A Turtle's genetic path to Object Oriented Programming

Reinhard Oldenburg, *oldenbur@math.uni-frankfurt.de*
Dept. of Mathematics and Informatics Education, Goethe University Frankfurt, Germany

Magnus Rabel, *rabel@math.uni-frankfurt.de*
Dept. of Mathematics and Informatics Education, Goethe University Frankfurt, Germany

Jan Schuster, *schuster@math.uni-frankfurt.de*
Dept. of Mathematics and Informatics Education, Goethe University Frankfurt, Germany

Abstract

The good old idea of Turtle graphics still has an enormous potential. This paper presents an approach which uses three-dimensional Turtle graphics as a pathway to object oriented programming.

Keywords

Turtle Geometry, 3D, Programming, Object Oriented Programming, Python

Introduction

The turtle introduced by Seymour Papert may be considered to be the most important prototype of an artificial and abstract (depending on its incarnation) object that students interact with. By now it has a rich history with many variations of the approach (e.g.Papert, 1993, Abelson & DiSessa, 1981, Hromkovič, 2010). While originally introduced with the Logo programming language, the idea has migrated to many other languages and environments influencing many microworlds designed for the learning of algorithmic programming concepts (e.g. Scratch, NetLogo, Kara). Turtle geometry has also been used successfully in mathematics (Heid & Blume, 2008).

This paper presents an approach that combines two logically independent yet didactically useful extensions to the original concept: 3D and object orientation.

There are several projects that extend the Turtle into the third dimension (Paysan, 1999, Wolfram Demonstrations Project, "Elica,", "NetLogo 3D,"). Our approach to computer science introduction in high school is based on the conviction, that a mainstream language should be used that spans the whole learning time in high school to give students the opportunity to acquire a solid understanding of the language's semantics so that they can use it to express algorithmic ideas. In our case we decided to use Python. For Python there exists the very nice extension package "Visual Python" (short: VPython, www.vpython.org) to do easy 3D graphics, but there has no turtle been implemented for it yet. Thus we did this ourselves and built the package VTools that includes the class Turtle3D accompanied by some other helper functions for 3D Programming. The advantage of having a turtle in a mainstream language over using specialized environments is that other concepts of computer science can be linked. E.g. the turtles in Netlogo are nice and useful in many ways but there is no way to do standard (object oriented) programming with them.

Object orientation is quite natural with turtles. The build-in 2D turtle that ships with each Python distribution (as well as many turtle extensions for Delphi, Java etc.) is defined as a class so that one can create more than one turtle and this is a good starting point to introduce the dot notation widely used in OO languages: If you have two turtles, `turtle1` and `turtle2`, you must specify to which turtle a command is addressed to and students grasp the meaning of `turtle1.forward(5)` at once. This should help as well against the sometimes observed misconception that students don't distinguish between class and objects.

Turtle graphics are used in some courses that aim at object oriented programming (OOP). In (Schaub, 2000) it is used as an introduction to programming but it is not continued when the concepts of OO are introduced. Thus it seems that there is a lack of literature connecting turtle graphics and more advanced OO programming concepts.

In this paper we first provide an overview of the 3D turtle, its possibilities and its use in introducing algorithmic concepts (which is quite standard) and consider OO in the following section.

The Turtle3D and its use in algorithmic programming

In the course of our school pilot project ”genetic computer science education” (cf. Schuster, 2011) we have been looking for a suitable environment for the introduction of algorithmics, that should open the scope for object oriented programming or at least does not block it. Classical turtle graphics could have been an option, but in our opinion the resulting pictures are not very motivating for high school students. As they are familiar with 3D representations in computer programs (games, …) it seemed obvious to use this as connection factor. Thus we developed a Python module (a library) that allows controlling a turtle in 3D space.

Features of the 3D turtle

The following code imports the library, creates a 3D turtle and executes some of its methods. The program paints a dashed line and a square. It can be executed interactively on the Python console or from a file.

```
from VTools import *
t=Turtle3D() # Creates a Turtle, with green pen color by default
t.forward(1)
t.penUp()
t.forward(1)
t.penDown()
t.forward(1)
t.turnRight(90)

t.setColor(color.blue)
for i in range(4):
    t.turnUp(90)
    t.forward(3)
t.forward(2)
```

This is the resulting image:

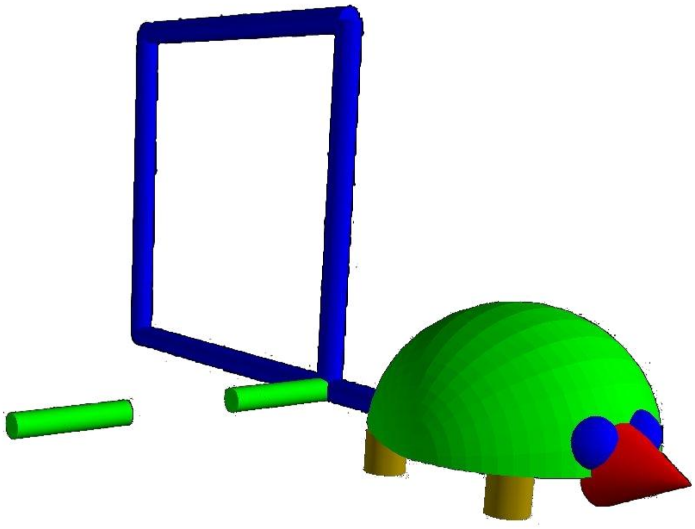


Figure 1. The turtle and a trace left by it

Only very few commands are needed to control the turtle. All turtle commands are methods that have to be applied to a Turtle3D object.

method	**action: the turtle …**	**example**
`forward(a)`	moves `a` steps forward.	`t.forward(5)`
`backward(a)`	moves `a` steps backwards.	`t.backward(5)`
`turnLeft(w)`	turns left by the denoted angle `w`.	`t.turnLeft(45)`
`turnRight(w)`	turns right by the denoted angle `w`.	`t.turnRight(45)`

`turnUp(w)`
`turnDown(w)`
	turns up / down by the denoted angle `w`.	`t.turnUp(45)`

`penUp()`
`penDown()`
	toggles its pen's status. The given example paints a dashed line.	`t.forward(30)`
		`t.penUp()`
		`t.forward(30)`
		`t.penDown()`
		`t.forward(30)`

`setVisible(b)`	If b is true, the turtle is visible.	`t.setVisible(False)`
`setAnimated(b)`	If b is true, the turtle moves slowly animated. If b is false, the resulting painting appears instantly.	`t.setAnimated(False)`

`setColor(col)`	Sets the color of the turtle, e.g. color.red, color.blue or	`t.setColor(color.yellow)`
		`t.setColor((1,0,0))`

	as rgb-values consisting of numbers ranging from 0 to 1.	`t.setColor((0.5,0.2,0))`
`setThickness(c)`	Sets the thickness of the line to c. The default thickness is 0.1.	`t.setThickness(0.5)`
`goto((x,y,z))`	moves to (x,y,z).	`t.goto((1,5,-3.2))`
`lookdir(v)`	looks into the direction of the given vector v.	`t.lookdir((0,1,0))`

Algorithmics with the 3D turtle

After the students have learnt the basic functionality, they can paint freely and express their creativity. Doing so, they can build several complex graphical objects with the turtle. The generated code may probably consist of many repeating blocks like the following but much longer (as we experienced in our teaching experiment, within one hour students can produce hundreds of lines):

```
t.forward(10)
t.turnRight(90)
t.forward(10)
t.turnRight(90)
t.forward(10)
t.turnRight(90)
```

While students are writing these kinds of repetitions (mainly by copying program text), they ask themselves if there is a way to automatically repeat the commands. This question emerges straight out of the student’s experience and leads a genetic way to the loop-concept that answers this question.

The elementary concept of reference through variables can be introduced quite naturally as well: The students’ wish to alter the dimensions of certain parts of the drawing (e.g. the side length of a cube) leads to the time-consuming and error-prone search in the code for places to change this length. It is better to previously specify variables to define them. By doing so the drawing is parameterised by the reference to the given value and can easily be repeated with other values.

Now, that students realize that certain operations (e.g. the drawing of a rectangle) reoccur again and again, the wish for procedural abstraction arises.

```
def rectangle(a,b):
    for i in range(2):
        t.forward(a)
        t.turnDown(90)
        t.forward(b)
        t.turnDown(90)
```

Other examples for reusable forms are regular polygons, circles, stars and so on. In the following example, rectangles are used to draw a cylinder.

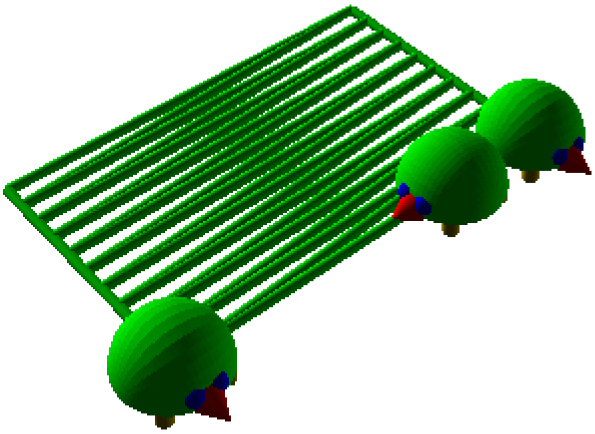
```
def cylinder(radius,height,n=150):
    for i   in range(n):
        t.backward(radius)
        rectangle(2*radius,height)
        t.forward(radius)
        t.turnLeft(360.0/n)
```

Definitions like these demonstrate an important role of functions: they are constructors (in a sense close to the meaning of constructor in object-oriented programming). They only differ from inherent constructors like Turtle3D() by the fact that the last mentioned ones return the constructed object so that it can be referenced later. In the case of the turtle this gives us the opportunity to have more than one turtle at a time.

Working with more than one turtle shows a serious disadvantage of the above definition: The code references always the same turtle. Thus the reusability is limited. The function should better be implemented as follows:

```
def rectangle(thisTurtle,a,b):
    for i in range(2):
        thisTurtle.forward(a)
        thisTurtle.turnDown (90)
        thisTurtle.forward(b)
        thisTurtle.turnDown(90)
```

For using more than one turtle there are several more or less meaningful applications. A less meaningful but instructive application is this: An area could be shaded by having two turtles drawing the outline and a third shading-turtle to walk from one to the other and back again (Fig. 1).



```
Ta=Turtle3D()
Tb=Turtle3D()
T=Turtle3D()

Ta.forward(8)
Ta.turnLeft(90)
Tb.turnLeft(90)

for i in range(10):
    Ta.forward(0.5)
    T.goto(Ta.pos)
    Tb.forward(0.5)
    T.goto(Tb.pos)

T.forward(2)
```

Figure 2. A shaded Rectangle built by three Turtles including the source code

A second perhaps more meaningful application is to implement a game of pursuit. An example can be found on our homepage.

OOP with Turtle3D

Using the turtle can also serve as preparation for object oriented programming. Each turtle has attributes (e.g. pen-color, pen-position, …) and own methods (forward, backward, …). These methods are already used the same way that will be relevant in object oriented programming: `objectName.methodName([args])`. To prevent misconceptions on distinguishing the concepts “class” and “object”, the students should use the opportunity to create two or more turtles from an early stage on. When introduced early, the students understand, why the turtle always has to be named (unlike in Logo) when calling a method and they understand `t1.penDown()` as an order to the turtle t1 to change its *and only its* state.

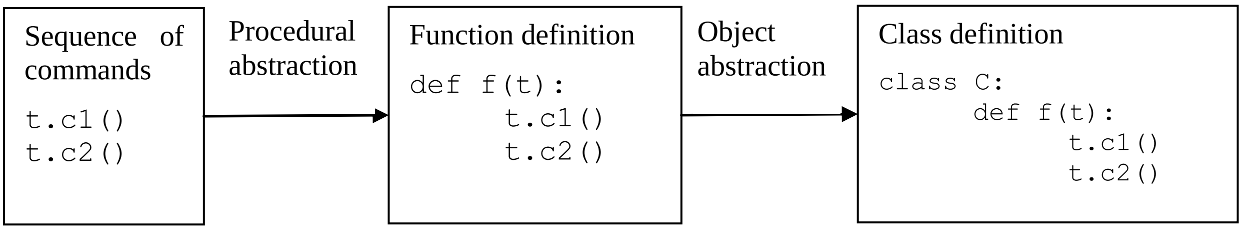
While using the turtle during our course, some students wanted to use their own written functions in exactly the same way as they used the built-in methods of the turtle – e.g. `t1.myFunction()`. If this wish shouldn't come up spontaneously it can be provoked by pointing out the difference between the "professional" build-in methods and the user's own procedures. Let's start with the following procedure to draw a circle (the rectangle above could serve as an example as well):

```
from VTools import *
from math import pi
t=Turtle3D()
t.setAnimated(False)

def circle(thisTurtle,radius):
    n=100
    for i in range(n):
        thisTurtle.turnLeft(360.0/n)
        thisTurtle.forward(2*pi*radius/n)

circle (t,5)
circle (t,8)
```

To have the drawing of circles on par with other built-in methods one would like to write e.g. `t.circle(5)`. Now, the restriction of Python (just like almost all other languages) is that you can't teach an old turtle new tricks (or in more technical language: one cannot add methods to a class or an object without changing the definition of the class). The way around is to create a new class of turtles that are cleverer than the original ones, a smarter descendant so to say:

```
class SmartTurtle(Turtle3D):
    def circle(thisTurtle,radius):
        n=100
        for i in range(n):
            thisTurtle.turnLeft(360.0/n)
            thisTurtle.forward(2*pi*radius/n)

t2=SmartTurtle()
t2.circle(10)
```

An important aspect that smoothes the passage to the OO language is that the definition to draw a circle could be repeated without change, it has just to be put into the body of a class statement. This step of abstraction is similar to the step of procedural abstraction used before:



```
t.c1()
t.c2()
```

```
def f(t):
        t.c1()
        t.c2()
```

```
class C:
      def f(t):
            t.c1()
            t.c2()
```

Figure 3. Steps of abstraction

A technical detail deserves attention: In Python the first argument of a method is usually called `self`. In the example above we have violated this convention in order to ease the transition from the purely procedural definition of `circle`. The advantage of this approach is that the role of this parameter becomes transparent. In the long run, however, it is advisable to urge the students to pick up the convention because `self` is a word that can be interpreted in almost all contexts.

An important feature of the introductory sequence described here is that inheritance is used at the beginning although it may seem that this is a more advanced concept that should be postponed. However, we profit from the fact that using objects from existing classes is very natural to students and poses hardly any difficulty. Taking objects from predefined classes is easy to understand as it complies with everyday knowledge of getting an object of a certain kind, e.g. getting a cup of coffee out of a coffee automat. Thus it is quite natural to work with things one already knows. In contrast, defining the first own class from scratch would is a much more abstract approach. Our approach is also close to the way OOP is used in most authentic applications (compare the arguments in Meyer, 2009 for using a large library from the beginning).

Forming a new, smarter class of objects by teaching them new things is a good mental image to understand the essence of classes and inheritance, it forms what we call basic conception (German “Grundvorstellung”, cf. Rabel, 2011). While many other conceptions (e.g. class as a building plan or as a factory) address the understanding of attributes, this view is especially good at giving insight into the relevance of methods. While defining global procedures is to teach the computer new things, defining a derived class with new methods is to define new behavior for specific objects.

Empirical Evidence

A variation of the outlined course has been taught to high school students at two German schools as part of the course on computer science.

During the introduction course to procedural programming by 3D turtle graphics a student was very interested in adding more functionality to the turtle objects we used. Due to course plan this wish had to be postponed for a while. But when the students were introduced to object oriented Programming this particular student remembered his wish from one year ago and started to extend the given turtle class. He proudly presented that extension in a short talk. In the following we provide a transcript of this presentation:

Student 1: Sometimes it just annoyed me that the turtle itself could not rotate around its own axis, which means you could only turn left or right or up and down and not around the nose. And then we found out that you just, uh, turn up ninety degrees and after that you can turn to the left or right, and then back down ninety degrees. As you see, I implemented here [see fig. 4] that you can simply call it [the rotate function] like all the other turtle functions instead of troublesome programming by yourself. And, um, I just happened to have stumbled on it, how the shape of the ... how the turtle’s shape is programmed. You can see it... here, well concealed from us, that there is another shape already: namely, the worm, all right? So here's the “TurtleForm”. This is the turtle itself, as it is programmed, and the mentioned “WormForm” and uh, I went through it a little bit, uh, how it works, and then I programmed a little bit by myself: The “QuentinForm”. Who knows Quentin [a figure from a German cartoon TV series]? Can everyone visualize that? Well, probably not. It is that yellow Ball.

Student 2 (from the audience): Ahhh, right!

Student 1: Then I have the ... uh ... Here you can ... Wait, I am going to label that first ... you can enter the shape and then subsequently I just put in the QuentinForm. The whole thing ... looks extremely funny, in my opinion ... And this is my new Turtle (smiles). So I find it extremely ... Yes, it is the result of boredom, but I think ... Now the mouth is gone ... There it is again. ... Well it came out of boredom and, uh, yes I think it is ... one sees that it is not ... not at all, not too difficult, to consider a new shape

... Yeah

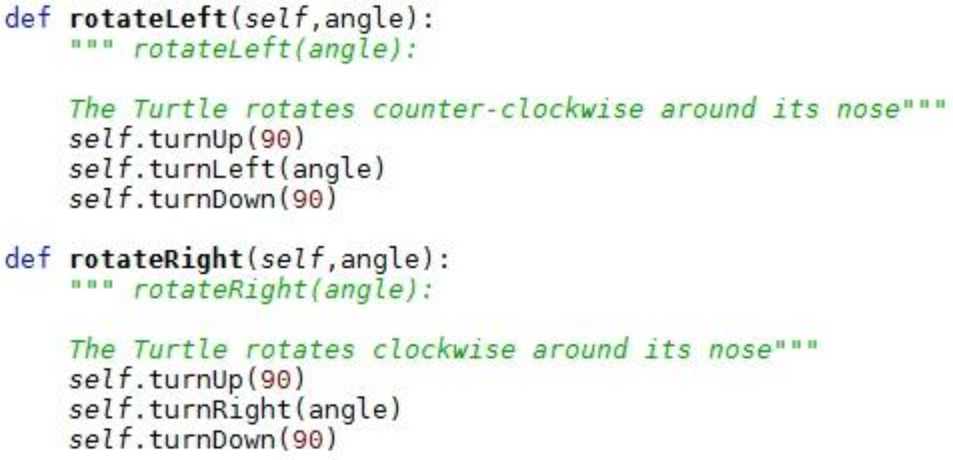
Teacher: Very nice. And this (sporadic applause) ... You´re right, applause, please. ... Um, a short question, um: Have you implemented the rotate-function?

Student 1: Yes.

Teacher: Would you execute it, please?

Student 1: So, if you look here now, you see “rotateLeft” and “rotateRight” ... given and then you just put in the angle, how much ... So, for example I put in 90, then it turns 90 degrees ... respectively the other way around ... (Murmur from the audience) ... Quite a lot (murmur) ... which doesn´t have any effect ... It is back on top ... So I think, you can do a lot more with that [turtle class].

Teacher: Yeah! So, the tournament has begun. All of you can develop further. Super!



```
def rotateLeft(self,angle):
    """ rotateLeft(angle):

    The Turtle rotates counter-clockwise around its nose"""
    self.turnUp(90)
    self.turnLeft(angle)
    self.turnDown(90)

def rotateRight(self,angle):
    """ rotateRight(angle):

    The Turtle rotates clockwise around its nose"""
    self.turnUp(90)
    self.turnRight(angle)
    self.turnDown(90)
```

Figure 4. The student´s source code

This transcript shows clearly that the concept of inheritance was appreciated by Student 1 as the concept that solved his problem and his success was honored by others.

The episode illustrates as well a principle of our genetic teaching style: We try to introduce new concepts not quickly. Instead we try to get the students involved in situations where they get aware of problems to be solved. New concepts introduced are the solutions to problems not just new theory waiting for an example of application.

Conclusion

The teaching experiment described in this paper provides evidence that the 3D turtle is a motivating tool for high school students of age 16-17. There are further possibilities to keep the subject interesting: Listening to key events and detecting collision of graphical objects is easily done and allows the implementation of simple games. The visual effect can be made even stronger when using stereo display. This is sufficient for a unit of several weeks that touches all basics of algorithmics and OOP. Of course, the knowledge acquired there must afterwards be transferred to other application areas, e.g. to non-visible objects.

Acknowledgements

We gratefully acknowledge financial support for the teaching experiment from Goethe University and organization support from the ministry of education. Furthermore, we thank the two high schools that made this teaching experiment possible.

References

Elica. Retrieved from http://www.elica.net/

NetLogo 3D. Retrieved from http://ccl.northwestern.edu/netlogo/3d/docs/threed/3d.html

Abelson, H., & DiSessa, A. A. (1981). *Turtle geometry: The computer as a medium for exploring mathematics*. *The MIT Press series in artificial intelligence*. Cambridge, Mass: MIT Press.

Heid, M. K., & Blume, G. W. (2008). *Research on technology and the teaching and learning of mathematics*. Charlotte, NC: Information Age Publishing.

Hromkovič, J. (2010). *Einführung in die Programmierung mit LOGO: Lehrbuch für Unterricht und Selbststudium* (1st ed.). Wiesbaden: Vieweg + Teubner.

Meyer, B. (2009). *Touch of class: Learning to program well with objects and contracts*. Berlin [u.a.]: Springer.

Papert, S. (1993). *Mindstorms: Children, computers, and powerful ideas* (2nd ed.). New York: Basic Books.

Paysan, B. (1999). *"Dragon Graphics": Forth, OpenGL und 3D-Turtle-Graphics*. Retrieved from http://bernd-paysan.de/dragongraphics-eng.pdf

Rabel, M. (2011). Grundvorstellungen in der Informatik. In M. Weigend, M. Thomas, & F. Otte (Eds.), *Informatik mit Kopf, Herz und Hand. Praxisbeiträge zur Infos 2011* (pp. 61–70). Münster: ZfL-Verlag.

Schaub, S. (2000). Teaching Java with Graphics in CS1. *ACM SIGCSE Bulletin*, *32*(2), 71–73. doi:10.1145/355354.571919

Schuster, J. (2011). Ein genetischer Zugang zum Programmieren mit CGI-Skripten in Python. In M. Thomas (Ed.), *GI Proceedings 189 Informatik in Bildung und Beruf. INFOS 2011 14. GI-Fachtagung Informatik und Schule, 12.-15.09.2011 Münster* (pp. 227–236). Bonn: Köllen.

Wolfram Demonstrations Project. *3D Flying Pipe-Laying Turtle*. Retrieved from http://demonstrations.wolfram.com/3DFlyingPipeLayingTurtle/