CS-2303, System Programming Concepts, A-term 2017

Project 6 (50 points)
Assigned: Tuesday, February 21, 2017
Due: Thursday, March 2, 2017, 6:00 PM

Programming Assignment #6 — Polymorphism

Abstract

Implement a simple 2D predator-prey simulation using derived classes and virtual functions.

Outcomes

After successfully completing this assignment, you should be able to:–

- Design an abstract base class and several derived classes from the base class.
- Design one or more virtual functions in the base class and provide concrete implementations of them in the derived classes.
- Enumerate the objects and invoke a method on each one.

Before Starting

Read Chapters 14 and 15 of *Absolute C++*, which introduce inheritance and polymorphism, respectively. This assignment is adapted from Programming Project 3 of Chapter 15. Depending upon your approach to this assignment, you may also find the following useful:– §7.3 about vectors and §17.3 about iterators.

Teams: You may optionally work in two-person teams. To register your team in *Canvas*, please send an e-mail to cs2303-staff@cs.wpi.edu. Your team should make one joint submission, and the names of both team members must be on each file.

Existing teams from Programming Assignment #5 will be carried over to this assignment. If you wish to dissolve or change teams, please send e-mail to the same address.

This Assignment

This program involves a simulation of a grid of *n-by-n* squares, some of which may be occupied by *organisms*. There are two kinds of *organisms* — *doodlebugs* (the predators) and *ants* (the prey). Only one organism may occupy a cell at a time. Time is simulated in steps. Each organism attempts to perform some action every step. No action may cause an organism to move off the edges of the grid.

Ants behave as follows:–

- *Move.* For every step, each ant enumerates its adjacent cells — *up*, *down*, *left*, or *right* — and randomly selects an unoccupied one that is on the grid. If all adjacent cells are occupied or off the edges of the grid, the ant does not move but rather remains in its current location. [1]
- *Breed.* If an ant survives for at least three time steps, at the end of the third time step (i.e., after moving) the ant gives birth to a new ant in an adjacent cell (i.e., *up*, *down*, *left*, or *right*). If

[1] You will have to establish a systematic way of ordering the actions of the ants. Obviously, a previous ant could move into the only possible space available to another ant, thereby preventing the second one from moving.

more than one empty cell is available, it chooses one at random. If no empty cell is available, no birth occurs.[2] Once an offspring is produced, an ant cannot produce another offspring until it has survived three additional steps.[3]

Doodlebugs behave as follows:–

- *Move.* For every time step, each doodlebug moves to an adjacent cell containing an ant and eats that ant. If more than one adjacent cell contains an ant, one is chosen at random. The ant that was eaten is removed from the grid. If no adjacent cell (i.e., up, down, left, or right) contains an ant, the doodlebug moves according to the same rules as ants. Note that a doodlebug cannot eat another doodlebug.
- *Starvation.* If a doodlebug has not eaten an ant within three time steps, at the end of the third time step, it dies of starvation and is removed from the grid.
- *Breed.* If a doodlebug survives for at least eight time steps, at the end of the eighth time step it spawns off a new doodlebug in the same manner as an ant. If no adjacent cell is empty, no breeding occurs. Once an offspring is produced, a doodlebug cannot produce another offspring until it has survived eight additional steps. Starvation takes precedence over breeding; that is, a starving doodlebug cannot breed.

During each time step, the doodlebugs act before the ants. That is, a doodlebug may eat an ant that was about to move and possibly to breed; as a result, that ant is dead and can no longer do either.

If an organism (i.e., an ant or a doodlebug) is eligible to breed but prevented from doing so by virtue of no empty adjacent cells, it remains eligible to breed on the next step.

This Assignment

Write a program to implement this simulation and draw the world using the ordinary characters **`'o'`** and **`'x'`** representing ants and doodlebugs, respectively. Create an abstract class called *Organism* that encapsulates basic data common to ants and doodlebugs. This class should have a virtual function called **`move()`** that is defined in the derived classes *Ant* and *Doodlebug*. You will also need a representation of the grid itself, and each cell of the grid should contain the *null* pointer (if empty) or a pointer to an **`Organism`**.

Program Arguments

The command line to run your program should resemble the following, where each of the arguments is an integer, and where any of the arguments (plus the following ones) may be defaulted:–

```
./PA6 gridSize #doodlebugs #ants #time_steps seed pause
```

1. **`gridSize`** — an integer representing the number of cells along one dimension of the grid (defaulting to 20)
2. **`#doodlebugs`** — an integer indicating the number of doodlebugs (default 5)
3. **`#ants`** — an integer indicating the number of ants (default 100)
4. **`#time_steps`** — the number of time steps to play (default 1000)

---

[2] This would be highly unusual, because the cell from which the ant moved would now be vacant.

[3] Obviously, if the ant cannot move, it also cannot breed, because there is no empty adjacent cell into which could be occupied by the newly born ant.

5. **seed** — an integer indicating the seed for the random number generator, with zero meaning to use the current time as the seed (default 1)
6. **pause** — an indication as to whether to pause. Blank or zero means do not pause. A non-negative value *n* means pause and print the grid after each *nth* step. Wait for one character input before continuing.

You may represent your grid in any way that you choose. For example, it may be two-dimension array similar to the ones we created for the Game of Life, or it may be an array of pointers to vectors, or it may be some other two dimensional data structure that is easy to access by indexes in the *x* and *y* directions. Each element of the grid should be an **Organism *** — i.e., a pointer to an object of type **Organism**.

Before the start, the specified number of Ants and Doodlebugs should be placed on the board at random locations.

Termination

The simulation should terminate after the number of steps specified on the command line or when all of the ants or doodlebugs are gone. After termination, print to **cout** a summary of the simulation, including

- the original command line as represented by **argv**,
- the number of steps simulated,
- the total number of ants during the course of the simulation and the number remaining,[4]
- the total number of doodlebugs in the course of the simulation and the number remaining, and
- a picture of the final grid.

Deliverables

You should create this project as a "**makefile** project with existing code" in *Eclipse CDT* as described in Lab #2.[5] When you are ready to submit, *clean the project* and then *Export* it to a **zip** file, also as described in Lab #2. The zip file should be named **PA6_username** or **PA6_teamName**, where **username** is replaced by your WPI login identifier, and where **teamName** is the name of the team as assigned in *Canvas*. The zip file should contain the following:–

- All of the *C++* files of your project, including your **.cpp** and **.h** files of your base and derived classes, plus at least one **.cpp** file for your **main()** function and simulation control.
- A **makefile**. The target name should be **PA6**. The **makefile** must be such that the graders can use it to build your project *outside of Eclipse.*[6]
- The output of at least two different test cases.

You must also submit a document called **README.txt**, **README.pdf**, **README.doc**, or **README.docx** summarizing your program and its principal classes and methods, how to run it, and detailing any problems that you had. If you borrowed any part of the algorithm or any test case for this assignment, be sure to cite your sources. Your **README** file should not be part of the *zip* file.

---

4 By "total number" in these two bullets, we mean the number of ants or doodlebugs at the start of the simulation plus the number of successful births.

5 The easiest way to do this is to replace the *Lab 2* files with your own and then edit the *Lab 2* **makefile** to refer to the names of the **.c** and **.h** files of this project.

6 This happens automatically if you create the project and export it *exactly* as specified in *Lab 2*. However, it is worth checking to make sure.

Before submitting your assignment, execute **`make clean`** to get rid of extraneous files, including *Debug* directories. Submit to *Canvas* under this assignment, which is named *PA6 -- Polymorphism*. Programs submitted after 6:00 pm on due date (March 2, 2017) will be tagged as late. Since this is the end of the term, no there can be no forgiveness for late assignments.

Grading

This assignment is worth fifty (50) points. *Your program* must *compile without errors in order to receive* any *credit.*

- The abstract **`organism`** class – 5 points:–
  - Correct definition of class and virtual methods – 5 points
- The **`ant`** subclass – 9 points:–
  - Correct definition of subclass – 3 points
  - Correct implementation of **`move()`** function – 3 points
  - Correct implementation of breeding – 3 points
- The **`doodlebug`** subclass – 12 points:–
  - Correct definition of subclass – 3 points
  - Correct implementation of **`move()`** function – 3 points
  - Correct implementation of breeding – 3 points
  - Correct implementation of eating – 3 points
- Simulation framework – 15 points:–
  - Grid implementation – 5 points
  - Primary simulation loop invoking the **`move()`** methods of organisms – 5 points
  - Processing command line arguments and setting up initial configuration – 3
  - Satisfactory output of steps of the simulation – 2 points
- Satisfactory **`README`** file explaining your class and your testing and showing the output of all test cases – 5 points
- Satisfactory execution of graders' test cases – 4 points
- Penalty of ten points if your project is not clean before creating the zip file or for submitting your programs in something other than a zip file.
- If the graders cannot build your program by executing **`make`** in a Linux command shell, your grade will be zero. *There will not be an opportunity to fix it before the end of the term.*

> It is therefore in your interest to be doubly sure that your program compiles correctly.
>
> The best way to do this is to build and run it exactly as the graders will — i.e., download the image as submitted to *Canvas,* unzip it to an empty directory *outside* your normal directory hierarchy, type the **`make`** command, and then run the program with a suitable command line, preferably one that you reported in your **`README`** file.