Programming and constructionism: Logo revisited

Using Basic Logo to draw a floorplan for a room

Logo was originally advocated as the primary vehicle for 'constructionism'. The idea was that learning to draw geometric objects using Logo was a way to promote mathematical learning. It is instructive to revisit the process that is involved in drawing simple diagrams using basic Logo. We attempt to draw the Donald floorplan discussed in the labs, first invoking a basic Logo extension to EDEN by loading the `Run.e` from `logoparserRoe2002`. For technical reasons, it is essential to use **tkeden-1.46**.

```
%eden
## initially set the pen up
## execute following commands to get to bottom left corner at {50,50}

%logo

pen up
backward 200
left 90
forward 200
right 90

%eden
## set pen down with pen colour black

%logo

pen down

forward 400
right 90
forward 10

right 90
forward 100
```

Now need to move without drawing to start drawing again from the lock of the door. I then draw three lines to complete the N wall, and inseret E and S walls:

```
%eden
## set pen up

%logo

pen up

backward 100
left 90
forward 100
```

```
%eden
## set pen down with pen colour black

%logo

pen down

forward 300
right 90
forward 400
right 90
forward 400

%eden
## leads to error: didn't correctly compute the length from the door lock
to the NE corner
```

I have made a mistake ... I try to correct this using undo()

```
%eden
undo();

## BUT - only one undo is possible! (it affects line l6 as explained below)
```

To undo more radically, we can fish out the donald definitions of the lines that are being drawn by the Logo commands, and redraw them ...
... consulting the donald definitions, we find that they are:

```
l6 = [{460.000000, 50.000000}, {60.000000, 50.000000}]
l5 = [{460.000000, 450.000000}, {460.000000, 50.000000}]
l4 = [{160.000000, 450.000000}, {460.000000, 450.000000}]
```

... but they really should have been as above with 460 replaced by 450 in every definition.

(Note in passing that the effect of the previous undo was to redefine the attributes of `l6` so that it became *transparent*.)

To fix the problem, we make the redefinition:

```
%donald
l6 = [{450.000000, 50.000000}, {50.000000, 50.000000}]
l5 = [{450.000000, 450.000000}, {450.000000, 50.000000}]
l4 = [{160.000000, 450.000000}, {450.000000, 450.000000}]

%eden

A_l6="color=black";
```

But now the Logo turtle is in the wrong place!
... fix this by finding out how the Logo turtle is modelled - this is done by:

```
_turtle_pos is cart(turtle_x, turtle_y);
```

The eden variables turtle_x and turtle_y are integers: so relocate via:

```
turtle_x = 50;
```

Can then resume the drawing of the floorplan - next focusing on the table:

```
%logo

pen up

right 90
forward 200

%logo
right 90
forward 200

pen down
forward 150
left 90

forward 150
left 90

forward 150
left 90

forward 150
left 90

forward 150
left 90

forward 150
left 90


pen up

forward 75
left 90

forward 75
```

At this point, I *at first* think about drawing the cable, and recognise many of the difficulties

- calculating the direction - which is arctan (325-50)/(325-250)

- calculating the distance

and worrying about how to cope with the non-integral distance and its effects - for instance, on my ability to restore my current state after drawing the cable.

I proceed instead to try to draw the octagonal part of the lamp first ...

```
%logo

right 180

forward 25
right 90
forward 12

%eden
## not the ideal place - intend 12.5, but that's not an option?


%logo

pen down

right 45
forward 25

right 45
forward 25

right 45
forward 25

right 45
forward 25

right 45
forward 25

right 45
forward 25

right 45
forward 25

right 45
forward 25

pen up
backward 12
right 90

forward 25
forward 12
```

It's now clear that this isn't right: the original octagon in the table_lamp is not a regular octagon - the sides alternate in length between 50 and sqrt(1250), which is about 35.36 ...

... finally I come to draw the cable: The kind of calculations that we now need to do to draw the cable: we ideally have the current position of the turtle at {325,325} (at the centre of table), and want to be at (250, 50):

```
writeln(atan(75.0/275.0)*180/PI);
```

This gives an angle of about 15.26 degrees

```
writeln(sqrt(75.0*75.0+275.0*275.0));
```

This gives a distance of about 285.04
Can use these to estimate how to draw the cable using Logo commands:

```
%logo

pen down

right 15
forward 285

%eden
## current turtle position is now close to the midpoint of the S wall of
the room


hdturtle();

A_l22="color=red";
```

Appendix: Making the Logo display window sensitive

```
## can be useful to make the Logo window sensitive:

%scout
window show = {
    type: DONALD
    box: [{5, 40}, {505, 540}]
    pict: "draw"
    xmin: 0
    ymin: 0
    xmax: 500
    ymax: 500
    border: 2
    relief: "groove"
    sensitive: ON
};
```

Matters arising ...

There are lots of reasons to question whether programming in Logo is an effective way to connect construction with learning (especially learning about the domain rather than about Logo programming!). Some considerations are:

- state is being described through side-effect: not clear that side-effects generate "observables" (e.g. how to reference what has been constructed? how to restore focus to features no longer current in the construction?)
- undo is critically context-dependent: more generally, contextualisation of procedural actions is at all times critical, but context is hard to control
- means to review and assess overall state is limited (cf. the way in which the EDEN implementation supplies an environment for observation)
- making the translation between what I can conceive and what I can construct can be very hard: cf. what is undecidable / infeasible in classical computer science
- essential need to remember where we are in the recipe - consequences of "losing our place" very difficult to redeem
- no conceptual support at all for making 'meaningful' transformations to the cumulative display (e.g. let's re-draw the floorplan after moving the table!)

Above all, there is a certain mindset that we have to have - one that novice programmers being introduced to programming in a computer science department will be familiar with: you avoid errors at all cost, plan meticulously and don't make experimental changes, treat the machine with the utmost respect - try to find out everything about it and make sure you take this into full consideration before you instruct it / interact with it.

In the words of Mordecai Ben-Ari (*Constructivism in Computer Science Education* - as cited in EM paper #107): “intuitive models of computers are doomed to be non-viable” - computer science students must contend with the computer as "an accessible ontological reality".

Of course, the computer environments proposed for constructionist learning have developed far from primitive Logo. Modern variants of Logo for educational use, like Imagine Logo have much in common with object-oriented programming environments where many of the procedural elements exposed in basic Logo are no longer so explicit. But how programming paradigms influence the expression of constructionist principles, and what aspects of a computer environment liberate constructionist principles still remain significant questions.

References

- Chris Roe, Computers for Learning: An Empirical Modelling perspective, PhD thesis, University of Warwick, November 2003 (especially Chapter 4, p.106)
- W.M.Beynon and Chris Roe. Computer support for constructionism in context. In Proc. of ICALT'04, Joensuu, Finland, August 2004, 216-220. (EM paper #080)
- S.Papert. Mindstorms: Children, Computers and powerful ideas. New York: Basic Books, 1980.
- S.Papert, I.Harel. Situating Constructionism. In S.Papert, I.Harel (eds). Constructionism: Research reports and essays, Ablex Publishing, pages 1-11, 1991.
