Turtle Graphics Programming with Newt

by Steve Weyer
Version 1.2, 4/24/95

The "Newt" discussed in this article is a shareware native programming environment on the Newton. Like a chameleon, Newt has evolved to provide different functionality to different users. The first version of Newt in Oct. '93, inspired by the Inspector Gadget and Dot2Dot examples from Apple, allowed you to draw graphics using NewtonScript — the turtle, its amphibious cousin and name inspiration, used Logo. You, the learner, could explore mathematics via a turtle microworld, or add NewtonScript methods to emulate Logo commands and data structures. This book will let you explore Newt as a graphics programming environment. NewtTurT is freeware and may be freely distributed on line services as long as it is unmodified and includes the file NewtTurT.1st.

With enthusiastic feedback from early users, I shifted efforts and emphasis so that you could create objects based on Newton interface prototypes and save an application. Newt's "application personality" will not be covered in this article. However, if you are interested in building applications on your Newton with interface objects such as buttons, text fields and checkboxes, look for the related Newton book: **NewtATut**: *Building Newton Applications with Newt.*

Using Newt from this book

For more interactivity, NewtTurT 1.2 (book) requires Newt 3.0 (or greater) plus the NewtDraw plug-in. See ***Where to Find Newt and Further Information***. This article may also available as an Acrobat PDF file (NewtTurT.pdf), which requires Adobe Acrobat Reader on your desktop system.

Newt's icon reflects both its original turtle personality (on left) as well as its later application personality (on right).

If you have the appropriate version of Newt installed, you can tap on its icon above to start it — if successful, Newt opens, then a few seconds later, the book reappears automatically. You can access ("fall into") Newt later by closing this book. You can return to this book by selecting its title "Turtle Graphics Programming with Newt" from Newt's overview list (bottom dot between two scroll arrows).

Later in this book, you can tap on an underlined expression or method to copy and evaluate or define it to Newt. (If you have the correct version of Newt installed, the examples will highlight). Tapping a method (several lines of code beginning with a bold **`methodname`** and a `func` statement) copies it to Newt's method editor area, compiles it as code, and saves it as text in the Notepad — while you remain in the book. Tapping an expression copies it to Newt's short Eval

field, evaluates it, and exits the book since there is usually a visible result — you can re-enter the book by selecting its title from Newt's overview.

Turtles and Logo

I would like to provide some brief history and context — the first two topics — before focusing on the third:

• past— the history of turtles and Logo

• pedagogy — how students learn by engaging in programming and problem solving

• practical — how to use Newt to draw a picture and write a method

The original turtle began life in the early 70's as a floor robot that could draw pictures when the user gave it commands like "forward", "right", "penup" and "pendown". By using information from its touch sensors, the turtle could be programmed to maneuver around objects. Turtles evolved and inspired work in several directions: faster and more accurate screen turtles; animation and simulation environments; controllers such as a "button box" for use without a programming language; and other physical environments like LegoLogo. In LegoLogo, students construct vehicles and devices using motors, lights and sensors which are controlled by programs. As wireless control systems become economical, and as the Newton's infrared communication range increases, it will be fun to use Newt and Newton as a remote controller.

Although turtles are practically synonymous with the Logo programming language, other languages such as Lisp and Smalltalk have also implemented turtle objects and environments. In addition to graphics, Logo (and other languages) have been used to teach children other aspects of programming, e.g., list processing and pattern matching. Logo is similar to Lisp in its structure and expressiveness, but much simpler in syntax.

The Newt environment uses NewtonScript (NS), Newton's built-in object-oriented language. NS shares ancestry with dynamic object-oriented languages such as Smalltalk, Self, Common Lisp, and ObjectLogo. For more advanced users, familiarity with object-oriented concepts and constructs is useful; for those familiar with Logo, it is easy to translate ideas and code; for others, Newt serves as a brief introduction to these ideas.

Learning by Children of all Ages

Seymour Papert at MIT, applying the learning theories of Swiss psychologist Jean Piaget, used the turtle as a "mathematics microworld" that children could explore by writing Logo programs. Piaget's experiments identified three modalities of learning: enactive, iconic and symbolic. At early ages, children learn most naturally through concrete aspects or physical manipulation, i.e., by touching and doing. Later, they are ready to interpret and reason with the help of other senses, e.g., by seeing and hearing. Finally, they are ready to reason using abstract concepts and symbols, in addition to the earlier learning styles.

In the turtle microworld, the students integrated information from the physical domain of the turtle, visual feedback from graphics, and symbolic representations via programs. Students role-played the turtle in order to experience a relative coordinate system and to understand how

programs might not reflect what the student had intended. The students engaged in planning, problem solving and "learning by debugging". They learned by breaking problems down into subproblems in order to use recursion.

Over the years, many people have been involved in Logo and similar learning environments. I will note just a few, and apologize in advance to those I have omitted: Logo and LegoLogo at MIT (Papert), Smalltalk at Xerox PARC and Vivarium at Apple (Alan Kay), Boxer at Berkeley (diSessa), and numerous other projects in Logo (BBN, Bank Street, Stanford). In the hands of a well-trained coach or teacher, such exploratory environments go beyond just programming or drawing pictures toward an experiential understanding of mathematics.

Newt follows in the footsteps of these earlier efforts by providing a portable, turtle-like learning environment.

Getting Started with Newt

Newt requires only a willingness to learn — not any programming experience. Much of what I will describe here can also be found in turtle.txt file and other files that accompany Newt (see **Where to Find Newt and Further Information**), or is discussed in more detail in the manual if you register.

You can draw with Newt using pen gestures:

• *tap* to draw a line to current position

• *drag* to move Newt without drawing

• *line* to change Newt's direction or draw a line along the same heading

• *scrub* to erase the drawing

If you would like to try this in Newt, close this book. A Newt symbol should be visible in the center of the screen. Remember that you can return to this book by selecting its title from the overview list. You can also re-open the *Newt Drawing* area and *Newt Controls* palette from the same overview if you accidentally close them.

You could now try drawing using buttons and numbers. For example, write in a number, e.g., 80, to the right of the *Dist* button in *Newt Controls*. Next, tap *Dist*. Newt draws a line.

Now, write a number to the right of *Deg*, e.g., 90. Tap *Deg*. Newt turns.

You can alternate tapping *Dist* and *Deg* to create a closed polygon or other figure.

Write in a number to the right of *Times*. Tap *Times* to repeat a series of *Dist* movements and *Deg* turns. Newt writes the NewtonScript expression into the *Eval* field and evaluates it.

Tap *Erase* or use the scrub gesture to clear the drawing area.

Tap *Home* to return Newt to center.

Here are some "interesting" values to try:

Times	Dist	Deg
5	40	72
6	40	60
8	40	45
9	40	40

Here are some other interesting values. Can you discover a pattern between the number of sides and degrees for each set of values?

Times	Dist	Deg
5	70	144
9	70	80
9	70	160
9	70	160
15	70	48
15	70	96
15	70	168

If you use larger values for *Dist*, Newt will clip the line at the edge of its drawing area. Negative numbers mean go backwards. If you use larger values for *Deg*, Newt will just ignore multiples of 360 and use the remainder. Negative values for *Deg* mean turn left (counterclockwise) instead of right. Unless you want to use the reset button or wait awhile, you should not use very large values for *Times* since the default behavior for Newt is to finish executing the entire expression before checking for user actions like button taps. There is a facility provided in Newt to replace the NewtonScript "for" iteration construct. This facility executes each step in the background and allows you to interrupt Newt — that's discussed in the manual and perhaps in a future article.

Consumer Alert!

Before delving into programming details, I offer a warning. Apple and Newton developers are still learning how to implement system and application software. In addition, while learning with Newt, you will take risks and make mistakes. For example, Newt allows you to execute arbitrary NewtonScript expressions. This can be a formula for enlightenment or disaster. So, remember that this is your Newton with your information and listen to your mother's advice about backing up your system.

Using common sense in following examples and suggestions, and limiting yourself to documented commands, you should reap many benefits and long hours of enjoyment from using Newt. At the same time, "a little learning [about programming] is a dangerous thing" (with apologies to Alexander Pope). I would caution against experimenting with random functions or methods. "Gee, I wonder what xxx does" might yield a simple error box, or it might zap a frame or soup in your system.

This tutorial is only a brief introduction to Newt and even less for NewtonScript. As noted elsewhere, look at Newt files and later in this article for pointers to further information.

Writing Methods

You can write a method to group the commands that you were executing one-at-time yourself. For example, if you are a Logo fan, you might prefer to use **`forward`** instead of **`go`**, and **`right`** instead of **`turn`** (tap on a method to define it):

```
forward
func(distance)
:go(distance)
```

```
right
func(angle)
:turn(angle)
```

You can now invoke these methods in expressions or other methods, for example:

```
local i; for i:=1 to 5 do begin :forward(100); :right(144); end
```

Here is a method that defines a polygon (using the usual go and turn):

```
poly
func(nsides, len)
if nsides > 0
then begin
  local i, ang := 360/nsides;
  for i:=1 to nsides
  do begin
    :go(len); :turn(ang);
    end;
  end
else :beep()
```

To make things easier for you, this book tells Newt to do Erase and Home before evaluating expressions, for example:

```
:poly(5,70)
```

You can use variables for fields like *Times*, *Dist*, and *Deg*. The following nested polygons uses the dist variable that corresponds to the number in the *Dist* field. It uses times to limit iteration:

```
local sides; for sides:=3 to times do :poly(sides,dist)
```

If Newt did not draw anything, was dist non-zero? Once an expression is in the Eval field in Newt, you can edit it and/or tap the Eval button to re-evaluate it.

```
:poly(Random(3,15),Random(10,20))
```

Here is a noisy figure:

```
local i; for i:=5 to 160 by 5 do begin :go(i); :turn(90); :beep(); end
```

Here is a classic squiral method involving rotated squares:

```
squiral
func (num,len)
// draw <num> squares
// each of size <len>
// after each, turn 360/num
if num>0
then begin
  local i, ang2 := 360/num;
  for i:=1 to num
  do begin :poly (4,len); :turn(ang2); end;
  end
else :beep()
```

Ten rotated squares:

```
:squiral(10,75)
```

cCurve draws a fractal-like pattern.

```
cCurve
func(num,dist,deg)
if num>0
then begin
  :cCurve(num-1, dist, deg+45);
  :cCurve(num-1, dist, deg-45);
  end
else begin
  :turnTo(deg);
  :go(dist);
  end
```

Since cCurve uses a double recursion (not a simple "tail recursion"), avoid large values for the first parameter. If you find a pattern that you like, you can tap the *Save* button (in *Eval Controls*), and Newt will save the current picture to your Notepad.

```
:cCurve(6,8,0)
```

You can also have multiple Newts. This expression creates four of them:

```
:addNewt('[[newt1,-55,75,1],[newt2,55,75,2],[newt3,55,-75,1],[newt4,-55,-75,2]],nil,nil,nil)
```

Generally, messages are sent to the default Newt object. Now as a default, the message is broadcast to the collection of Newts. (This is covered in more detail in the manual). Try this expression to see how multiple Newts draw in tandem:

```
:squiral(8,40)
```

Or, if you would like to see four Newts drawing doubly-recursive cCurves in parallel (you may wish to exit to Newt first and turn off the Newt? checkbox for faster drawing):

```
:cCurve(6,4,0)
```

Or, you can send a message to a specific object

```
newt1:poly(6,20)
```

This expression removes multiple Newts, returning you to the lonely default Newt:

```
:removeNewt(all Newts)
```

Finally, an interesting variation on **poly** where the side increments each time:

```
polyspi
func (n,len,ang)
begin
  local i;
  for i:=len to len+n by 1
  do begin
    :go (i); :turn (ang);
    end;
end
```

Here is an example expression to try:

```
:polyspi(80,25,89)
```

In order to keep this book relatively small, I will now end the tour of Newt's graphics environment. You can experiment on your own with Newt directly using *turtle.txt* and *graphic0.nwt* for a partial guide. If you are intrigued by the possibilities of programming your own Newton to create graphics or applications, I would encourage to register so that you can receive a more detailed manual and set of examples.

Where to Find Newt and Further Information

You should be able to obtain Newt 3.0 (or the latest version) from the following online sources (usually as filenames similar to newt-devenv-30.sit/.hqx or newt30.sit/.zip):

- America Online(AOL): PDA:Software Libraries:Newton
- Compuserve: GO NEWTON (DL 8 or 9)
- eWorld: ShareWare:Newton
- Internet (anonymous ftp):

  -newton.uiowa.edu/pub/newton/software/dev or /app (or /submissions)
  -ftp.amug.org/pub/newton
  -sumex-aim.stanford.edu/info-mac/nwt/dev
- Usenet newsgroup: comp.binaries.newton

Registered Newt users receive a 70 pp. manual that introduces NewtonScript and describes Newt commands and methods, a set of 160+ examples, notification and discussion of future releases, and relief from shareware procrastination and guilt.

The North Atlanta Newton User Group (NANUG) newsletter *_protoReality 1.3*, available on many networks, contains an interview with me and a turtle-oriented article by my daughter. Erica

Sadun reviewed Newt as turtle in *PIE Developers*, Vol. 2.4, July 1994. Finally, I welcome comments and suggestions. You can contact me via one of several email addresses:

- weyer@netaxs.com
- AmericaOnline, eWorld, NewtonMail: SteveWeyer
- CompuServe: 74603,2051
- http://www.netaxs.com/~weyer (my home page with latest Newt info)

Bio

Over the past 20+ years, Steve has implemented and managed R&D projects involving object-oriented languages and prototyping environments, AI tools, hypertext systems and education. When not borrowing time from his family to work on Newton applications and generally recovering from the culture shock of transplanting from Silicon Valley to rural Pennsylvania, he consults for a pharmaceutical client on enabling technologies including pen-based systems.