Java Programming

Assessment (2020/21)

Stuart Porter

Email: stuart.porter@york.ac.uk

1. Flocking Algorithms
2. Assessment

Flocking

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

Birds

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

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
  - Position (x)
  - Velocity (v)
  - Direction (θ)
  - Angular Velocity (ω)
  - etc

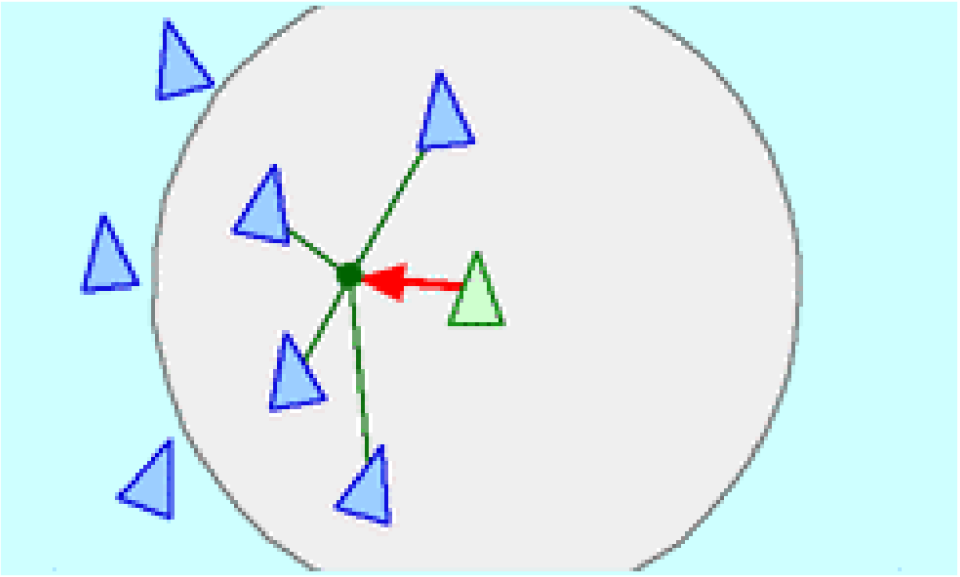
- The state of ***every*** agent in the flock is continuously updated
  - Based on their current state and the state of other agents within a given radius

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

Cohesion

Cohesion

- Causes agents to head to the centre of mass of those around them



Cohesion

- Take an agent with a direction θ (or angular velocity ω)
- Calculate the ***average position*** $\overline{x}$ of all the other 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 $\overline{x}$

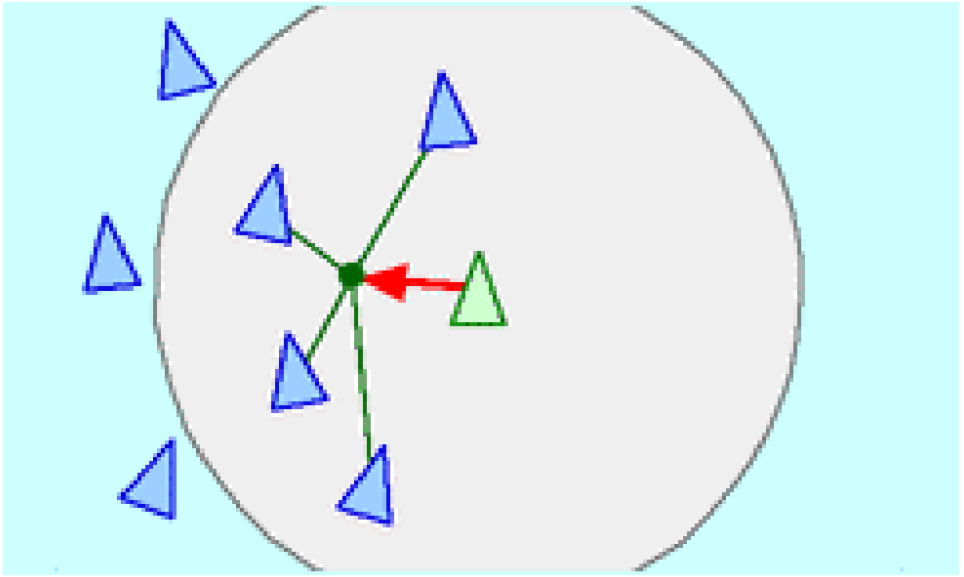
$$\theta = \theta + k_c\theta_c$$

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

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

Cohesion

- If $k_c = 1$ then all agent's ***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

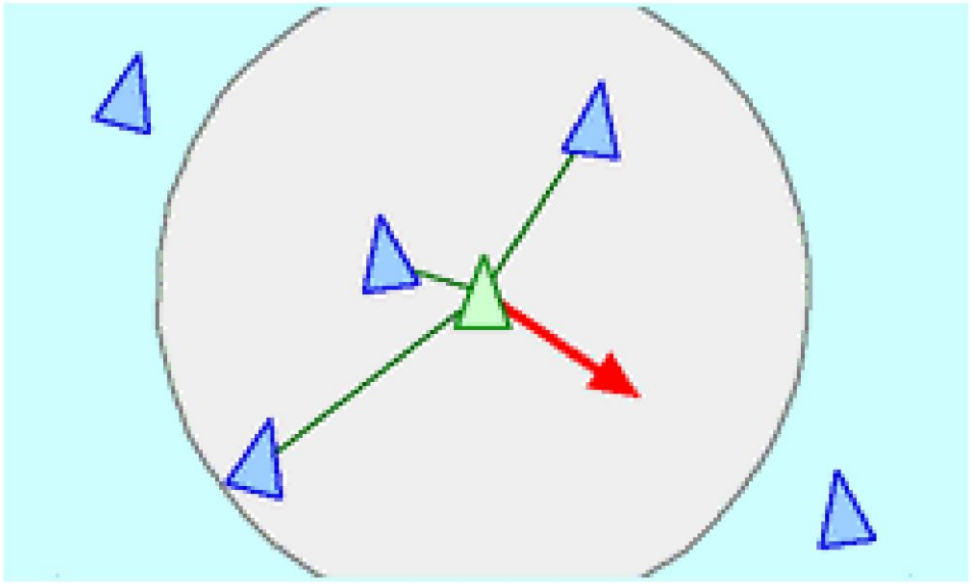


$$\theta = \theta + k_c\theta_c$$

Separation

Separation

- Can be considered the opposite of cohesion
- Causes agents to head away from those around it



Separation

- Take an agent with a direction θ (or angular velocity ω)
- Calculate the ***average position*** $\overline{x}$ of all the other 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 $\overline{x}$

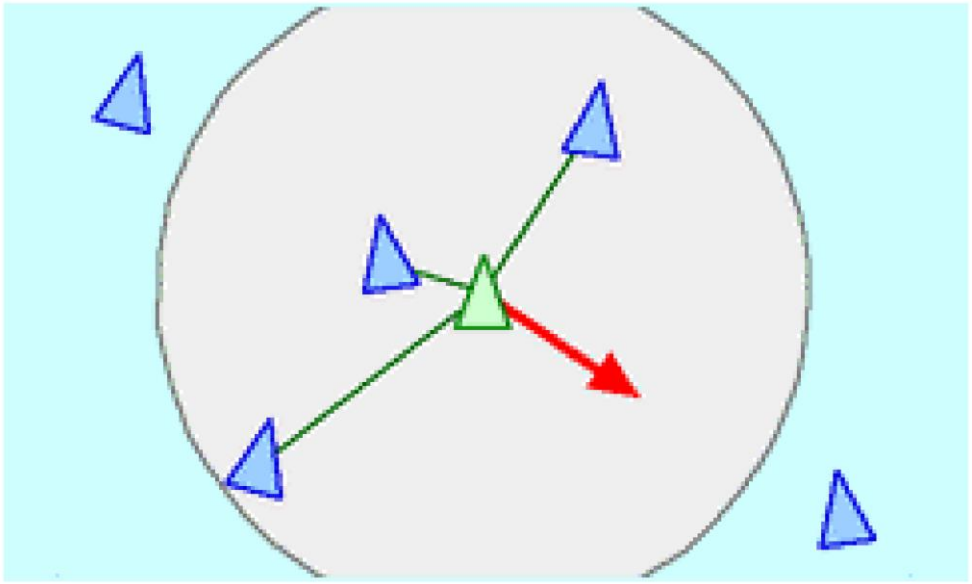
$$\theta = \theta + k_s\theta_s$$

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

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

Separation

- If $k_s = 1$ then all agent's ***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

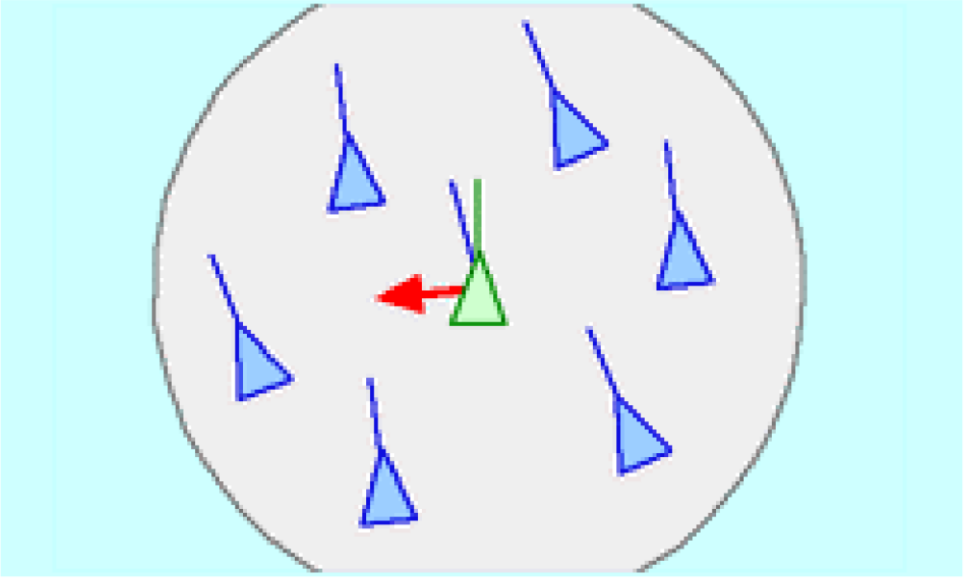


$$\theta = \theta + k_s\theta_s$$

Alignment

Alignment

- Causes an agent to align its direction with those around it



Alignment

- Take an agent with a direction θ (or angular velocity ω)
- Calculate the ***average direction*** $\overline{\theta}$ of all the other 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 $\overline{\theta}$

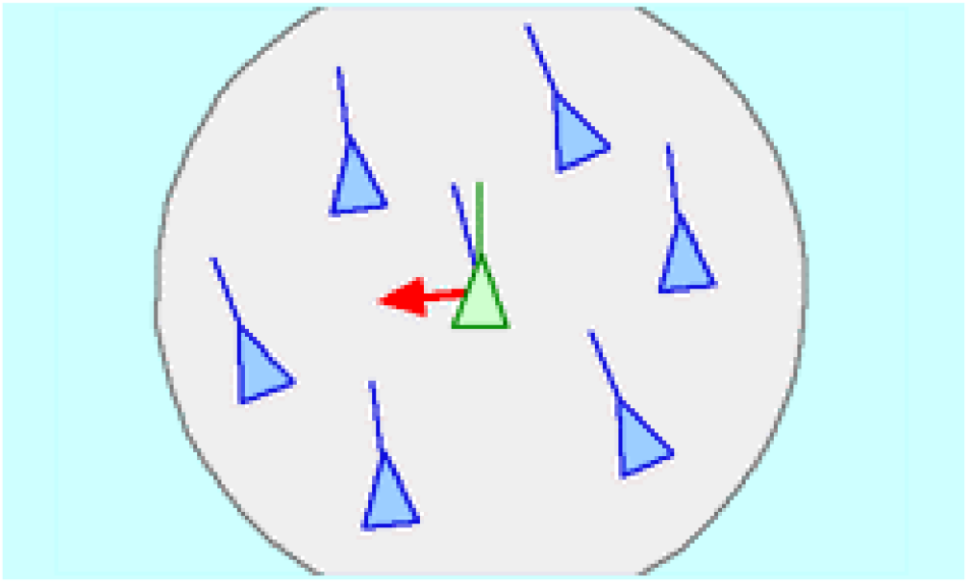
$$\theta = \theta + k_a\theta_a$$

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

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

Alignment

- If $k_a = 1$ then all agent's ***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

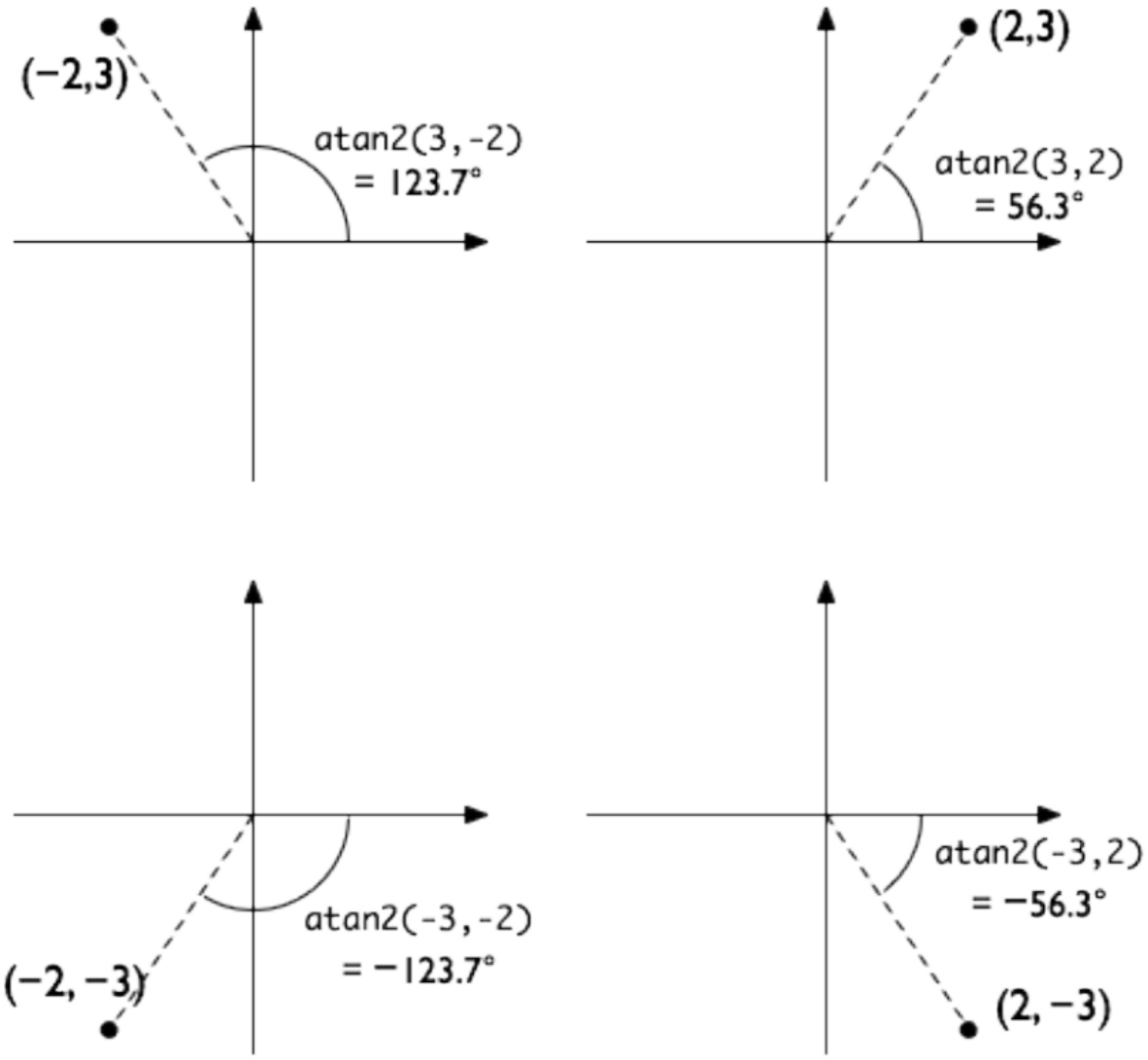
Can create different ***flocking characteristics***

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

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

- Can also choose whether to control angle, θ, or angular velocity, ω

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) to (x_2, y_2), the following can be used

```
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);
```

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_(behavior)

- 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

Conclusion

On to the assignment!

********** WARNING **********

Following is an ***overview*** of some of the main aspects of the ***assignment***.

BUT:

- The snippets of the assignment are ***NOT the whole assignment specification***
  - They are bits that are useful to talk about in this briefing
    - Aspects that we sometimes get questions about

So:

- You ***MUST*** refer to the formal, complete assignment specification when undertaking the assignment
  - On the module page

If:

- You think there is any contradiction between what is said here and the assignment specification, the specification is always correct.
  - But, in that case, please ask for clarification.

Assignment

SUMMARY DETAILS

This coursework for this module consists of the following two components:

Project Code contributes **65%** of the assessment for this module.

Individual Report contributes **35%** of the assessment for this module.

Clearly indicate your **Exam Number** on every separate piece of work submitted.

Submission is via the VLE module submission point. **The deadline is 12:00 noon on 13/5/2021, Summer Term, Week 4, Thursday.** Please try and submit early as any late submissions will be penalised. Assessment information including on late penalties is given in the Statement of Assessment.

Assignment

1 Task: Flocking Simulation

You are to design and implement a flocking simulation in Java using object-oriented programming techniques and Java libraries that have been introduced in lectures and labs. You **must** use directly the canvas and geometry classes developed in your “drawing” and “geometry” packages from the laboratories – failure to use these will result in an overall mark of **zero** for the “Project Code” component of the assignment; you may add to these classes as appropriate. You should also make use of the classes/code developed in the “turtle” package – you should modify these as appropriate for the new task.

Your application **must** simulate flocking behaviour in a two-dimensional world displayed graphically on the computer screen. Each individual in the flock should be able to move around and interact with others by following a set of simple rules that result in emergent flocking behaviour.

Your program **must** allow for the number of individuals and flocking parameters to be set and/or varied by the user through interacting directly with your application.

There is a rectangular obstacle of dimensions (dx=200, dy=100) pixels with its top-left corner at (250,150) on the canvas. This is impervious to any individuals – they cannot move into it or through it. You **must** implement this obstacle as part of the base specification.

To achieve high marks, the project should demonstrate good object-oriented design practices, such as inheritance and polymorphism, along with modular well-written code with good documentation and a well written and well-structured report.

To achieve high marks the program should be extended to include other complexities, such as control over simulation speed, obstacles, collision detection or other types of individuals for the flock to interact with (perhaps add predators into the simulation from which prey flee). However, a simple program that works is better than an unfinished, overly ambitious attempt that does not compile or execute correctly.

Assignment

You are to write your program in standard Java, version 11 and it must compile and execute correctly on the departmental laboratory computers using the Java command line tools or Eclipse.

Submission will be via the VLE in two parts:

1. **Individual Report**: a PDF report (no more than eight A4 pages in length)
2. **Project Code**: A zip file containing:
   - The Java source code for your program (fully commented and properly indented).
   - A simple text file named README.txt that explains which source files should be compiled and the entry point used to run your final program.

Assignment

The marks are broken down into the categories below (all marks in percentages). The **Project Code** will be marked under "Object-Oriented Design" and "Program Implementation", the **Individual Report** will be marked under "Report".

Appropriate Object-Oriented Design Practices	**(30)**
Use of multiple classes	6
Use of composition within your classes	4
Use of inheritance within your class structure	4
Use of polymorphism when referring to your objects	4
Use of interfaces within your class structure	4
Use of inner and/or anonymous classes	4
Use of public, private and protected	4

Program Implementation	**(35)**
Compiles, executes and runs without error	5
Execution (how well does the actual program work)	6
Completeness (how complete is the submitted program, how functional)	4
Modular, non-duplicated code (short, reusable, single-purpose methods)	4
Program documentation (commenting, formatting, suitable names of variables, etc)	4
Clarity and simplicity of program (good program structure, logical flow)	4
Extended functionality (going beyond the basic specification)	8

Assignment

Report	**(35)**
Explanation of the flocking algorithm you used	5
Explanation of design	10
Explanation of program implementation	10
Summary of test procedures and results	5
Readability and formatting	5

The End!