Tell and Draw, Card Programming

Developed by: Jun-PyO Park, Seung-Bum Kim, Dong-Hee Park and Seung-Joon Choi.

Arranged by: Tim Bell.

This is an engaging extension of the CSUnplugged "Marching Orders" activity. The extension was developed by the PINY team in Seoul, Korea. As well as introducing programming, it exposes students to the idea of open-source development, and also programming language design.

Card programming—*Programming Language design*

Summary

Computers are usually programmed using a "language," which is a limited vocabulary of instructions that the computer can obey. The computer will only follow instructions that are part of the "language", and they follow those instructions to the letter. In this activity students are challenged to think about how such a language would be designed so that it is powerful enough to do interesting things, and yet is easy to understand. Furthermore, most software systems are too big for one person to develop, and one popular model for getting contributions to a system from others is "Open Source" software, where a program is made publicly available, and others can try to improve it, and share their improvement with the community. The activity is intended to give the students insight into programming, open source software, and how programming languages are designed.

Curriculum Links

- ✓ Computing: Programming languages, software engineering, open source software
- ✓ English: Interpersonal Listening

Skills

- ✓ Giving and following instructions.
- ✓ Planning

Ages

- ✓ 7 years and up
- ✓ Note: this is an extended activity that could take two or three hours to complete

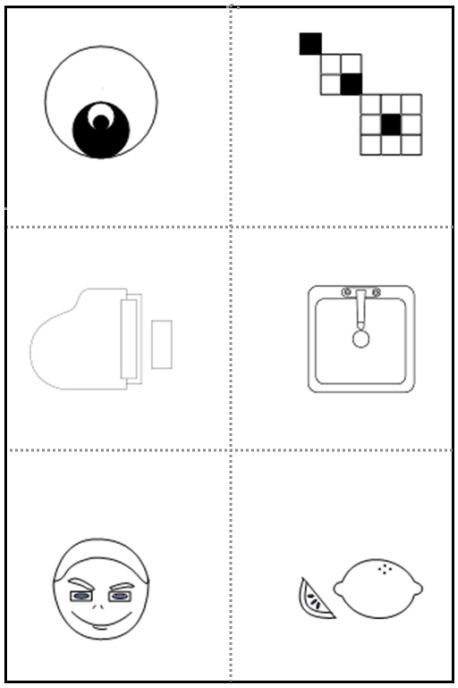
Materials

You will need:

- ✓ Cards with pictures such as the ones shown on the next page.
- ✓ Blank index cards (allow about 20 per pair of students)
- ✓ Grid lined paper

Each student will need:

- ✓ Pencil, paper and ruler



Card programming

Introduction

There are three phases to the activity: (1) Communicating an image by verbal description (2) Communicating an image with a written description, and (3) Designing a list of instructions and using them to communicate images.

First, discuss whether it would be good if people followed instructions exactly. For example, what would happen if you pointed to a closed door and said, “Go through that door”?

Computers work by following lists of instructions, and they do exactly what the instructions say—even if they don’t make sense!

Verbal description

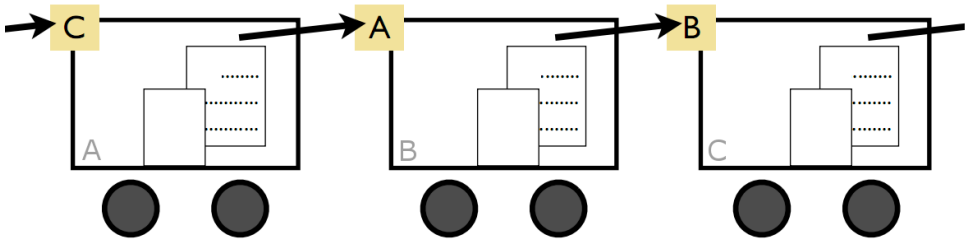
Have the students get into pairs. The first student in each pair is given an image that they must communicate verbally to the second, and the communication must be one-way (the second student cannot ask any questions). The second student attempts to draw the image based on the description. The pairs are given a fixed amount of time (say, 5 minutes) to complete the task.

After this, show the pairs of images (original and drawn) to the class, and have a vote or discussion on which is the best. For example, the pairs of images could be put up on the wall, and students could vote by putting a sticker on their favorites.

Written description

Now have the students do a similar exercise, but this time with a written description (textual only). Give the first person in each pair 5 minutes to write a description, and then the second student has a further 5 minutes to try to draw the image given only the textual description.

Before judging the quality of the outcome, take the original image, the textual description, and the image that was drawn to one person in another pair (they could be rotated around the room).



The person who has the three sheets of paper is then asked to edit the textual description (use a red pen to mark it up) to try to improve it, taking into account any errors that appeared in the image that was drawn from the original written description.

The new description is then given to their partner, who must attempt to draw the image.

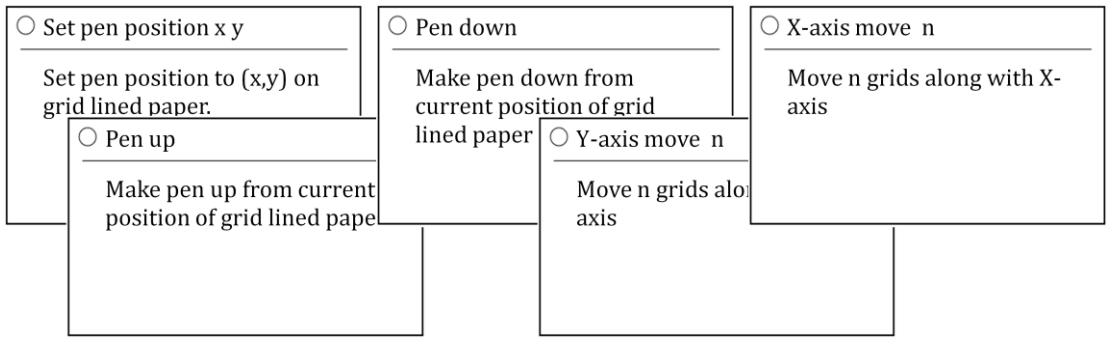
There will now be three versions of the image: the original, the first attempt, and the second attempt from the (hopefully) improved description. Put the three versions up on a notice board for voting and discussion. Did the edited description improve the outcome? Did it make it worse in some cases? Is it helpful to be able to do this i.e. have different people work on a description to improve it, and allow people to use the new description if they think it is better?

Note: you may find that often the second description is better, but sometimes the editing is a step backwards and the outcome is worse! This is typical of computer programming, where you hope that most changes result in an improvement, but sometimes they can make things worse – sometimes even much worse!

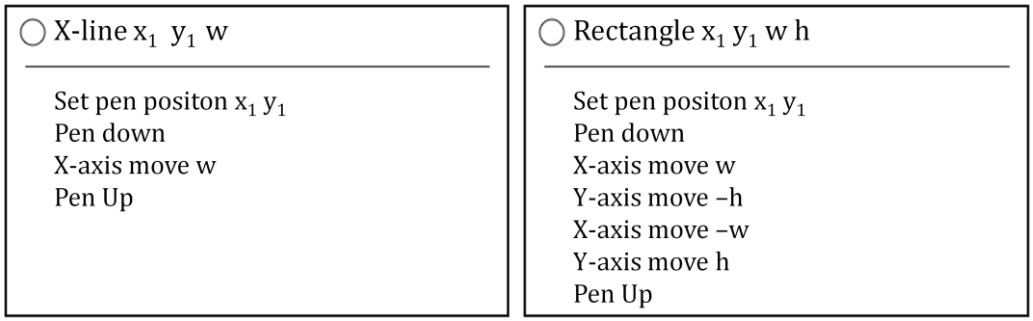
Card programming language

In this phase of the activity, students are asked to prepare generic instructions that they can later put together to describe an image. To make the descriptions easy to keep accurate, the drawings will be made on a numbered grid, so the instructions can refer to (*x,y*) coordinates. Typical instructions might be to move the pen to a position on the grid, and to draw a line between two given points. Each instruction should be written on one card. The challenge for the students is to try to anticipate what sort of instructions will be useful.

Here are some examples from students:



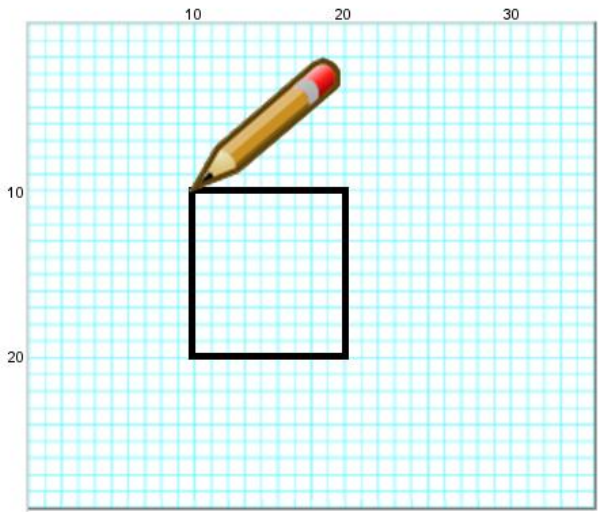
The students are then given an image (drawn on a grid) that they must communicate using the instructions. Some sample images are given below. They are only allowed to use the instructions they've created, or they can make new ones that can be expressed using only their existing instructions.

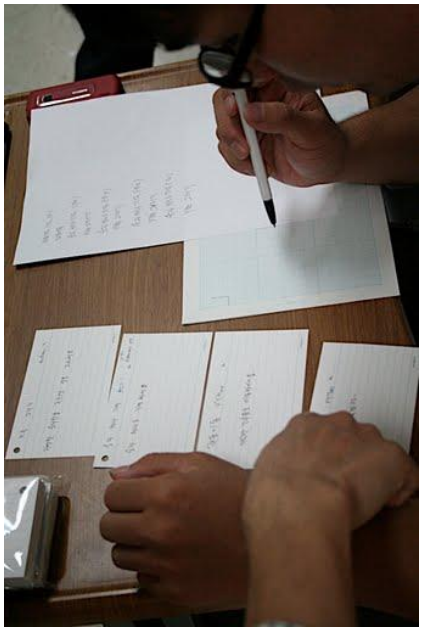


This time, the image is not drawn by a partner, but by a person who is designated to be a "tester" (perhaps a teacher, or a student who is familiar with programming). Students queue up to present their image to a tester, who follows their instructions on the cards precisely, and draws the image. The student then takes the image away, and attempts to improve the program, bringing it back each time for testing.

This simulates the write-compile-debug cycle for programming. Having a delay (queue for testing) forces students to think very carefully about their programs – if they were doing it on a computer they might be tempted to make small changes to see if it fixes a problem, rather than reasoning through corrections carefully.

Finally, students can be given a parameterized task, such as writing a program to draw a square which has a given size.





Follow up

There are a number of on-line systems that use "turtle graphics" like the card-programming system above. One example is "Design by Numbers" (dbn.media.mit.edu), which can be run from the website.

What's it all about?

Computers operate by following a list of instructions, called a program, that has been written to carry out a particular task. Programs are written in languages that have been specially designed, with a limited set of instructions, to tell computers what to do. Some languages are more suitable for some purposes than others, and there are many researchers who work hard to design new languages to make it easier for programmers to do their job.

Regardless of what language they use, programmers must become adept at specifying *exactly* what they want the computer to do. Unlike human beings, a computer will carry out instructions to the letter even if they are patently ridiculous.

It is important that programs are well written. A small error can cause a lot of problems. Imagine the consequences of an error in the program of a computer in a space shuttle launch, a nuclear power plant, or the signals on a train track! Errors are commonly called "bugs" in honour (so it is said) of a moth that was once removed ("debugged") from an electrical relay in an early 1940s electronic calculating machine.

The more complex the program, the more errors there are likely to be. This became a major issue when the USA was working on the Strategic Defence Initiative ("Star Wars") program, a computer controlled system that was

intended to form an impenetrable defense against nuclear attack. Some computer scientists claimed that it could never work because of the complexity and inherent unreliability of the software required. Software needs to be tested carefully to find as many bugs as possible, and it wouldn't be feasible to test this system since one would have to fire missiles at the United States to be sure that it worked!

Most programs these days are developed by teams of programmers, in some cases by thousands of people! To coordinate such large projects, formal rules and methodologies are needed, and looking after these sorts of issues is called *software engineering*.

One interesting approach to developing programs is *open source software*, where the author of a program makes their work available for other programmers to inspect, improve, and re-publish. This process can enable very large projects to be completed (for example, the widely-used Firefox web browser and OpenOffice productivity suite were developed using an open source approach).