Keele University

Learning Programming Through Competiton

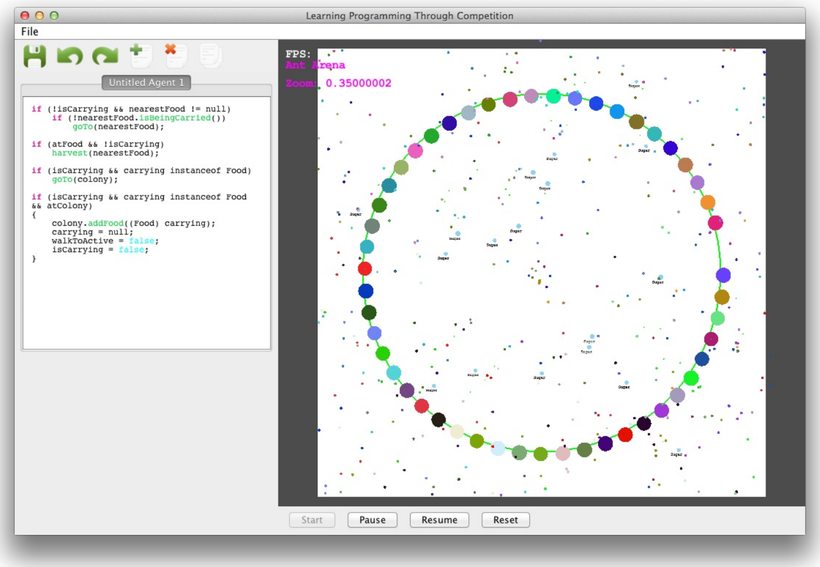
By Paul Rose - Supervised by Dr Theocharis Kyriacou
Computer Science 2012

About this Project

This project aims to aid the teaching and learning process of computer programming, by creating an application to allow basic programming concepts, sequence, selection and iteration, which have suggested to be the three main concepts that students struggle with (Dehnadi and Bornat, 2006), to be tried and experimented with, in a competative environment.

It is hoped that this environment can better help students understand the material they are taught, whilst keeping them motivated and engaged as according to Jenkins (2001), comptuer science students today, compared to 15 years ago do not have the same level of pratical skill or the willininghess to develop the skills needed. However this seems not only linked to motivation as "Even universities that can select, from among the best students, however, report difficulties in teaching Java" (Boyle et al., 2003, p.1).

This application would hopefully be a catalyst for students, helping more to pass intoductory programming courses and avoid the current "national crisis" in the learning and teaching of computer programming (Boyle et al., 2003, p.1).



This screenshot shows 50 colonies and thier ants searching the arena for food. The code on the left shows the different level of control a player can have when altering the state of their ant.

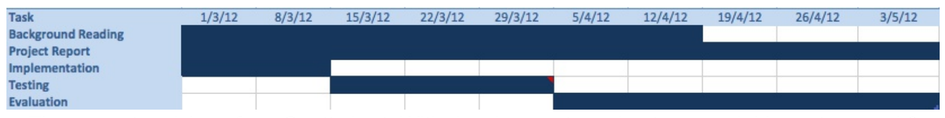
How to play

The aim of the ant arena is to keep your colony alive by foraging for food, while fending off the other ants around you either by clever foraging tatics or by aggressive means.

Students launch a copy of the program and code their logic for their ant in the code box on the left side of the program. When a student wants to see how their code works, they press start on the simulation and their code is compiled and included in the program, allowing for instant visual feedback on their code.

When the pratical is nearing a close, the supervisor will ask students to export their code and submit a copy, where they will start an arena instance with all of the participating students code.

Rules for the arena are flexible, but will defaultly include random food drops that diminish over time. As it costs colonies food to create ants and live and ants also use food to live and carryout actions, eventually colonies ants die out, leaving only the sucessful coded ones alive. The game ends when there is only one colony left or when time runs out.



The current revision of my Gantt chart. With development coming to an end, I hope to spend 3 weeks testing the ant arena before compiling my results into my final report

Progress so far

At this time the basic arena code is completed allowing large numbers of ant agents to interact with all objects in the world. Custom code can be written and included in the program at compile time. Different arena rules concerning food, arena size and limits on objects can be configured.

The code editor works in a rudimentary fashion and needs features activated such as edit tools, loading and saving and more colour highlight options. However, this is not an important feature and can be dropped in favour of an external editor such as BlueJay.

Things to work on

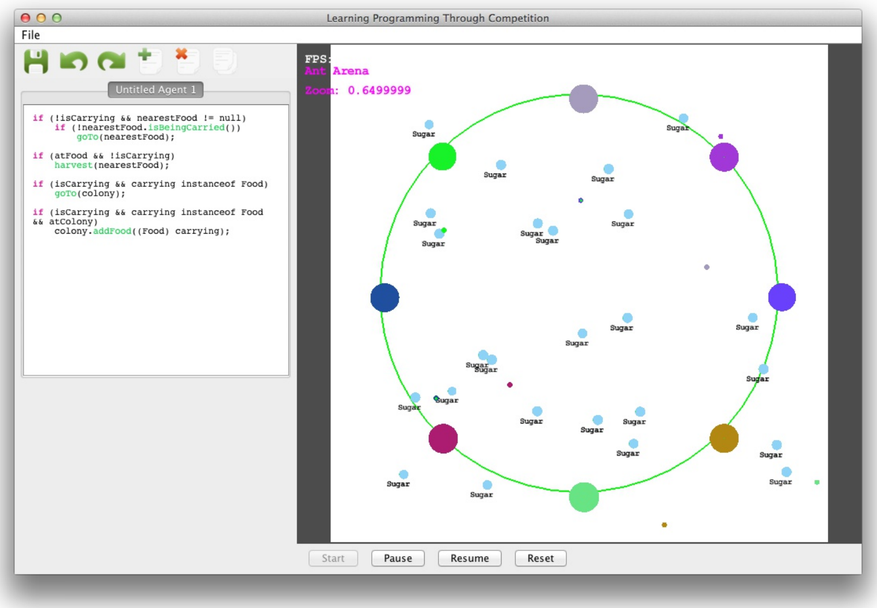
Better helper methods are needed to make it easier for beginners to understand what is happening. In most cases this means wrappers on Java constructs like instanceof.
Testing is needed to verify the program will work in the enviroment it was deisnged for.

References:

Boyle, T., Bradley, C., Chalk, P., Fisher, K., Jones, R., and Pickard, P. (2003). Improving pass rates in introductory programming, 4th Annual LTSN-ICS Conference, NUI Galway, 2003 [http://www.ics.heacademy.ac.uk/Events/conf2003/Pete%20Chalk.pdf] (Accessed on 30/01/12)

Dehnadi , S., and Bornat, R. (2006). The camel has two humps (working title). [Available at http://hssc.sla.mdx.ac.uk/research/PhDArea/saeed/paper1.pdf] (Accessed on 24/01/12)

Major, L., Kyriacou, K., and Brereton, P. (2011). Simulated Robotic Agents As Tools to Teach Introductory Programming, [Available at http://library.iated.org/view/MAJOR2011SIM] (Accessed on 29/02/12)



This screenshot shows the Ant Arena running on the right, with the code used to program the ants on the left. On the arena large cirlces represent colonies while small ciricles of the same colour represent their ants. Ants with green circles are carrying food.

Project Requirements

This project is required to be:

* Engaging - something that can be related to by first time users and therefore more motivating.
* A common arena for at most 50 students in a single arena/ playing field.
* Able to be finished in 3 hours - i.e the length of a practical slot
* Written in Java - as new students are taught java at the start of the course

The Ant Arena

This project is loosely based on ideas from a recent paper, Simulated Robotic Agents As Tools to Teach Introductory Programming (Major et al., 2011), where users were tasked with programming the navigation for simulated robots. The deviation from the original application is the introduction of a competative arena where multiple students are able to write code that will be run untill there is a single or set of winners.

The decision to use the concept of ants as the agents which will act out user code, stemmed from the versitility of programming possibillties for students and the simplicity of the concepts and rules that could be applied to arenas, whilst fulfilling the requirements agreed with Dr Kyriacou, the stakeholder.

```
if (!isCarrying && nearestFood != null)
    if (!nearestFood.isBeingCarried())
        goTo(nearestFood);

if (atFood && !isCarrying)
    harvest(nearestFood);

if (isCarrying && carrying instanceof Food)
    goTo(colony);

if (isCarrying && carrying instanceof Food
&& atColony)
    colony.addFood((Food) carrying);
```

This screenshot shows some helper methods and basic selection

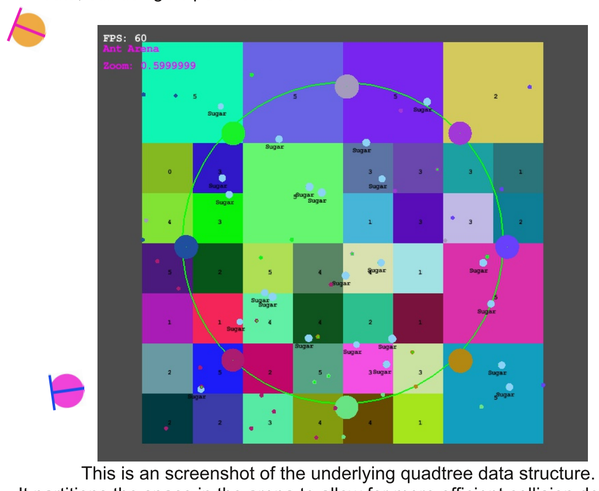
```
if (!carryingObject)
{
    for (Entity e : collisions)
    {
        if(e instanceof Food)
        {
            if(!e.isBeingCarried){
                pickup(e);
            }
        }
    }
}else if(carriedObject instanceof Food){
    goto(colony);
}else if(carriedObject instanceof Ant){

    gotoWater();

    if(atWater())
    {
        throwObject();
    }

}else{
    walkForwards();
}
```

This screenshot shows use of iteration and manual manipulation of collison data, still using helper methods

Sequence, Selection and Iteration

One of the advantages of the ant arena is its versitility when coding ant behaviours. A beginner with basic knowledge of sequence and selection can use helper methods and convenient references to find close objects of value and interest and then engage with them. A more advanced student who has learnt about variables and iteration can directly access all properties of the ant, such as state variables and collision lists as well as references to other objects to create a more complex ant agent.

This versitilly allows users of different abillity to take part as well as not narrowing the use of the application to a beginners only approach.



This is an screenshot of the underlying quadtree data structure.
It partitions the space in the arena to allow for more efficient collision detection, reducing a $O(n^2)$ complexity to $O(nlog(n))$.

This image shows an ants circle of vision, with blue lines representing a collision with another object. Novice programming students will have available to them the references to the nearest and biggest food source. Advanced programming students can iterate through the lists of collisions and make better decisions on what objects to interact with. For example to attack an ant carrying food, or to favour one food source over another

A simple UML class diagram showing the layout of the program. It has been kept as decoupled as possible from the main GUI to enable seperate program entry points. For example the Ant Arena can be lauched without the Text editing components for a class presentaion.