Teaching Java First: Experiments with a Pigs-Early Pedagogy

Raymond Lister

Faculty of Information Technology
University of Technology, Sydney
PO Box 123, Broadway, NSW 2007, Australia

raymond@it.uts.edu.au

Abstract

This paper introduces PigWorld, a tool for teaching Java to novice programmers, via the objects-early style. Three design guidelines for object-early assignments are discussed: (1) emphasize message passing between instances of the same class; (2) use only simple loops that search for smallest or largest values in a sequence; (3) teach linked lists before collections and collections before arrays. The paper offers a first step to resolving the dilemma over whether procedural programming must be taught before objects, via the following principle: in the procedural style, algorithms are encoded explicitly within the methods of an object, but in the object oriented style, algorithms emerge implicitly from the interactions between objects.

Keywords: CS1, Java, Objects-Early, Pigs.

1 Introduction

The Joint Task Force on Computing Curricula (2001) notes that Java and C++ *"are significantly more complex than classical languages. Unless instructors take special care to introduce the material in a way that limits this complexity, such details can easily overwhelm introductory students*." In what is probably the largest study ever done of the conceptual structures of novice programmers (Petre *et al*., 2004), there is evidence that the current methods of teaching Java and C++ have overwhelmed the students. In that study, students were asked to place various programming constructs and concepts into categories, based on criteria chosen by the student. The construct "Object" was most commonly placed with three other terms, "Tree", "List", and "Array". In our view, this indicates that many students think of objects naively, as being primarily a place where data is stored. We believe this is the result of a preponderance of examples of classes containing little more than private data members and their "get" and "set" methods; hence this student view of objects-as-data-structure.

Some computer scientists believe the problem goes beyond the choice of programming language, arguing more fundamentally that the teaching of the procedural programming paradigm must come first, as object-oriented programming "in no way replaces the older paradigm ... rather it is in addition to it" (Burton and Bruhn, 2003). Rebuttals of such arguments have been written (Culwin, 1999, Mitchell, 2001), but it must be conceded that the principled teaching of object-oriented programming has proved more difficult than many of us had originally hoped.

Long before Java, ways were sought to teach programming more gently. The seminal "Karel the Robot" (Pattis, 1981) was used to teach Pascal, in the context of programming a virtual robot to perform tasks in a grid-world. The Karel approach has been updated for Java in at least three independent cases (Becker, 2001, Bergin, 2003, Buck and Stucki, 2001). However, the emphasis in some cases on algorithms encoded within a single class, frequently recursive algorithms, brings a strong procedural flavour to the teaching.

This paper describes PigWorld, our vehicle for exposing students to object-oriented programming. PigWorld is in the "Karel" tradition, with the world's principal characters being pig's who move, eat, and make love within a grid-based micro world. The organic metaphor of PigWorld is not the first use of a non-robotic, organic metaphor (Barnes & Kolling, 2003, Brady, 2002, Pattis, 1997), but our use of the organic metaphor in PigWorld has led to a micro-world which places emphasis on interactions between objects, particularly instances of the same class. As the development of PigWorld has evolved over several semesters, we have developed an approach that doesn't treat "objects" and "procedural" as a dichotomy. While we place greater emphasis on objects, PigWorld allows us to incorporate an algorithmic component to the subject. However, instead of encoding those algorithms within the methods of an object, our algorithms emerge from the interactions between objects.

2 An Overview of Pigworld

Figure 1 shows a picture of a typical PigWorld scenario. In this scenario, there is one boy pig, on the left hand side, and one girl pig. (On a computer screen the color band on the bottom of boy pigs is blue, and for girl pigs it is pink.) There

are two pig food trees, and adjacent to one of the trees is a pill-like object, indicating that one of the trees has dropped some pig food. In PigWorld, hungry pigs move, one square at a time, toward the nearest pig food, which they then eat to absorb its energy. Moving around PigWorld takes energy. In Figure 1, the number and bar on the right of the girl pig icon indicates that she has 44 units of energy, while the boy pig has zero energy.

As pigs move around the world, they trail behind them a rope. The rope serves as a reminder to a pig of squares it has already visited, and by this device they can navigate the maze. In Figure 1, both pigs are trailing rope, across several squares.

At the bottom of Figure 1, there is a tool bar of various radio buttons. These buttons allow the user to add other elements to PigWorld, including walls. (Pigs cannot pass through walls.) Toward the right of Figure 1 are two controllers, one for each pig. The "Id" number in the top left of each controller matches the number displayed in the top left of a pig.

When a pig's energy level has risen sufficiently, the pig will lose interest in food. Instead, the pig becomes "in the mood for love". When a pig is in such a mood, little hearts appear on the pig's icon, and the pig's tight lips give way to a smile.

While in the mood for love, a girl pig emits a (virtual) noise. The "oink" propagates throughout PigWorld, with the volume of the oink decreasing with distance from the girl pig. Any boy pig also in the mood for love will move from his current square to the adjacent square where the "oink" is loudest. Thus, eventually, the amorous girl pig will be surrounded by a choice of suitors.

If a girl pig and a boy pig, both in the mood for love, should occupy adjacent squares, they may mate to produce a new pig. The progeny will appear on a vacant square adjacent to the parents. After mating, the two parents enjoy a short post-coital nap. When sleeping, a pig displays "Zzz" on its icon.

Pigs have poor eyesight, so in PigWorld they use radar to sense their surroundings. Radars return an instance of the class "Echo", containing the compass angle and distance to the entity just detected. Radars see through walls, so although a pig might know there is something of interest nearby, the pig may need to navigate around walls to reach it.

Life in PigWorld is not all food and sex. Wolves and other predators haunt PigWorld, preying on pigs.

(A small point about implementation: the current version of PigWorld uses AWT, but a Swing version is anticipated.)

3 The Class Hierarchy

Figure 2 shows the classes comprising PigWorld, as displayed in BlueJ (Barnes & Kolling, 2003), with the "uses" arrows turned off. (We find BlueJ to be an excellent teaching environment, but PigWorld can run in any Java programming environment, as it is pure Java.)

The first strong impression from Figure 2 is that PigWorld contains over 40 classes. For 4-6 weeks prior to PigWorld, students work on lab exercises which have no more than 3 classes. After their initial shock of seeing 40 classes, and some soothing words, students settle down and work comfortably in PigWorld. We scaffold tasks so that students aren't left to wander through 40 classes. In PigWorld assignments, we specify the particular method of a class to be modified, and frequently specify the methods of other classes that may have to be invoked. We support the guideline (Kolling & Rosenberg, 2001) that students should start by making small changes to large projects, emphasizing reading of classes over writing of code.

A second impression from Figure 2 is the class hierarchy, as indicated by the inheritance arrows. The students mostly work on classes that extend from "Thing", the classes "Animal", "Vegetable", and "Mineral", and in turn their extensions.

Each class that inherits from Thing has a matching "view" class. Most animals share a single AnimalController class. Early in semester, the primary virtue of the Model/View/Controller pattern in PigWorld is that it allows us to concentrate on the model classes without being distracted by graphics and event handling. Later in semester, students perform assignment tasks on the view/controller classes.

4 Assessment Structure

Like all subjects at our university, there are four passing grades, from the lowest "Pass", then "Credit", "Distinction", and finally "High Distinction". The assessment structure for this subject is unusual. It was described at last year's conference (Lister and Leaney, 2003). Students who are satisfied to merely pass this subject need not attempt the PigWorld assignment. The assignment is broken up into a series of tasks. In the most recent semester, in a class of 103 students, 49 students (48%) completed at least one of the seven tasks of the PigWorld assignment. As the first of those seven tasks is quite simple, we conclude that half the class simply elected to settle for a "Pass", not that half the class found all aspects of the assignment too hard.

5 Traditional OOP Moments

Each semester we set our students an assignment to augment PigWorld in various ways. The assignment solution in one semester becomes the (refactored) case study for the next

semester's class. In this section, we describe some experiences with orthodox object-oriented exercises from past semesters.

5.1 Message Passing Between Class Instances

In PigWorld's first semester, in what we thought was a simple familiarization exercise, we asked students to prevent pig procreation if the pair were brother-and-sister. Each pig carries references to its parents, so preventing such procreation requires each pig in a potential mating pair to compare their parent references to the equivalent references of the other pig. Students could encode this behaviour in a single "if" condition. We were therefore surprised to find that some students had difficulty, especially if they had prior experience in a procedural language. These students had developed a buggy concept of objects, which was closer to procedural. They saw a class and an instance of a class as being equivalent. Therefore, communication between instances of the same class appeared to them to be a recursive-like process in which an object communicated with itself. While in earlier Java exercises, they had written code for message passing between objects, these earlier exercises had involved objects of different classes, and thus the students' buggy notion had not been exposed. The modification to prevent brother-sister procreation was the first time they had written code for sending messages between instances of the same class. In subsequent semesters, we have made a point of including assignment tasks that require method passing between instances of the same class. Indeed, we have come to believe that communication between instances of the same class should be taught as early as possible, and reinforced as often as possible: it is fundamental to the teaching of object oriented programming.

5.2 GUI Exercises

PigWorld abounds with opportunities to add simple GUI elements. The use of the Model/Viewer/Controller means that students are exposed early to this concept.

In the most recent semester, pigs began to manipulate rope (discussed at length later), and a button for destroying a pig's rope was set as an assignment exercise. However, as rope is unique to pigs, the "rope" button could not be added to the AnimalController, so students were instructed to add the "rope" button to a new class, "PigController", which extended AnimalController. Of the 49 students who attempted the assignment, 37 (76%) successfully completed this task (i.e. 36% of the entire class).

5.3 Radars, Directions, and Echoes

In the first semester, PigWorld was without walls, and pigs roamed free. In that semester, students wrestled with the code within Pigs that uses a radar to locate food and other pigs.

Before writing that code, students first needed to understand two classes, the first being the "Direction" class, which essentially encapsulates an angle. In addition to the appropriate "get" and "set" methods for that angle, the Direction class also contained static directions, "EAST" (0 degrees), "NORTH_EAST" (45 degrees), "NORTH" (90 degrees), etc. The other class that students needed to understand was "Echo", which encapsulated both a reference to an instance of Direction and a distance.

A pig needed to proceed as follows. First, the pig needed to instantiate a radar, specifying as a parameter to the constructor what sort of entity the pig wished to detect. Then the pig needed to make a series of invocations of the "ping" method within that radar, to perform a 360 degree sweep of the world. The first invocation of "ping" would see the radar commence a sweep in the direction EAST, proceeding anticlockwise until an instance of the target class was detected. The "ping" method then returned a reference to an instance of Echo. The pig would then make further invocations of "ping", with each invocation continuing from where the previous invocation finished. To indicate that a full 360 degree sweep has been completed, the radar returns null. Throughout the sweep, the pig needed to retain the instance of Echo returned by the nearest entity detected thus far. On completing the sweep, the pig then stepped toward that entity.

To make a pig do all the above, students only needed to write about 10 lines of code. (Furthermore, the basic algorithm to do so effectively requires the student to find the smallest element in a sequence, an algorithm they are explicitly taught. This issue is discussed again below, under "Procedural vs. Object Algorithms".) Students reported that it took extensive study of the three given classes, Radar, Echo, and Direction, before they could successfully complete their 10 lines of code. That was exactly what we hoped, as we wanted this task to be an exercise in reading classes more than writing code.

5.4 Adding Classes/Creatures

Assignments require students to perform a series of graduated changes to PigWorld, culminating in the addition of complete classes. The organic metaphor makes it easy (even fun) to invent new creatures/classes for PigWorld.

In the first semester, students added a class "Wolf", which used its radar to hunt down and eat pigs. In the most recent semester, students added another pig predator, "RopeCrawler", which relentlessly tracks down a pig by following the trail made by the pig's rope. Of the 49 students who attempted the assignment, 19 (39%) successfully completed this task (i.e. 18% of the entire class).

6 Procedural vs. Object Algorithms

Even when taught objects-early, students must learn some basic control logic, otherwise they can only produce objects with data members, the associated get/set methods, and methods that return the result from a straightforward mathematical formula.

To produce interesting assignments within PigWorld, we have found it is sufficient to teach one basic algorithm: finding the smallest or largest element in a sequence of elements, where the sequence is terminated by a specified value. Initially, this algorithm is taught as a lab exercise, well prior to the students' introduction to PigWorld. In the lab exercise, the sequence of elements are numbers read from input, with the sequence terminated by zero.

We advocate that students being taught objects-early in their first semester should not be taught anything of a procedural nature more sophisticated than how to encode the above algorithm within a method. This simple algorithm requires a loop containing an "if" statement. One consequence is that the standard quadratic sorting algorithms, which require nested loops, are not then taught in the first semester.

We are not banning algorithms beyond the one described above. In fact, we advocate that students should be taught more sophisticated algorithms, as we shall illustrate in following sections of the paper. Crucial to understanding our argument is the distinction between coding an algorithm in the procedural style – as a behaviour implemented within a single object via explicit control logic – as opposed to implementing algorithms that emerge from interactions between objects. The following sections illustrate the latter.

7 Maze Traversal via Flower Power

When we added walls to PigWorld, pigs could no longer always move directly toward the object of their desire. Inspired by the prominent role of flowers in Jeroo (Sanders and Dorn, 2003), we added a new class, "Flower", to PigWorld. Flowers only have one significant attribute, their age. Students were asked to make the following changes to pig behavior. As a pig moved, it would leave a trail of flowers. If a pig moved on to a square that already had a flower, then the pig would replace that flower with a new flower. Thus, the age of a flower on a square is an indication of how long it has been since a pig visited that square. A pig still used its wall penetrating radar to choose the entity toward which it wanted to move, and if the next square in the direction of that entity was not blocked by a wall, or occupied by a flower, then the pig moved on to that square. However, if that square was blocked by a wall or had a flower on it, then the pig had to choose a different adjacent square to move to, based on the following three criteria:

1. A square not containing a flower is preferred.
2. If there is a choice between two squares, neither of which contains a flower, then the square in the direction nearest to the pig's intended target is preferred.
3. If all available squares contain flowers, then the square with the oldest flower is preferred

Criteria 2 and 3 are simple variations on the basic algorithm for finding the smallest element in a sequence.

The above algorithm does not implement a shortest path algorithm. Instead, the algorithm implements an exhaustive search algorithm. (We admit to one negative experience with PigWorld: many students persisted in thinking of the algorithm as a shortest path algorithm, which caused much confusion when they tested their code.)

In a more traditional procedural approach to searching a maze, a pig might store information internally about its progress through the maze, perhaps in a stack. Effectively, such a pig would be building an internal model of the world. In the objects-early approach to teaching algorithms, as illustrated by the "flower power" algorithm, a pig instead performs actions that embed in the world information about its progress through the world. As a general design guideline for objects-early assignments, we advocate minimizing the state information internal to an object, instead placing that state information into the world, external to the object.

(A small point: In retrospect, "Flower" was an unfortunate choice of name for the class used in this algorithm. Flowers are simple objects that merely extend Mineral. In a future refactoring, the name is likely to change, but we are also tempted to leave it as an illustration that no class hierarchy is perfect.)

7.1 Formal Approaches to Algorithms

In the debate on teaching objects-early versus teaching procedural first, the mistake is sometimes made that objects-early is incompatible with any treatment of algorithms. In the following two subsections, we pose two "flower power" problems that lend themselves to a classic formal treatment of algorithms.

7.1.1 Exercise 1

The exercise for the students is as follows: either prove that a single pig in any connected maze will eventually visit all squares, or provide a counter example. This exercise lends itself to a proof by induction.

7.1.2 Exercise 2

The exercise for the students is as follows: In a world without walls, and with flowers of different ages on all squares, find the worst case time before a pig will return to

its current square; giving the initial pattern of flower ages that causes this worst case behavior.

8 Dijkstra Meets the "Oink" Algorithm

As summarized in the overview of PigWorld, any girl pig who is in the mood for love emits an "oink" that attracts any boy pigs who are in the same mood. A more formal explanation follows.

In PigWorld, sound is propagated by instances of the class "Air". Each instance of Air is associated with a square, and each Air instance has references to the Air instances of neighboring squares. Sound does not propagate through walls, so if two squares are separated by a wall, then their associated instances of Air are not connected. This data structure is constructed at startup, and so is immediately available to students. In the most recent semester, students were required to add code to "GirlPig", "BoyPig", and "Air", to implement the behaviors described in the next paragraph.

A girl pig emits an "oink" by sending a message to the instance of Air associated with her current square. That message contains a single integer parameter, representing the volume of the "oink". The Air instance passes on that message to any neighboring Air instances, but decrements the volume. Those instances in turn pass on a decremented volume. When a boy pig is in the mood for love, he need only interrogate the instances of Air associated with neighboring squares, and move to the square with the highest volume. Once again, that choice of square by the boy pig is a simple variation on the basic algorithm for finding the largest element in a sequence.

The behaviours of the objects described in the above paragraph collectively implement a classic algorithm, commonly attributed to Dijkstra, for finding the shortest path through a graph. When this algorithm is implemented in the traditional procedural style, the control structure is beyond most first year students. Of the 49 students who attempted the assignment, 43 (88%) successfully completed this task (i.e. 42% of the entire class). As this task was part of an optional assignment, it was not supported by any discussion in lectures. Students attempted the task based upon the description in the assignment specification and answers to questions posted to an online discussion board. In a conventional procedural style programming subject, Dijkstra's algorithm would receive considerable attention in lectures, and students would still probably struggle to understand it.

The techniques used in the "oink" algorithm are fundamental to object-oriented programming. The part that students struggle with most was the propagation of the sound by the instances of Air. As with the incest prevention code discussed earlier, we suspect that the struggle was primarily caused by their weak grasp of message-passing between instances of the same class. We believe that the "oink" algorithm is an entirely appropriate exercise for a first semester objects-early programming class, as it entails the most fundamental object-oriented concepts we want to teach.

9 Ropes and Linked Lists

There are two problems with the earlier "flower power" algorithm for traversing a maze. The first is that the flowers laid down by one pig interfere with behavior of other pigs. The second problem is that, even in a single-pig world, once a pig has satisfied a goal (e.g. found some food), the flowers laid down by the pig up to that point may interfere with the pig achieving its next goal (e.g. finding another piece of food). These problems were addressed last semester by replacing the role of flowers with rope. Each pig trails its own rope, so pigs do not interfere with each others' maze traversal, making PigWorld truly multi-pig. Furthermore, once a pig satisfies a goal, it "rolls up" its rope and begins a fresh maze traversal.

The class "Rope" extends Mineral. An instance of Rope is associated with a square. Each instance has the references of up to two other pieces of rope, the instance laid out by the pig before this instance, and the instance laid out after. Thus, the entire trail of rope is a doubly linked list. Last semester, the code that allowed pigs to lay out rope was supplied as part of the case study. As part of their assignment, students were set the task of implementing the "rolling up" of rope, implemented as the destruction of the linked list, by having the instances in the list pass along a "destroy" message. The solution takes less than 10 lines of code. Of the 49 students who attempted the assignment, 40 (82%) successfully completed this task (i.e. 39% of the entire class).

In a procedural language, the destruction of a linked list can be done as a short piece of iterative code, which could in principle be taught in the first semester. In practise, however, a thorough study of linked lists is usually left until a later semester in procedural programming, where recursive algorithms are used extensively. Besides, the procedural-style first semester syllabus is already full with other algorithms to study, particularly the venerable bubblesort and other iterative processes on arrays.

In the objects-early approach to teaching programming, we believe the linked list is the natural choice for the first data structure. Linked lists reinforce core elements of object-oriented programming, as they require students to pass messages between multiple instances of the same class. In Java, arrays are a peculiar beast. Java arrays are objects, but they come with an additional syntactic overhead which obscures for students the fact that arrays are objects. We recommend that Java arrays not be taught at all in the first semester, as they distract students from the core elements of object-oriented programming.

10 The Collections Framework

Having introduced the link list, the next logical step, before arrays, is to introduce collections. As described earlier, pigs instantiate a radar and then repeatedly invoked the radar's "ping" method to perform a full 360 degree sweep. In the most recent semester, Radar also came with a new method, "sweep", which performes a full 360 degree sweep in a single invocation, returning an instance of "List" that containes all the echoes. As an assignment exercise, students were required to modify code, replacing the multiple use of "ping" with a single invocation of the collections-oriented "sweep". Students then had to use an Iterator to find the appropriate (i.e. "smallest") element in that List returned by "sweep". Of the 49 students who attempted the assignment, 35 (71%) successfully completed this task (i.e. 34% of the entire class).

11 Conclusion

This paper presented PigWorld at three conceptual levels. The first and most mundane conceptual level is PigWorld itself, a piece software, a vehicle for teaching object-oriented programming in Java. We are happy to share this piece of software with others.

In the second conceptual level of presentation, we advocated some objects-early assignment design principles: (1) emphasize message passing between instances of the same class; (2) use only simple loops that search for smallest or largest values in a sequence; (3) teach linked lists before collections and collections before arrays. In describing the "flower power" algorithm, we illustrated a PigWorld design guideline: minimize the state information internal to an object; instead place that state information into PigWorld itself, external to the object

In the third conceptual level of presentation, we illustrated how a solid treatment of algorithms is not incompatible with the objects-early approach to teaching programming. We introduced the following principle: in the procedural style, algorithms are encoded explicitly within the methods of an object, but in the object oriented style, algorithms emerge implicitly from the interactions between objects. For some years, advanced texts on object-oriented programming have advocated keeping individual classes simple, using compositions of objects to perform complex tasks. We have designed PigWorld to bring that previously advanced view down to the first semester of object-oriented programming.

12 References

Barnes, D., Kolling, M (2003): *Objects First With Java: A Practical Introduction Using BlueJ*. Prentice Hall.

Becker. B (2001): Teaching CS1 with Karel the Robot. *Proc. ACM SIGCSE 32nd Technical Symposium on Computer Science Education*, Charlotte NC. 50-54 ACM Press. Related materials available at http://www.cs.uwaterloo.ca/~bwbecker/robots/

Bergin, J., Stehlik, M., Roberts, J., Pattis, R: Karel J. Robot: A Gentle Introduction to the Art of Object Oriented Programming. http://csis.pace.edu/~bergin/KarelJava2ed/Karel++JavaEdition.html. Accessed September 2003.

Brady, A (2002): *The Marine Biology Simulation Case Study*. The College Board: New York.

Buck, D., Stucki, B (2001): JKarelRobot: A Case Study in Supporting Levels of Cognitive Development in the Computer Science Curriculum *Proc. ACM SIGCSE 32nd Technical Symposium on Computer Science Education*, Charlotte NC. 16-20 ACM Press. Materials available at http://math.otterbein.edu/JKarelRobot/

Burton, P.J., Bruhn, R.E (2003): Teaching Programming in the OOP Era. *SIGCSE Bulletin*, **35**(2): 111-114.

Culwin, F. (1999): Object Imperatives! Proc. ACM SIGCSE *30th Technical Symposium on Computer Science Education*, New Orleans LA. 31-36 ACM Press.

Joint Task Force on Computing Curricula. Computing Curricula 2001 Computer Science. *Journal of Educational Resources in Computing*, **1**(3): entire issue.

Kolling, M., Rosenberg, J. (2001): Guidelines for Teaching Object Orientation with Java. *SIGCSE Bulletin*, **33**(3), 33-36.

Lister, R and Leaney, J (2003): First Year Programming: Let All the Flowers Bloom, *Fifth Australasian Computing Education Conference (ACE2003)*, Adelaide, Australia.

Lister, R and Leaney, J (2003): Introductory Programming, Criterion Referencing, and Bloom, *Proc. ACM SIGCSE 34th Technical Symposium on Computer Science Education*, Reno NV, ACM Press.

Mitchell, W. (2001): A Paradigm Shift to OOP has Occurred... Implementation to Follow. *Journal of Computing in Small Colleges*, **16**(2), 95-106. ACM Press.

Pattis, R.E. (1981): *Karel the Robot: A Gentle Introduction to the Art of Programming*. Wiley.

Pattis, R.E. (1997): Teaching OOP in C++ Using an Artificial Life Framework. *SIGCSE Bulletin*, **29**(1): 39-43.

Petre, M., et al. (2004) A large-scale elicitation of students' knowledge of programming constructs. Submitted for publication.

Sanders, D., Dorn, B. (2003): Jeroo: A Tool For Introducing Object-Oriented Programming *Proc. ACM SIGCSE 34th Technical Symposium on Computer Science Education*, Reno NV. 201-206 ACM Press.

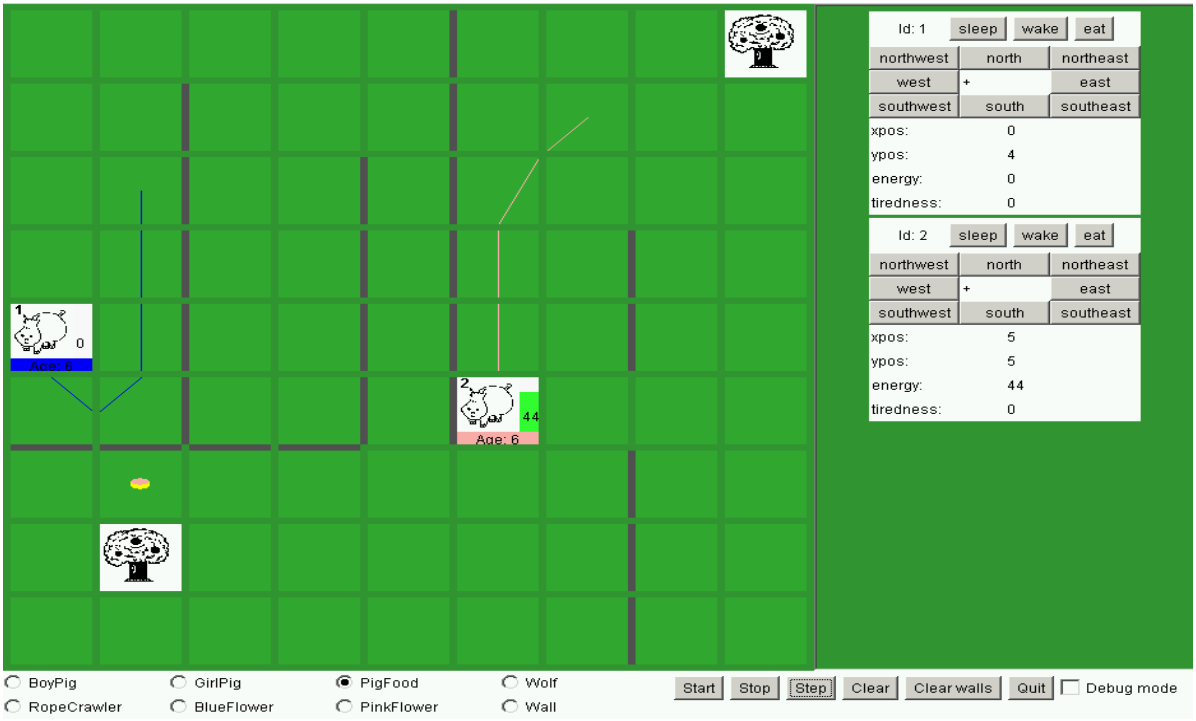

Figure 1. The PigWorld user interface.

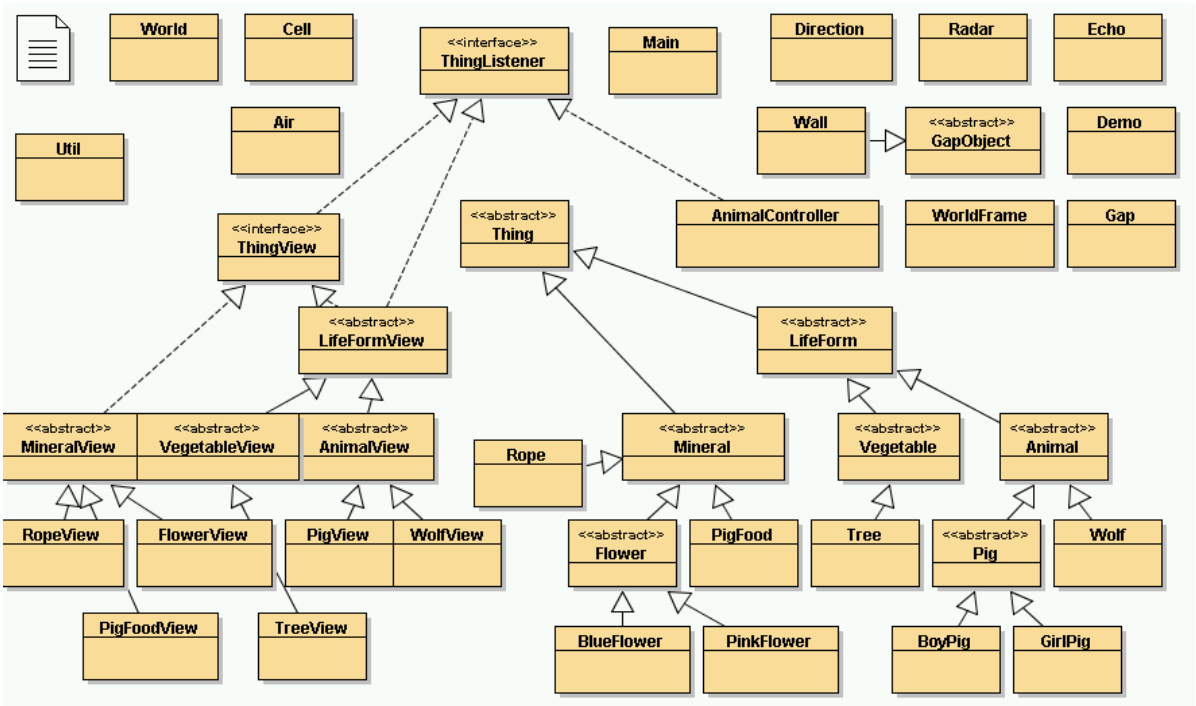

Figure 2. The PigWorld class hierarchy as displayed by BlueJ, without "uses "arrows.