CS22510 assignment 1
Aphids and ladybugs

Fred Labrosse Neal Snooke

February 17, 2015

1 Introduction

This is the first of two assignments in CS22510. It will count for 40% of the total marks for the module. You should therefore spend roughly 40 hours of work on this assignment. **The deadline for this assignment is Thursday 19th March 2015 at 4pm and must be submitted via blackboard.** The second assignment is worth 20% of the module's mark.

The first assignment is about implementing a given task in C++ while the second assignment is about writing a short essay discussing the merits of the three languages covered in the module with respect to the task, highlighting where each language would have helped or hindered in the implementation of the solution to the task.

2 The task

The aim of the assignment is to build a program that simulates a contrived world where creatures live and die. There will be two types of creatures, namely *aphids* and *ladybugs* (or *ladybirds* in the UK, although they don't really look like birds!). They will live in a discrete world where positions are integers. They will move and interact with other creatures following specific rules (see below). The input to your program will be a number of configuration files that specifies the initial positions of aphids and ladybugs as well as parameters of the task. The output will be an animation of the simulation.

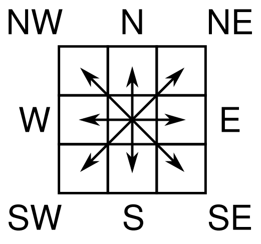
Given that the assignment is about programming in C++, design an object oriented solution to the problem and justify your choices in the submitted document (see below).

2.1 Overall simulation

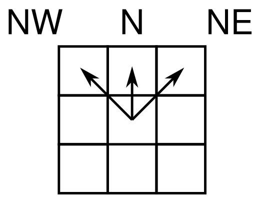
The world is assumed to be a 2D grid of cells which may contain 0 or several of each creature type. The cells in themselves do not do anything. However, it may be advantageous to represent them in a way that information can be attached to them. Section 2.5 gives more details. Cells that are not at the edge of the world have eight neighbours. Cells that are at the edge of the world have fewer (five for the edge cells, three for the corner cells).

The simulation will be turn based, i.e. each creature is updated in turn by a "manager" in a way that does not require the manager to know anything about the rules specific to creature types. All the manager needs to be able to do is maintain a list of the creatures currently alive, `update()` each creature in turn, kill (`delete`) a creature, create (`new`) a new creature and move one from one cell to another, when told to. The simulation finishes when there are no more creatures to process (or when the program is interrupted obviously).

Creatures all share a common interface. They need to be able to say in which cell of the world they currently are, need to `update()` themselves (following the rules given below). They also have to tell the manager where they move (any of the eight possibilities from where they are, but see Section 2.2.1 for more details), which creature they have killed (if any) and if a new creature needs to be created.



(a) Possible aphid movements



(b) Possible ladybug movements for a preferred direction to the north

Figure 1: Motion for the two types of creature

2.2 The rules

The creatures will obey a number of rules that specify how they move and interact with other creatures. When asked to update, a creature will first possibly move and then possibly interact with any creature(s) that may be in the cell it currently is in.

2.2.1 Motion

At each update, creatures move with a given probability p_m. If a creature decides that it has to move, they do so using the following rules.

Aphids are small and nimble so they can change direction easily. Their motion is such that they randomly move into any one of the eight neighbouring cells of the cell they currently are in, Figure 1a.

Ladybugs are larger and so have a tendency to move in straighter lines than aphids. They will therefore have a preferred direction of motion (one of four, which could be called North, South, West, East). For the given direction, three choices are possible, selected at random, Figure 1b. At each update, the direction may change before motion occurs with a given probability (p_{cd}) to one of the four directions selected at random. If any of the creatures tries to move out of the world, its motion is mirrored by, for example, moving down instead of up when the creature tries to "escape" at the top.

2.2.2 Interactions

When two or more creatures are present in any given cell and the same time, they interact in a manner that depends on creature types and numbers. Each creature first interacts with creatures of the other type (combat), and then with creatures of the same type (procreation).

If a ladybug arrives in a cell where there is one (or more) aphid(s), then it will attack the aphid. This happens only once, i.e. a ladybug only attacks one aphid. The ladybug will kill the aphid with a given probability p_{la}.

If an aphid arrives in a cell occupied by a ladybug, then it will attack the ladybug (not very realistic, but...). This happens only once, i.e. an aphid only directly attacks one ladybug, but could help other aphids in their attack (see below). The aphid will kill the ladybug with a probability p_{al} that is calculated based on the number of aphids in the cell:

$$p_{al} = p_{al_b} + n \times p_{al_n},$$

where p_{al_b} is the given base probability for an aphid to kill a ladybug, n is the number of aphids present in the cell and p_{al_n} is the given additional probability per aphid present in the cell[1].

[1]Note that p_{al} may end up being greater than 1.0 and is therefore not strictly speaking a probability, but we will overlook this for the assignment.

Following the combat stage, if a creature is in a cell where there is at least one other creature of the same type, it will give birth to a single other creature of the same type in the same cell, with a given probability p_b. Note that this happens only once per creature per update.

2.3 Animation

The simulation will output after each update of all the creatures the current state of the world in pseudo graphical representation. This will display in the console a 2D grid each cell containing two digits, the first giving the number of aphids, the second the number of ladybugs. If there are no creatures of one type, output a space (‘␣’) instead of the digit 0. If there are more than 9 creatures of one type, output the character ‘~’ at the corresponding position.

You may have to introduce a pause in the animation (for example with a `sleep()`) to be able to see anything.

2.4 Input files

Your program will have to read configuration files, described here. To enhance modularity of the solution, there will be several types of configuration file for the various aspects of the program.

Each line of the configuration files will contain the definition of one aspect, using one or several numbers.

In tables 1, 2 and 3, the lines in the file are in sequence and the column “Lines” in the tables gives how many lines of each type there are for each entry. The “Format” column gives the format of each line. The “Description” gives what is on each line.

Table 1: Manager configuration description

Lines	Format	Description
1	2 integers	Horizontal and vertical size of the grid
1	1 integer	Number a of aphids
a	2 integers	Coordinates of the corresponding aphid in the grid
1	1 integer	Number b of ladybugs
b	2 integers	Coordinates of the corresponding ladybug in the grid

Table 2: Aphids configuration description

Lines	Format	Description
1	1 float	Probability for the creature to move p_m, Section 2.2.1
1	1 float	Base probability for an aphid to kill a ladybug p_{al_b}, Section 2.2.2
1	1 float	Additional probability per aphid present in the cell in helping to kill a ladybug p_{al_n}, Section 2.2.2
1	1 float	Probability for two aphids to give birth when they meet in a cell p_b, Section 2.2.2

Table 3: Ladybugs configuration description

Lines	Format	Description
1	1 float	Probability for the creature to move p_m, Section 2.2.1
1	1 float	Probability for a ladybug to change direction p_{cd}, Section 2.2.1
1	1 float	Probability for a ladybug to kill an aphid p_{la}, Section 2.2.2
1	1 float	Probability for two ladybugs to give birth when they meet in a cell p_b, Section 2.2.2

```
10 10
5
3 5
4 8
2 9
1 6
1 9
4
5 9
1 1
3 8
9 2
```

(a) Manager

```
0.7
0.2
0.1
0.4
```

(b) Aphids

```
0.7
0.2
0.4
0.2
```

(c) Ladybugs

Figure 2: Configuration file examples

Each configuration file is to be read by the appropriate class. You can assume a fixed file name. If the file is not present, use default values given in Figure 2.

2.5 Various considerations

The simulation is inherently object oriented and almost everything can be represented as a class. In some cases however, there might be a compromise between the OO approach, memory usage and running efficiency. This is particularly true of the way grid cells are handled. Which ever design you use, you should briefly justify it in the document you submit as part of this assignment (see Section 3).

Many actions of the simulation rely on probabilities and random operations. When a probability is needed, one approach is to get a random number between 0.0 and 1.0 and compare it to the probability. The function `rand()` returns a random number between 0 and `RAND_MAX` so `static_cast<double>(rand())/RAND_MAX` returns a `double` between 0.0 and 1.0. To get one out of 8 random choices (for example for the aphid motion) you can multiply that number by 8 and `round()` it. C++11 has a number of facilities to handle random number generation in its `random` include file.

Killing or dying results in deleting an object. This implies making stray pointers possibly point to invalid memory (memory that has been de-allocated) and may invalidate iterators, depending on data structures used. This needs attention. A sensible approach is to decide what in your program is the owner of what and only owners can delete what they own.

2.6 Additional features

To possibly achieve a mark higher than 70%, you will have to implement additional features. Possible ones are given below, but others are possible.

You might notice that the simulation never finishes and that depending on the various probabilities, one or the other populations thrives. You might want to implement a stopping criterion that stops the simulation should one of the two populations die. This is more difficult than it appears if you want to keep a good object orientated approach!

Creatures need food to live, and progressively die if they don't get food. You could simulate this by adding to creatures a "life" attribute that decays as the simulation goes and increases when the creature kills another creature and/or when it finds food in the world (another type of object that could be added). When the "life" reaches 0, then the creature dies.

Another interesting feature is to consider that the world might not be a regular square grid but cells could, for example, have six or eight sides, not four, and that the cells at the edge of the worl could be neighbours of the cells at the other end of the world, i.e. implementing a non-flat world.

The animation as described in Section 2.3 is rather crude and not very effective at displaying what is happening in the simulation. You could develop a graphical interface to produce a nice display of the simulation.

3 What to hand in

The hand in must be done using Blackboard. You must hand in the following files:

- **A README file** that describes what is what. In particular this must give the names of the various files you hand in. This must also describe how to build your code, including a mention of standard libraries/packages and the versions you used.
- **Your program**. Make sure that all the necessary files are included. Do not include standard libraries.
- **An example of the output**. This must be the "animation" saved in a text file. It is likely that you will have to interrupt the animation. Do so when you see that one of the populations is either dead or exploding.
- **A screencast**, showing:
  1. the building of your code, with all errors (hopefully none) and warnings (hopefully not too many);
  2. the running of your code, clearly showing the output.

  The screencast should probably include voice to explain what is going on. Before compiling your code for the screencast, make sure you clean it so that the compilation does do something. Also, make sure you choose the correct compilation options so that warnings are given. The screencast should only show a couple of minutes of the simulation.
- **A document** (maximum two pages long) that explains what you have done and justifies your design.

4 Assessment

This assignment will be assessed using the Assessment Criteria for Development. This can be found in the Student Handbook, Appendix AA, at `http://www.aber.ac.uk/compsci/Dept/Teaching/Handbook/AppendixAA.pdf`. The usual requirements for coding projects apply, namely that programs should be well commented with comments that add real value and do not just, in essence, duplicate code. Programs should have good layout and must use meaningful names for variables, classes and other identifiers.

More specifically, the detailed marking scheme for this particular assignment is given in Table 4.

5 Final remarks

You are more than welcome to use existing, non standard libraries to help you in your task. This could include generic libraries such as boost. Make sure you acknowledge the use of these libraries in the documentation of your code.

This is an individual assignment and it must therefore be completed as a one person effort by the student submitting the work as their own. In particular, your attention is drawn to Section 5 of

Table 4: Assessed aspects of your work and their value

What	Value
Code layout	5%
Identifier names	5%
Comments	5%
Readability	5%
Documentation	10%
Program quality and design	15%
Program compilation	5%
Program success	20%
Additional features	30%
Total	100%

the student handbook[2]. If in doubt, make sure that you properly acknowledge material (including code that is reused or closely derived from others) in order to avoid any suspicion that you are trying to cheat and ask for advice if you are not sure. By submitting your work to Blackboard, you agree to the statement about the Declaration of Originality. This statement is the same as that shown on the Anonymous Marking Sheet Cover. You do not need to submit a separate declaration form.

6 Final, final remarks

There is no specific environment on which you need to run your code, or C++ specific standard to use. Just make sure you specify this information in the README file.

The files posted on Blackboard have Unix line endings. You can either use tools like `unix2dos` to convert the files or tell which ever `readline()` function you use to use the correct line termination.

If you have any more questions, please contact Fred Labrosse (`ffl@aber.ac.uk`) or Neal Snooke (`nns@aber.ac.uk`).

[2]`http://www.aber.ac.uk/compsci/Dept/Teaching/Handbook/`