Software Design

Flocking Algorithms

Stuart Porter

Email: stuart.porter@york.ac.uk

1. Flocking Algorithms

2. Useful tips / techniques
   - What to avoid
   - How to fix

Flocking Algorithms

Flocks

- ***Groups*** of animals that ***appear*** to act as one ***coherent*** body

- Each individual / agent / boid can only ***see*** other flock members within a given radius

- All the individuals / agents / boids in the flock follow the same ***simple*** rules
  - By following these rules, complex ***emergent*** behaviours can be seen
    - That are not always apparent from the individual rules

  - Flocking is ***not*** overall control – it is ***low-level*** behaviour
    - So, should still use basic "`move(int)`" and "`turn(int)`" methods

Birds

https://www.youtube.com/watch?v=hdh896hdKxU

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

Fish

https://www.youtube.com/watch?v=D6HdoIsLMFg

Flocking Simulations

- Flocking behaviour of animals can be simulated
- Programs which simulate flocking behaviour are called ***flocking simulators***
- Flocking simulators have applications in
  - Biology / Ethology
  - Animating animals in films
  - Adding realistic behaviour in video games
  - Autonomous robotics

Flocking Simulations

- Each agent is described by its ***state***, parameters such as:
  - Position (x)
  - Velocity (v)
  - Direction (θ)
  - Angular Velocity (ω)
  - etc

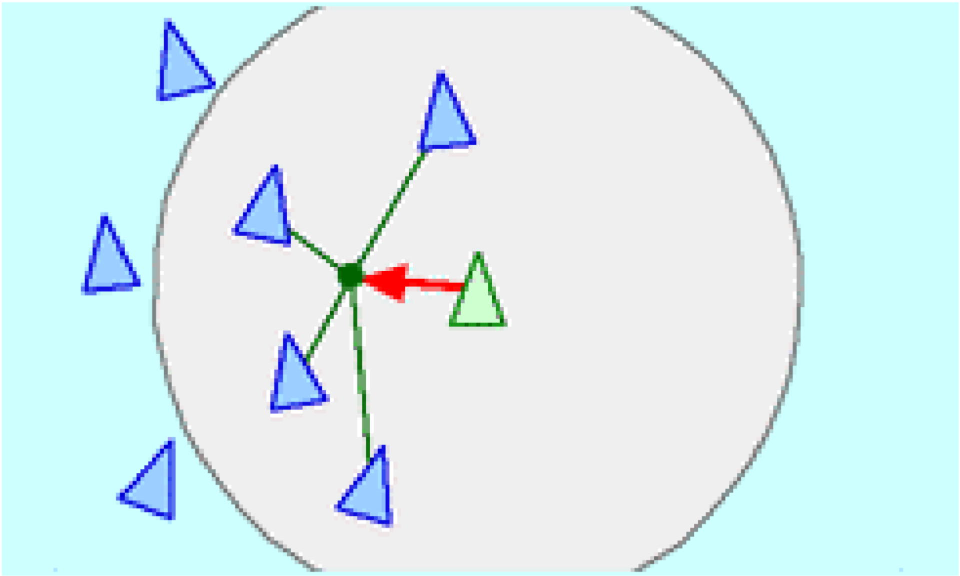
- The state of ***every*** agent in the flock is continuously updated
  - Based on their current state and the state of other agents within a given radius

- Exactly ***how*** to update an agent's state is determined by the flocking behaviour being simulated

Cohesion

Cohesion

- Causes agents to turn ***towards*** those around them
  - (*Long range attraction*)



Cohesion

- Take an agent with a direction θ (or angular velocity ω)

- Calculate the ***average position*** $\overline{x}$ of all agents within a radius r
  - $\overline{x}$ is "centre of mass"

- Calculate the angle through which the agent must turn θ_c to face ***towards*** $\overline{x}$

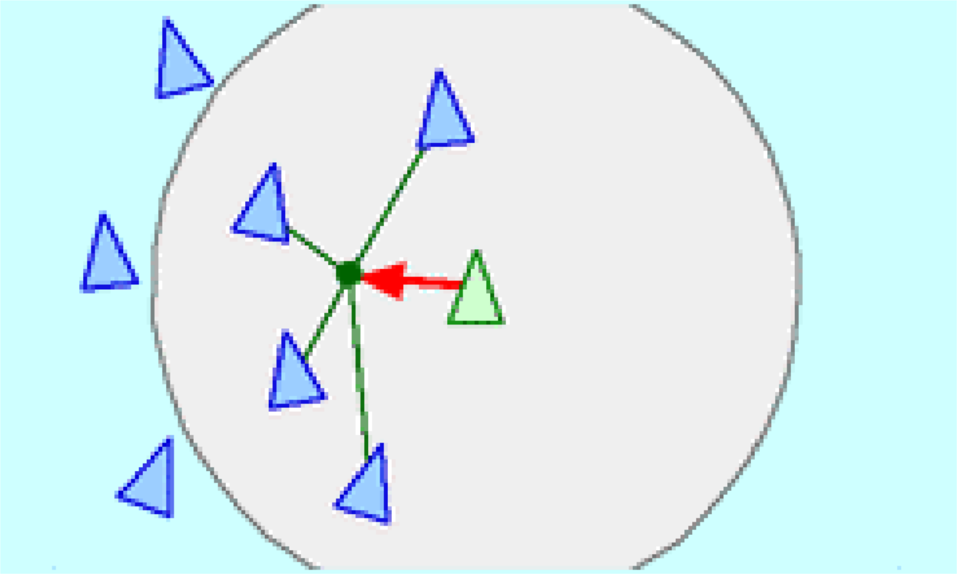
- Update the agent's ***direction*** (or angular velocity) using θ_c such that it turns towards $\overline{x}$

$$\theta = \theta + k_c \theta_c$$
$$(\omega = \omega + k_c \theta_c)$$

- k_c is used to vary the ***amount*** of cohesion behaviour

Cohesion

- If $k_c = 1$ then all agents ***always*** face ***towards*** the centre of mass of the other agents within a radius r
- If $k_c = 0.5$ then all agent's exhibit ***some*** cohesion
- If $k_c = 0$ then all agent's exhibit ***no*** cohesion

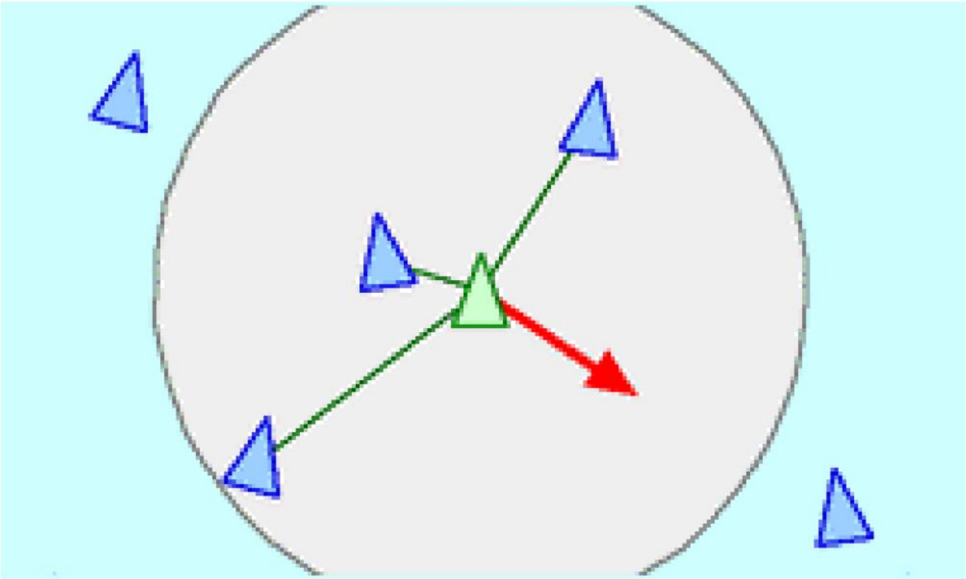


$$\theta = \theta + k_c \theta_c$$

Separation

Separation

- Is ***like*** the opposite of cohesion
  - But is ***not*** just the opposite
- Causes agents to turn ***away*** from those around them
  - (*Short range repulsion*)



Separation

- Take an agent with a direction θ (or angular velocity ω)
- Calculate the ***average position*** $\overline{x}$ of all agents within a radius r
- Calculate the angle through which the agent must turn θ_s to face ***away from*** $\overline{x}$
- Update the agent's ***direction*** (or angular velocity) using θ_s such that it turns away from $\overline{x}$

$$\theta = \theta + k_s \theta_s$$
$$(\omega = \omega + k_s \theta_s)$$

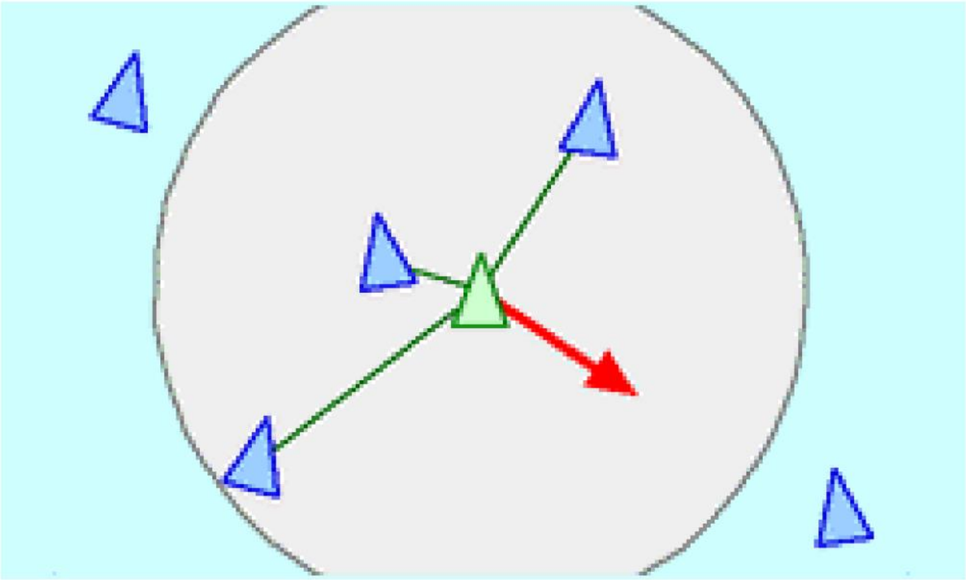
- k_s is used to vary the ***amount*** of separation behaviour

Separation

- If $k_s = 1$ then all agents ***always*** face ***away*** from the centre of mass of the other agents within a radius r

- If $k_s = 0.5$ then all agent's exhibit ***some*** separation

- If $k_s = 0$ then all agent's exhibit ***no*** separation

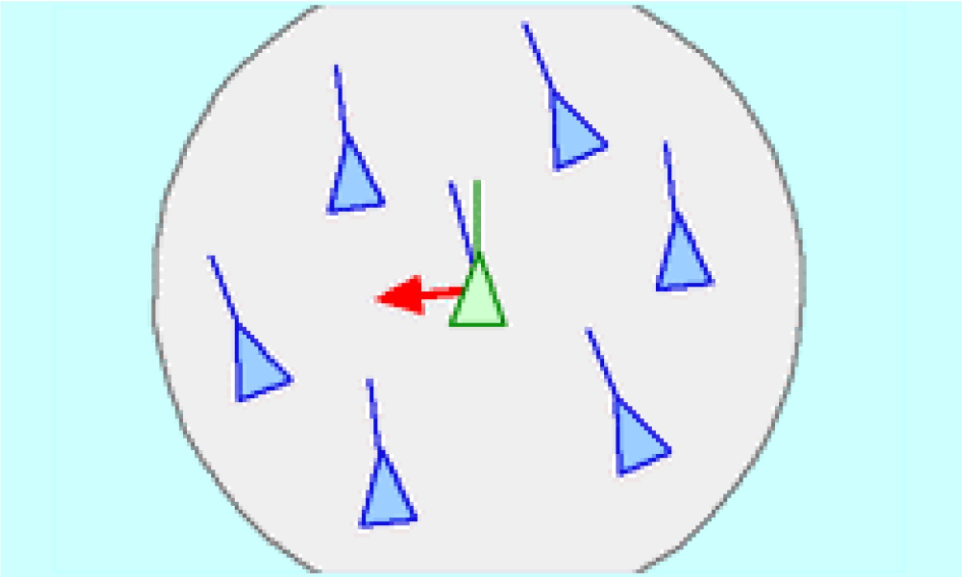


$$\theta = \theta + k_s \theta_s$$

Alignment

Alignment

- Causes agents to turn ***towards*** the ***direction*** of those around them



Alignment

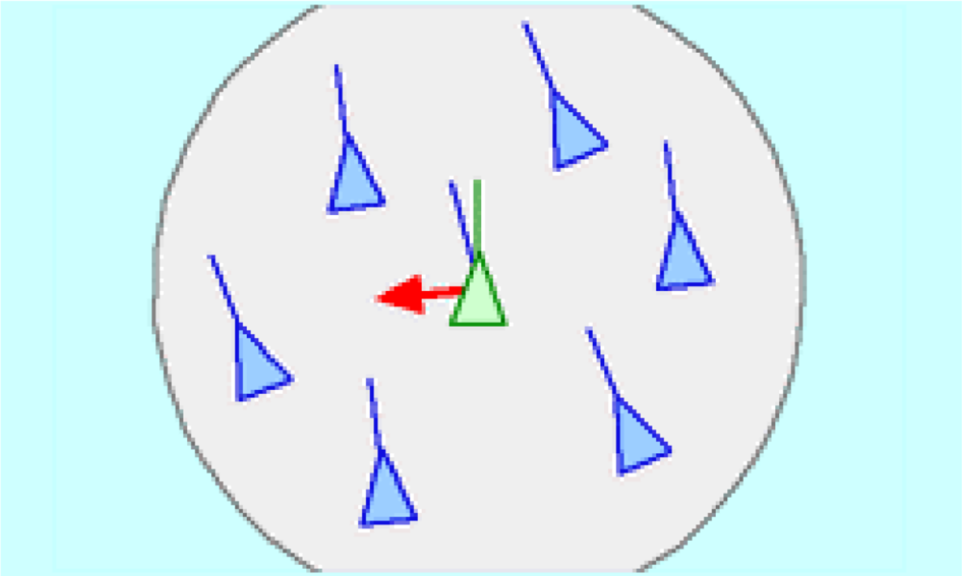
- Take an agent with a direction θ (or angular velocity ω)
- Calculate the ***average direction*** $\overline{\theta}$ of all agents within a radius r
- Calculate the angle through which the agent must turn θ_a to ***align with*** $\overline{\theta}$
- Update the agent's ***direction*** (or angular velocity) using θ_a such that it aligns with $\overline{\theta}$

$$\theta = \theta + k_a \theta_a$$
$$(\omega = \omega + k_a \theta_a)$$

- k_a is used to vary the ***amount*** of alignment behaviour

Alignment

- If $k_a = 1$ then all agents ***align perfectly*** with those within a radius r
- If $k_a = 0.5$ then all agent's exhibit ***some*** alignment
- If $k_a = 0$ then all agent's exhibit ***no*** alignment



$$\theta = \theta + k_a \theta_a$$

Combining Behaviours

Can create different ***flocking characteristics***

- Can combine the three simple algorithms
  - Cohesion
  - Separation
  - Alignment

- Then vary the values of
  - k_c
  - k_s
  - k_a
  - r (radius / radii)

- Can also choose whether to control angle, θ, or angular velocity, ω
  - Important: modifying direction ***not*** explicit movement

Useful Links

- One possible definitive resource for Boids
  - http://www.red3d.com/cwr/boids/
    - Maintained by their inventor Craig Reynolds

- Wikipedia high-level summary
  - https://en.wikipedia.org/wiki/Flocking

- Game development blog post
  - https://gamedevelopment.tutsplus.com/tutorials/3-simple-rules-of-flocking-behaviors-alignment-cohesion-and-separation--gamedev-3444
    - Written by Vijay Pemmaraju, also has informal, detailed introduction

Useful Programming / Debugging Tips / Techniques

What is the Problem with this Code?

```
public class SimpleTurtleProgram {
    // Declarations...
    private DynamicTurtle turtle;

    public SimpleTurtleProgram() {
        // GUI Stuff...
        // Create a turtle...
        DynamicTurtle turtle = new RandomTurtle(canvas, X_START, Y_START);
    }

    private void gameLoop() {
        while (continueRunning) {
            turtle.undraw();
            turtle.update(deltaTime);
            turtle.draw();
            Utils.pause(deltaTime);
        }
    }

    // Other stuff...
}
```

Dealing with Exceptions

```
Exception in thread "main" java.lang.NullPointerException
at turtle.Turtle.move(Turtle.java:74)
at turtle.Turtle.draw(Turtle.java:34)
at turtle.DynamicTurtle.<init>(DynamicTurtle.java:21)
at Lab5Turtle.<init>(Lab5Turtle.java:30)
at Lab5Turtle.main(Lab5Turtle.java:70)
```

Dealing with Exceptions

```
Exception in thread "AWT-EventQueue-0" java.lang.NullPointerException
at drawing.Canvas.paint(Canvas.java:73)
at java.desktop/javax.swing.JComponent.paintChildren(JComponent.java:907)
at java.desktop/javax.swing.JComponent.paint(JComponent.java:1083)
at java.desktop/javax.swing.JComponent.paintChildren(JComponent.java:907)
at java.desktop/javax.swing.JComponent.paint(JComponent.java:1083)
at at java.desktop/java.awt.Window.paint(Window.java:3940)
at java.desktop/javax.swing.RepaintManager$4.run(RepaintManager.java:876)
at java.desktop/javax.swing.RepaintManager$4.run(RepaintManager.java:848)
at java.base/java.security.AccessController.doPrivileged(Native Method)
at at java.desktop/java.awt.EventQueue.dispatchEventImpl(EventQueue.java:770)
at java.desktop/java.awt.EventQueue$4.run(EventQueue.java:721)
at java.desktop/java.awt.EventQueue$4.run(EventQueue.java:715)
at java.base/java.security.AccessController.doPrivileged(Native Method)
at java.desktop/java.awt.EventQueue.dispatchEvent(EventQueue.java:740)
at java.desktop/java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:109)
at java.desktop/java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:101)
at java.desktop/java.awt.EventDispatchThread.run(EventDispatchThread.java:90)
Exception in thread "main" java.lang.NullPointerException
at drawing.Canvas.removeMostRecentLine(Canvas.java:124)
at turtle.Turtle.undraw(Turtle.java:51)
at Lab5Turtle.runTurtleGame(Lab5Turtle.java:52)
at Lab5Turtle.<init>(Lab5Turtle.java:40)
at Lab5Turtle.main(Lab5Turtle.java:70)
```

Using the `static` Keyword

- We have seen ***objects*** that have their own ***fields*** and ***methods***
- Every **`Person`** object had its own **`name`** and **`age`** fields
- But what if we want to associate data with a class ?
  - Rather than any instance of that class (object)
  - i.e. what if we want to associate ***data*** with the “***blueprint***”
- For this we can use the **`static`** keyword

Don’t Use static

Code Conventions, Naming

- Have now seen various examples of classes, objects, fields, local variables
- Strong code convention in Java for naming these

- All class names ***MUST*** start with ***upper*** case initial
  - E.g. “MyClassName” not “myClassName”

- All fields / local variables (hence objects) ***MUST*** start with ***lower*** case initial
  - E.g. “myVariableName” not “MyVariableName”

- All constants ***MUST*** have all ***upper*** case with ***underscores***
  - E.g. “MY_CONSTANT” not “MYCONSTANT” or “My_Constant”

- Strong convention (so you ***will*** be penalised in assessment for violating this)
  - Can use Eclipse RMB “Refactor | Rename…” to easily correct

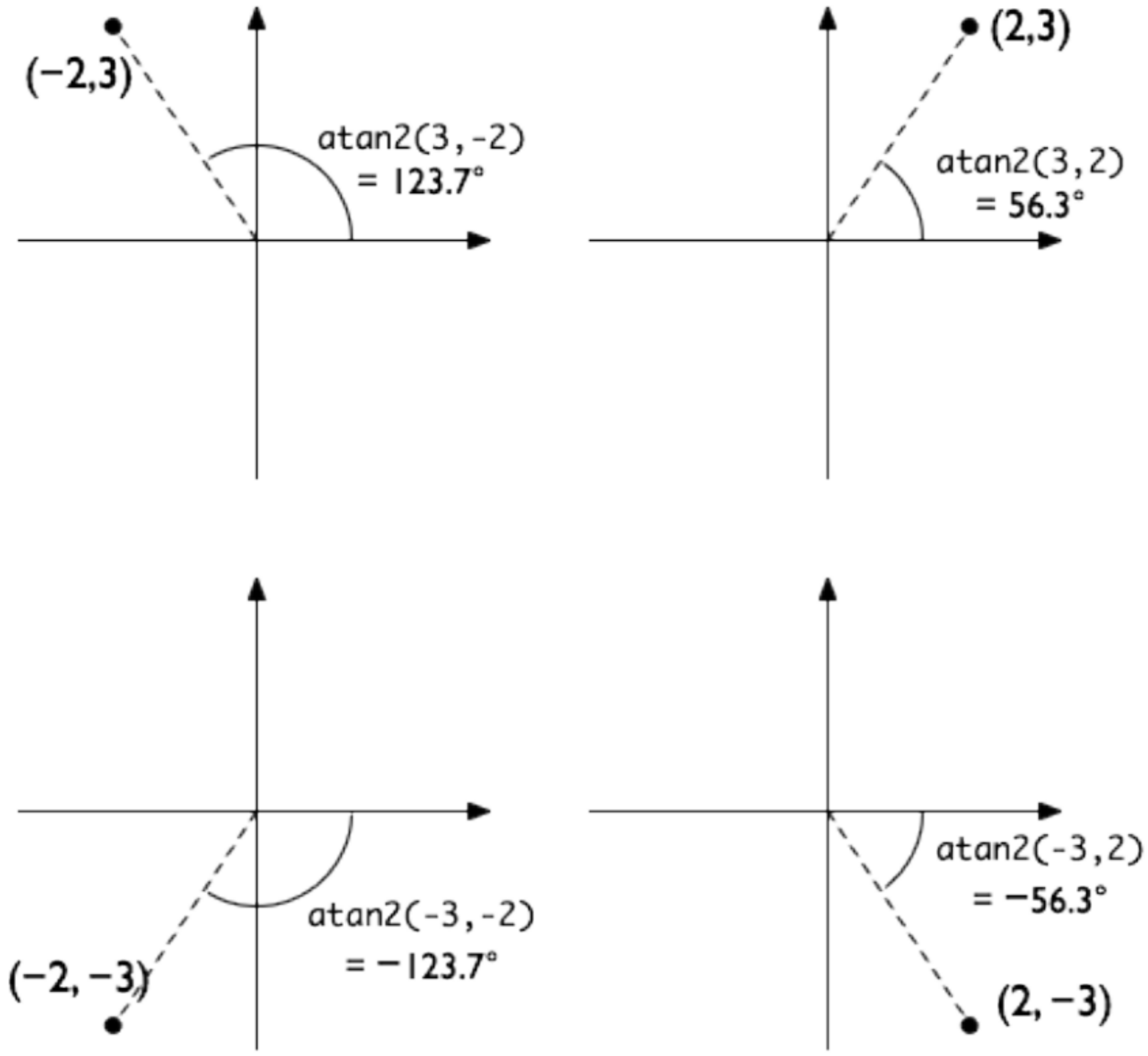
Queries – Programming

- Update time
  - For debugging, can decouple update and pause timings

- Debugging
  - Lots of parameters e.g. speed, position, etc
  - Use the debugger to monitor all parameters
    - Command line output useful but overwhelming
  - Could use: `if (DEBUG) { /* Print some stuff */ }`

- Static
  - Avoid
  - Decide which object a method is operating on

Helpful Hint: `Math.atan2`



Helpful Hint: `Math.atan2`

- To calculate the angle ***from*** the positive x-axis ***to*** the line between (x_1, y_1) and (x_2, y_2), the following can be used

```java
double x1 = 5;
double y1 = 10;

double x2 = 3;
double y2 = 7;

double xDiff = x2 - x1;
double yDiff = y2 - y1;

double angleInRadians = Math.atan2(yDiff, xDiff);
double angleInDegrees = Math.toDegrees(angleInRadians);

// Place in 0->360 range
if (angleInDegrees < 0)
	angleInDegrees = 360 - (-angleInDegrees);
```

Summary

- Flocking Algorithms
- Useful tips / techniques
  - What to avoid
  - How to fix