avoidance

See Reynolds, Buckland for details

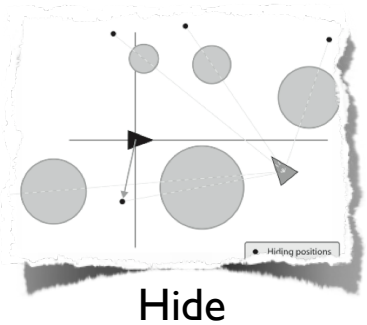
More Behaviours...



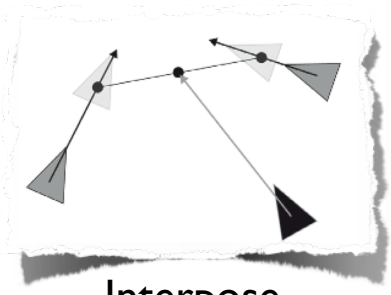
Follow Path



Follow Field



Hide



Interpose

See Reynolds, Buckland for details

Combining Behaviours

- Game agents rarely use single behaviour, but combine them using **blends** and **modes**
- **Mode**: agent selects a behaviour based on current conditions
- **Blend**: agent uses weighted sum of forces from multiple behaviours (trimmed to f_{max})
- Can build up sophisticated behaviours using a mixture of blending and modes

Combining Behaviours
Capture the Flag

- **Attacker**: seeks the flag, evades the defenders and avoids obstacles
    - If there is a defender-free "corridor" between it and the flag it stops evading and just seeks.
- **Defenders** pursue the attacker and avoid obstacles
    - They know not to predict movement of attacker past the flag

http://opensteer.sourceforge.net/

Combining Behaviours
Pedestrians

Each agent combines the following behaviours

- Path following along the red line, with a reversal of direction at path endings
- Collision avoidance with the other agents
- Obstacle avoidance with three large obstacles
- Wandering

http://opensteer.sourceforge.net/

Combining Behaviours
Weighted Blending

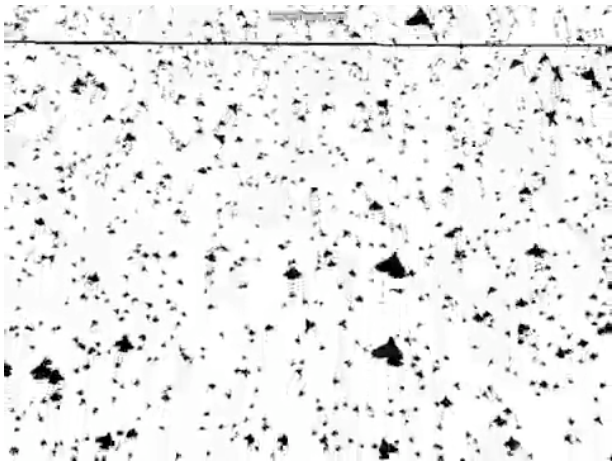
- Blended behaviours sometimes cancel each other out
    - An agent trapped between a wall and a pursuer fails to avoid either
- Behaviours can be prioritised by giving important ones larger weights. But we risk...
    - Higher behaviours that **dominate** lower ones even when weak
    - **Wasting time** computing lower behaviours that end up contributing little

Combining Behaviours
Prioritised Blending

- Behaviours are calculated from highest to lowest priority and added to a weighted running sum **S**
- After calculating a behaviour's force **f**
    - If $|\mathbf{S} + \mathbf{f}| \geq f_{max}$ then add (weight × trim($\mathbf{f}, f_{max} - |\mathbf{S}|$)) to **S** and stop calculating
    - Otherwise, we add weight × **f** to **S**
- Now high priority behaviours *dominate when they need to*
    - When large they prevent the calculation of lower behaviours
    - Don't need to give them huge dominating weights

Emergent Behaviour

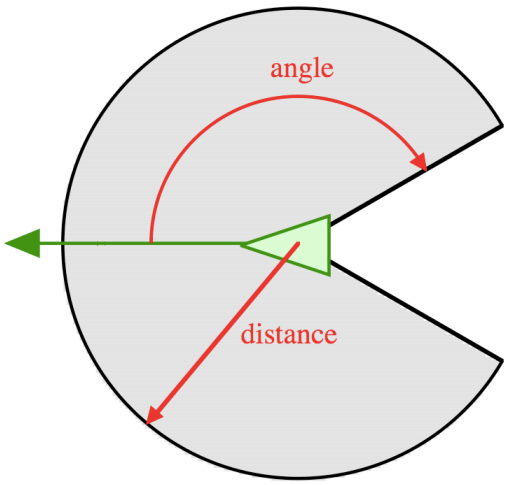
- Complex global behaviour can arise from the interaction of simple rules
- **Flocking** is a weighted sum of three simple steering behaviours: **separation**, **cohesion** and **alignment**



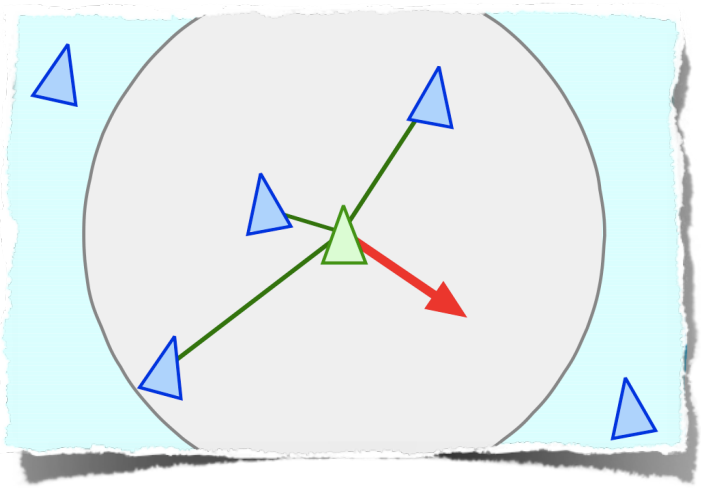
http://www.youtube.com/watch?v=xM9DHHBqBCk

Flocking
Neighbourhood

- Each agent (or **boid**) can only perceive a local neighbourhood described by an angle and a distance
- The set of n neighbours N contains all the agents within this neighbourhood

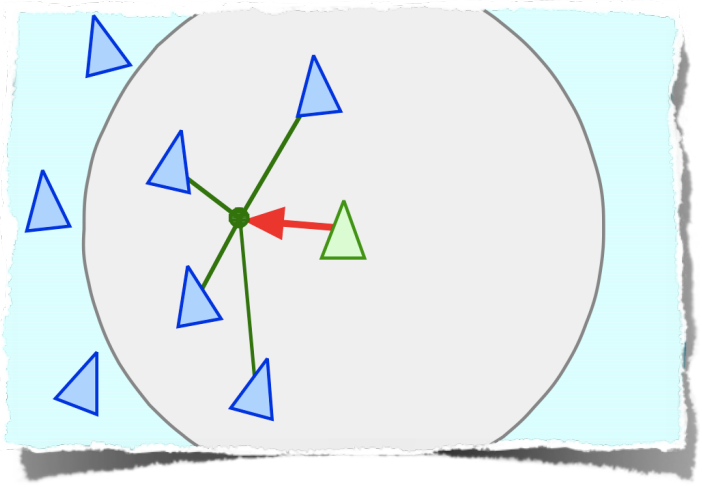


Flocking
Separation



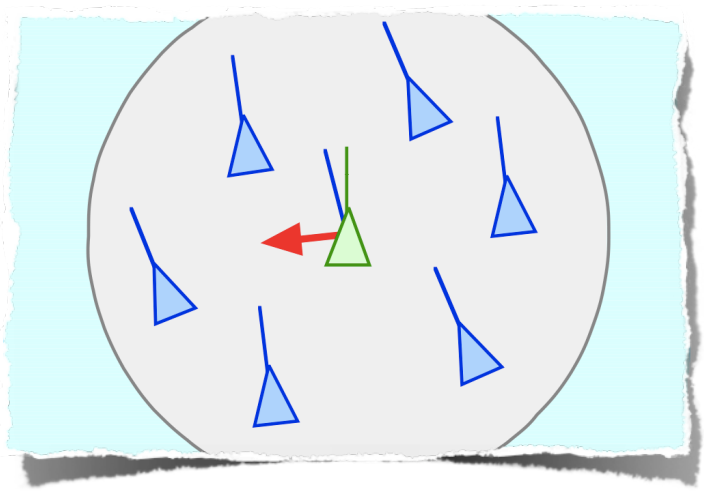
$$\mathbf{f}_{sep} = \sum_{a \in N} \text{unit}(\mathbf{p} - \mathbf{q}_a) / |\mathbf{p} - \mathbf{q}_a|$$

Flocking
Cohesion



$$\mathbf{f}_{coh} = \text{Seek}\left(\sum_{a \in N} \mathbf{p}_a / n \right)$$

Flocking
Alignment



$$\mathbf{f}_{align} = \left(\sum_{a \in N} \mathbf{v}_a / n\right) - \mathbf{v}$$

Pigeons in the Park

http://www.red3d.com/cwr/papers/2000/pip.html

Steering Behaviours
Summary

- Steering agent model and simulation
- Seek, Flee, Arrive, Pursue, Evade, Wander, ...
- Combining behaviours
  - Modes & blends, weighted & prioritised blends
- Emergent behaviours: flocking

Reading and Resources
(Optional)

- Craig Reynolds (1999) *Steering Behaviors For Autonomous Characters*, http://www.red3d.com/cwr/papers/1999/gdc99steer.html (see this website for a collection of steering-related resources)
  - http://opensteer.sourceforge.net/
- Steering in Unity http://www.arges-systems.com/articles/35/
- Mat Buckland (2005) *Programming Game AI by Example*
- Ian Millington & John Funge (2009) *AI for Games*, 2nd edition

Image sources for this lecture: Buckland, Millington, Reynolds