THE UNIVERSITY *of York*
*Department of Electronics*

# Java programming assignment

# Flocking simulator

**Y3504462**

May 2015

# Contents

# 1 Introduction

The goal of this assignment is to use an Object Oriented Programming language, in this case Java, to create a program that simulates flocking birds or a school of fish. This should be done using object oriented techniques like inheritance and polymorphism. The program should also allow for the user to select how many birds they want to be in the simulation at once. In addition the program can implement other features such as predators and obstacles that the birds have to avoid.

The simulator builds on Boids originally made by Craig Reynolds in 1986. His version uses three simple rules: Cohesion, alignment, and separation. Cohesion is the rule that make the boids stick together. Separation makes sure that all the boids dont just become one massive blob, like they would if the cohesion rule was the only one in effect.The final rule, alignment, steers all the boids in the "neighbourhood" towards the same direction. For the duration of this report I will refer to the boids as birds to fit my simulators theme.

After the contents page the report then goes through the specifications of the task, and then explains the conceptual design of the simulator. After that the implementation goes through my decision making and problems I ran into while writing the code, before ending with a conclusion of my result.

# 2 Specification

The program has to simulate creatures on a two dimensional plane that all move around and interact with each other using simple rules. These rules should allow the creatures to flock together, and therefore act as one larger unit rather than many small ones.

The rules and graphics all needs to be made using the Java programming language, and has to use the object oriented techniques taught to us in the labs and lectures. It should also be possible for the user to choose conditions for the simulation like how many creatures there are and where they start out on the plane. The program should also be able to handle errors well, and demonstrate test code used while the code was being written to test different aspects of the code.

In addition the program can also have additional features like predators, walls, or other obstacles that the creatures have to avoid while still sticking to the rest of the flock.

# 3 Design

To complete the task the first thing I would need is something to draw the birds on. The easiest way to accomplish this will be to use the canvas class made by Stuart Lacy that's been provided for us in the labs. This class does however contain a few methods and variables that aren't needed, so I can remove things like the penIsUp penIsDown methods. Secondly I will need a class for drawing the birds, and a class for making them move. Because of their similarities, building on the Turtle and DynamicTurtle classes from the labs is a good springboard. The DynamicTurtle class inherits from the Turtle class extending it to allow to take a set speed and a way of calculating its new position based on the speed and a given time. Using this formula:

$$\text{Distance} = \text{Speed} \times \text{Time}$$

Furthermore to allow for user input I should prompt the user with dialogue boxes. I will also need some code to prevent the birds from exceeding the x or y limits of the canvas. This requires me to have getters for the birds' positions, and setters to redirect them when they're out of bounds.

To implement the flocking each bird has to have a neighbourhood where the three rules mentioned above apply. The rule of cohesion will have to take the average position of all the birds in the neighbourhood and set that as the ideal location for the birds. Separation will work on a smaller distance from each bird's center to counteract the cohesion to a certain extent. The alignment will work by taking the average angle of the birds in the neighbourhood and steering them all towards that angle. To avoid too sharp a movement, the birds should adjust their own angles and position with a fraction of the ideal, and not the full amount.

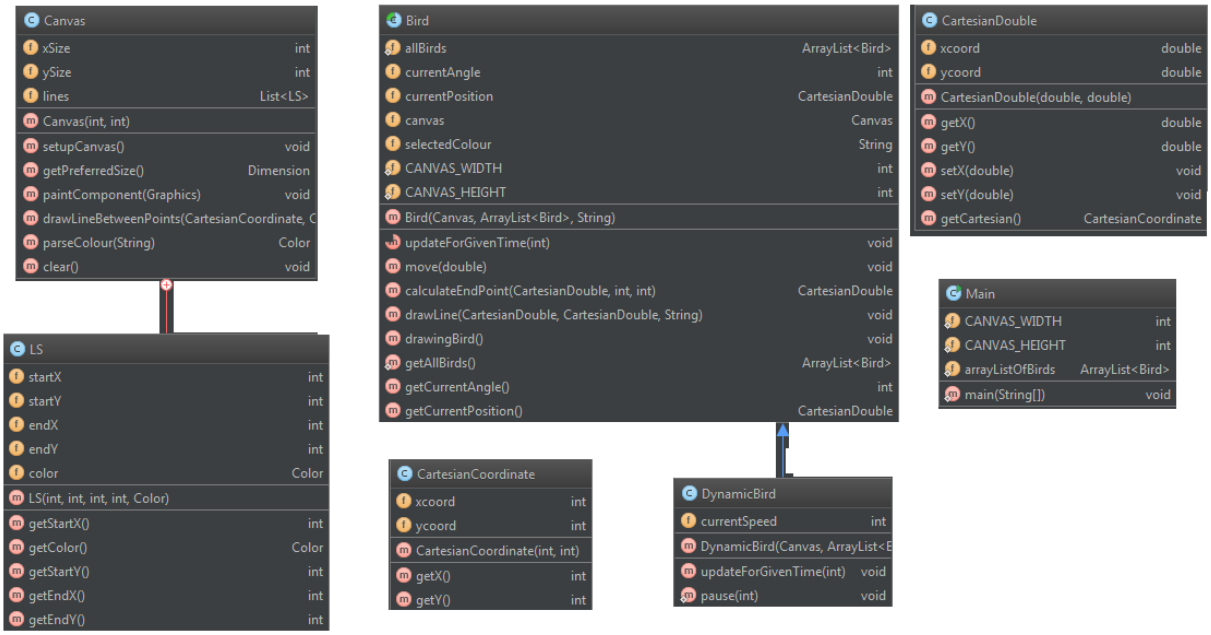


Figure 1: Class Diagram for the implemented system

A full class diagram can be found in the Implementation part of this report.

# 4 Implementation

## 4.1 Overview of the simulator

When the simulator starts the user is greeted with a prompt and a canvas in the background. The canvas has a resolution of 1280x720 pixels, the standard aspect ratio for most modern monitors. The prompt has the text: Please enter the number of birds you want in the simulation (integer):.

The user can now type in the desired number of birds they want using a keyboard. Any integer value will work, and typing in a string or a symbol will just re-initialise the prompt and ask the user again. After entering an int the and pressing the enter key or clicking ok a new prompt pops up that says: Please enter the colour you would like the birds to have: You can choose between blue, red, yellow, orange, cyan, magenta, black, and green". In this prompt the user can type in a string with their desired colour. The string is not case sensitive so any combination of capital and lowercase letters will work as long as there are no breaks before, after, or in the word. Failing to type in an accepted colour will result in the command line printing out an error message saying:

"Unknown colour YourStringHere, defaulting to black."

Some examples of accepted strings would be:

- "rEd"
- "BluE"
- "cyan"

These on the other hand, are all invalid:

- "re d"
- "blue "
- " cyan"

and selecting the colour black for all the birds. If you want to change the number of birds, or their colour, you will have to relaunch the application.

Once both the prompt have received an accepted value the chosen number of M shaped birds will load on to a white canvas with random locations((x,y) values) and angles assigned to them. They all have the same speed. When a bird reaches one of the canvas maximum x or y values the bird will wrap around, and spawn again on the opposite side of the canvas from where it left with the same speed and angle as before. FLOCKING INFO HERE To exit out of the simulator you can press the X in the top right corner on Windows, or in the top left on OSX.

## 4.2 Reasoning

## 4.3 Main:

I wanted to keep my main simple, so most of the functionality is found in the other classes, mainly Bird.java. The first thing I set is the canvas size which I wanted to have a 16:9 aspect ratio. I chose 1280x720 pixels because its relatively big, but will fit nicely on most screens. To handle the birds that will be in the simulation I set up an ArrayList. This is done so that the amount of elements can be changed dynamically. For the user input the easiest way I thought of to handle it was to prompt the user with some boxes they could fill in. A problem I had with just asking the user for an integer was that the box would just disappear and the simulation not run if the user didn't listen to the instructions.

```
userInputBirds = JOptionPane.showInputDialog(null, "Please enter the number
of birds you want in the simulation:\n(integer)");
```

I chose to solve this with a do while loop that reprompts the user until they type in an int.

```
do {
  userInputBirds = JOptionPane.showInputDialog(null, "Please enter the
  number of birds you want in the simulation:\n(integer)");
} while (!userInputBirds.matches(".*\\d.*"));
```

- The colour input already has error handling in the Canvas class and defaults to black if the input isnt accepted.
- To generate the wanted number of birds on the canvas I use a simple for loop taking the parsed string as its parameter.
- To keep the canvas updating for the duration of the simulation I use a while loop thats always true.
- The while loop clears and redraws the birds in their new position for every run through.
- Inside the while loop I put in a for loop to be able to update each birds position on the canvas.
- DynamicBird.pause(25) slows down the whole simulation by 25 milliseconds, a time I found fitting for giving the illusion of movement while still moving relatively fast.

## 4.4 Bird and DynamicBird:

Bird uses the same constants as the Main, but it also adds variables relevant to a static bird like its angle, current position and its colour. The constructor is the same that as the one the Turtle class we used in the lab was, except that it has been modified to take random values for each instantiated bird's starting position and angle, and a selected colour. To get the random positions I used the Math.random() method from the Java standard library and multiplied the values I got with the canvas width and height to get the random x and y coordinates respectively. I used the same method for the angle, but I instead multiplied the number with 360 to create a random angle between 0 and 360 degrees.

I made an abstract version of updateForGivenTime in Bird so because I planned on making multiple types of birds, predators, that would potentially need to use slightly different versions of the same code. In short making the code more flexible.

The move method handles where each bird is, and where it should be going, as well as drawing the bird in the new position that calculateEndPoint works out. In early builds the wrap around and canvas border code was in the main class, but after creating multiple birds I found it best to move it to the Bird class so that everything was handled separately. The series of if statements ask whether or not the bird is still going to be on the canvas for the next frame, and if not it determines where it should override its position to.

As mentioned determining where the bird should move to next is handled by the move method which remains unchanged from the original Turtle code we used in labs. The drawingBird method has also stayed the same except for the actual shape that gets drawn on the canvas. I chose to go with the bird theme and make the boids look more like the classic 'M' shape birds in flight have.

There are some traces of code that I would have liked to implement, but didn't have the knowledge/time to do. For example the bird constructor takes ArrayList as a parameter:

```
public Bird(Canvas canvas, ArrayList<Bird> birds, String colour)
```

The reason for this is because it would have made the creation of neighbourhoods possible, and therefore is essential for the actual flocking. This however proved to be the most challenging aspect of the task for me. I was not able to figure out how to implement my ideas for the flocking. The code therefore carries some traces of this with some line commented out that would have be necessary for the three rules to work, but that have no place in the current code, due to me not being able to code the rest. Examples of this is the getters at the bottom of the Bird class, and some of the parameters mentioned above.

### 4.5 CartesianCoordinate, CartesianDouble and the Canvas:

CartesianCoordinate, CartesianDouble, and the Canvas classes are pretty similar to the ones that we used in labs. For the first mentioned class I've added a getter and a setter for each x and y. CartesianCoordinate was written by me in labs and has no changes from then. The canvas however has been optimised for the purpose, where I've removed quite a few unused methods that weren't needed for this project. I also had some issues with the birds flickering on the canvas. I managed to fix this by avoiding calling the repaint method multiple times, and instead moving it to it's own method and calling it once in the main.

## 5 Conclusion

This report has looked briefly at the origin of Boids, and the logic behind their implementation of their simplest form using the three rules: Alignment, cohesion, and separation. In terms of the specifications my code has some holes, the most obvious being the lack of flocking rules. This would of course be the main focus for further work on the simulator. There also lacks support for file reading and random starting positions. These two points are also something I would look into if I had more time.

Despite having these holes, the points of the specifications that are met, are met well to very well. The changes to the canvas and the streamlining of the code makes it so that everything runs really smoothly (at least on my own computer). I'm also happy with how well the wrap around and user inputs are working. There is also decent error handling, where non int input will be ignored. As far as error handling goes I would probably look into reprompting the user if they input a colour string that has a space in it instead of defaulting to black, but that would not be a priority.

All in all the simulator is missing some important features, but it does what it does really well.

## 6 References

http://www.red3d.com/cwr/boids/, visited 23.04.2015, 13:50. Author Craig Reynolds.