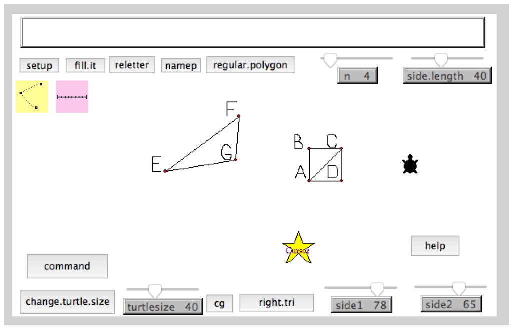
Build You Own Geometry Toolkit with MicroWorlds

©1995-2016
Gary S. Stager, Ph.D.

Objectives

Students (middle and high school) will use MicroWorlds EX create their own tool for exploring two-dimensional geometry similar to Geometers' Sketchpad, Cabri, or GeoGabra. [1]

As students build functionality (via programming) into a tool for creating and measuring geometric constructions, they reinforce their understanding of important geometric concepts. As the tool gets more sophisticated, students learn more geometry, which in turn leads to a desire to explore more complex geometric issues. This is an ecological approach to programming. The tool gets better as you learn more and you learn more as the tool becomes more sophisticated.



Along the way, students become better programmers while using variables, list processing, and recursion in their Logo procedures. They will also engage in user interface design.

Introducing the Activity

A starter MicroWorlds project (file) is provided to each student or team working on this project. That file contains graphics for each letter of the alphabet, used for labeling points on a figure, and starter procedures that would be too hard for most students to write themselves. These starter procedures are building blocks upon which powerful ideas in geometry and computer science may be built.

If at any point students do not like the turtle shapes provided, they may recreate them. Throughout the project, students will be asked to decide on which new features to add and how they will appear to the end user. The syntactical protocols used to define lists of points and other key data were chosen for ease of use, but are also subject to modification.

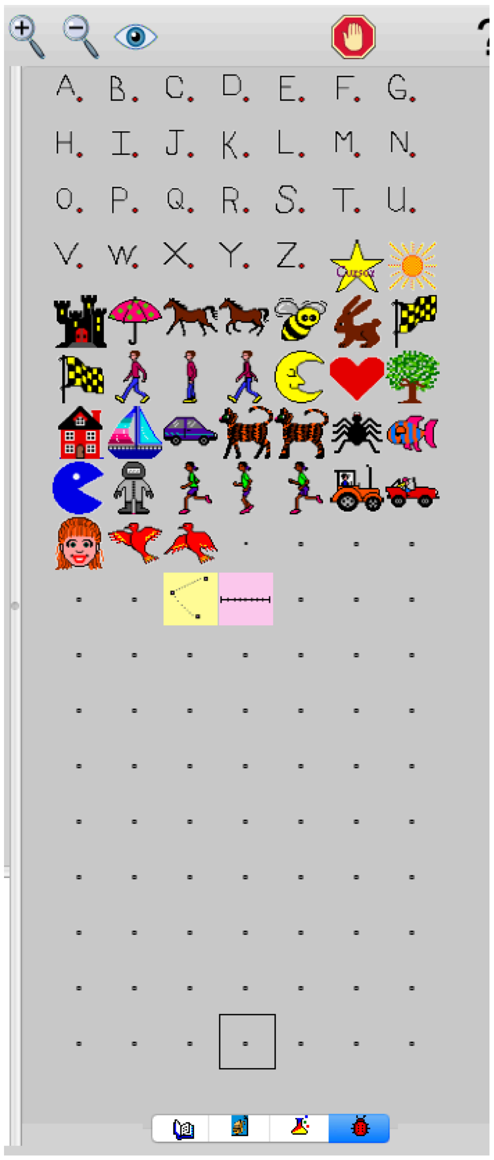
We essentially begin this design project by moving a cursor (a turtle), dropping a point on the screen, moving to another position, dropping a second point, and asking the computer to:

1. Draw a segment between the two points
2. Report the length of that segment

[1] I would not show commercial models of the software to students until after they have programmed some new functionality into their own tools.

Getting Started

- Figures will be constructed from a collection of points.
- Each point is a turtle capable of being moved and reporting where it is on the page.



Create 26 shapes in the shapes center containing a point in the middle and one letter of the alphabet beside it. (Copy and paste similar shapes to save time) Be sure to name each point with the name of its letter. You may also wish to create a point shape with no label and name it *point*. If you start with the `toolkitstarter` project, you don't need to create these shapes OR you can make them more attractive. The project's shape center is found in the tab with the beetle on it.

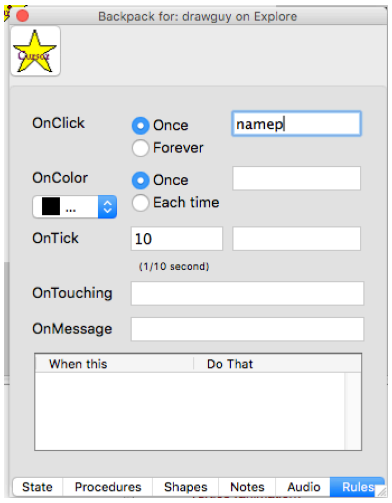
Create a shape for the cursor turtle to wear. I chose the star since it stands out nicely. How about a little mathematician turtle?

Create a textbox that tell the user the name of the last point created. Name that text box *`Last.Point`*

The cursor turtle's name in the project template is `drawguy.`

Drag the cursor turtle to your desired position and type `namep` in the command center to drop a point where the cursor turtle is. Can you program the cursor turtle to run the `namep` procedure when that turtle is clicked on?

Remember that procedure names may not contain spaces. Therefore, smush words together or use a period while naming things.


Backpack for: drawguy on Explore

OnClick [✓ Once] Forever namep

OnColor [✓ Once] Each time

OnTick 10 (1/10 second)

OnTouching

OnMessage

When this | Do That

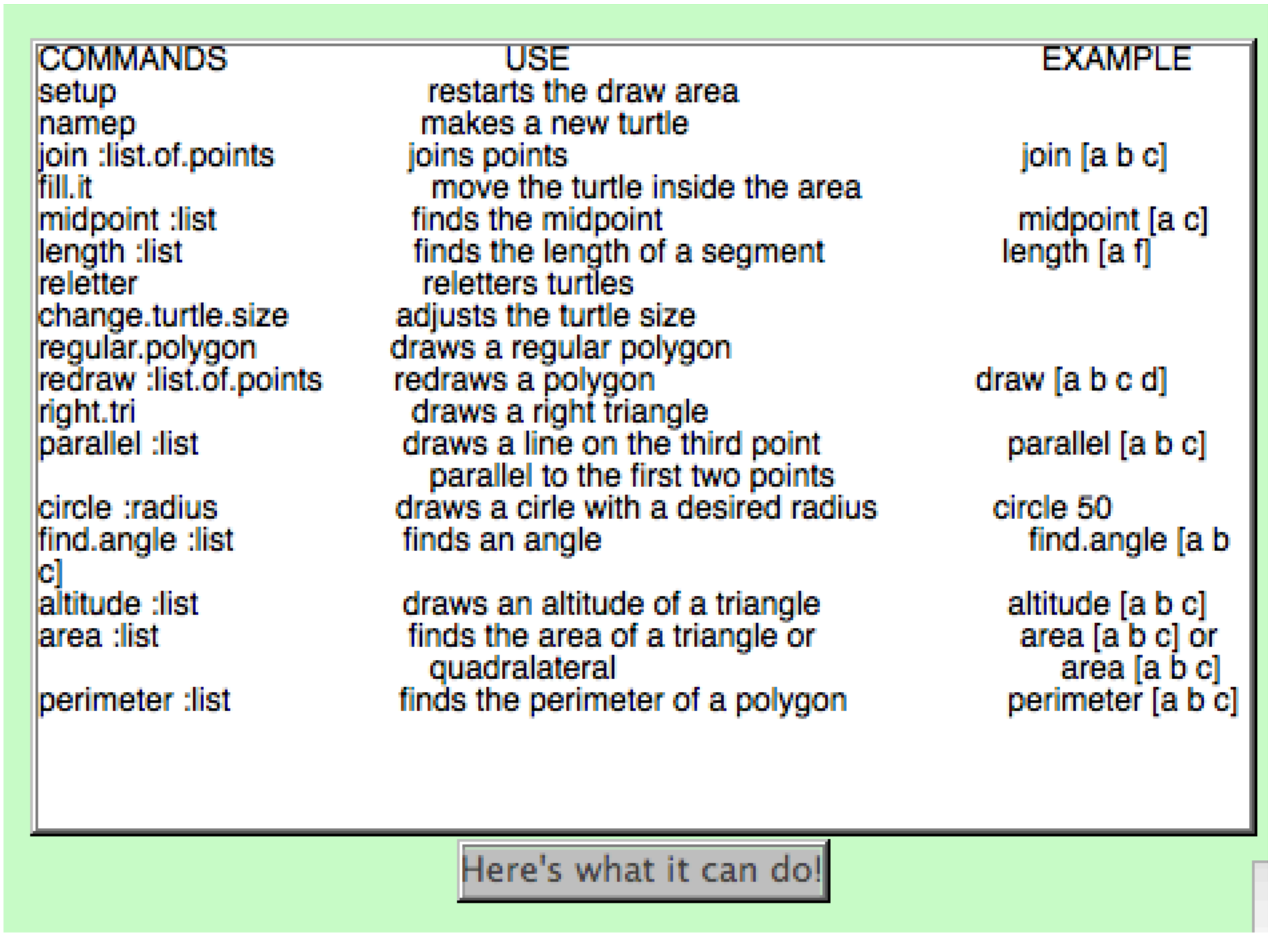
The procedure `join` takes a list of points as input and draws segments between them. `Join [a b]` creates a line segment. `Join [a b c a]` connects three points and returns to the origin.

As you add functionality to your project, create "help" instructions on a separate page to assist users. You may name the page `HELP` and create a button that goes to that page. On the `HELP` page, you might have a button taking users back to the geometry toolkit page.

Challenges

1. Can you create a button to draw any polygon? You will need to write a procedure to do the thinking and drawing.
   Use `Question` or sliders to provide input to the polygon procedure.
2. Can you make a button(s) to draw different types of triangles?
3. What sorts of other figures should your toolkit be able to draw?
4. Create a turtle or button to join a list of points using the `join` procedure.
5. What sorts of ways do you want to measure and compare your figures? Can you write procedures to find area, circumference, length, etc? How about to evaluate conditions like congruence?
6. Can you provide some information on the name and position of the current point?
7. Is there some way you can make the alphabetic label on a point disappear when there is a crowded drawing and reappear at other times? In other word, create something like a `hidepoints/showpoints` button?

The following is an example of the sort of HELP page you might create for users.



COMMANDS	USE	EXAMPLE
setup	restarts the draw area	
namep	makes a new turtle	
join :list.of.points	joins points	join [a b c]
fill.it	move the turtle inside the area	
midpoint :list	finds the midpoint	midpoint [a c]
length :list	finds the length of a segment	length [a f]
reletter	reletters turtles	
change.turtle.size	adjusts the turtle size	
regular.polygon	draws a regular polygon	
redraw :list.of.points	redraws a polygon	draw [a b c d]
right.tri	draws a right triangle	
parallel :list	draws a line on the third point parallel to the first two points	parallel [a b c]
circle :radius	draws a cirle with a desired radius	circle 50
find.angle :list c]	finds an angle	find.angle [a b
altitude :list	draws an altitude of a triangle	altitude [a b c]
area :list	finds the area of a triangle or quadralateral	area [a b c] or area [a b c]
perimeter :list	finds the perimeter of a polygon	perimeter [a b c]

Key Procedures with Explanations

These procedures are found in the template file, but their purposes are explained here.

`to startup`
`reletter`
`end`
	Startup automatically runs when the project is loaded. The letters list is rebuilt.

`to setup`
`cg`
`reletter`
`newturtle "cursor`
`end`
	Reset the letters list and hatch a turtle named, cursor.

`to reletter`
`clearnames`
`make "letters [A B C D E F G H I J K L M N O P Q R S T U V W X Y Z]`
`end`
	Reletter builds a list called, letters, containing the alphabet. First it clears all global variables and then defines one called, letters. Each new point will be named based on what remains in Letters.

`to turtles`
`get first pagelist "turtles`
`end`
	Turtles is a useful tool that reports a list of all active turtles on the current page

`to namep`
`cursor,`
`make "position pos`
`setlast.point first :letters`
`newturtle first :letters`
`setpos :position`
`setsh first :letters`
`make "letters bf :letters`
`st`
`cursor,`
	Namep is a very important procedure used to create new points on the page. First, we talk to the turtle named cursor and remember it's position in the variable, position. Then we hatch a new turtle and name it whatever the first letter is in the letters list. Then the new point moves to the location of the cursor and its shape is set to the point it represents. Then the letters list is made shorter by removing the first letter. We then tell the cursor that to listen for instructions.

`end`	

`to join`
`:list.of.points`
`cursor,`
`pu setpos ask first`
`:list.of.points [pos]`
`join.them bf`
`:list.of.points`
`end`
 	
`to join.them :list`
`if empty? :list`
`[stop]`
`cursor,`
`pd setpos ask first`
`:list [pos]`
`join.them bf :list`
`end`
 	
`to length :list`
`if not (count :list)`
`= 2 [stop]`
`talkto first :list`
`make "foo distance`
`last :list`
`cursor,`
`output :foo`
`end`
	Join takes a list of points as input.
	
	Ask the cursor to move to the position of the first of the points.
	
	Hand the join.them procedure all but the first point.
	
	Join the remaining points.
	
	Check to see if the list is empty.
	
	Talkto the turtle named, cursor.
	
	Draw a line to the next point.
	
	Run the procedure again - minus the first point.
	
	Length takes a list of two points as input - make
	sure there are only 2!
	
	Talkto the first of the points (like a ,)
	Store the distance between the two points in the global variable, foo.
	
	Talk to the cursor turtle.
	Output the length between the two points.

An Example of a Personal Tool Procedure

```
to square :length
repeat 4 [namep fd :length rt 90]
end
```

Helpful procedures
The `clearall` procedure gets rid of all the turtles, except for the cursor when you wish to start with a fresh screen.

```
TO CLEARALL
clean
kill turtles
reletter
end
```

```
to kill :list
if empty? :list [stop]
if not (first :list) = "cursor [remove first :list] kill bf :list
end
```

Once you create reporter procedures that calculate measurements of your constructions, you can use a procedure like `makemeasurement` in a button or turtle as part of your interface.

```
to makeameasurement
question [Type the measurement expression you want?]
announce run answer
end
```

The following procedure is an example of how you might program your toolkit to take two points as input and draw the midpoint of the segment between the two points.

```
to midpoint :list
if not (count :list) = 2 [stop]
drawguy, pu
make "oldpos pos
setpos where first :list
towards last :list
fd (distance (last :list)) / 2
notch
drawguy, setpos :oldpos
end
```

```
to notch
pd setc 9
rt 90 fd 5 bk 10 fd 5 lt 90
pu
namep
end
```

A procedure, such as `command`, might be attached to a button of the same name and be used as an interface feature, allowing the user to click on the button and type in an instruction to run without having to use the MicroWorlds command center.

```
to command
question [Type in the expression you wish to run]
run answer
end
```

The `fill.it` procedure fills an adjacent area in with a random color and texture.

```
to fill.it
drawguy, setc 10 + random 137
setpenpat 1 + random 38
fill
setc "black
setpenpat 1
end
```

An Explanation of MicroWorlds EX Primitives that May be New to You

`talkto (tto)` *`turtle-or-text-box`*
Makes the turtle or text box current. This command has the same effect as typing the name of a turtle or text box followed by a comma.

Examples:

```
talkto "t1
fd 50
talkto "text1
print "hello
```

`newturtle` *`name`*
Creates a new turtle with the name indicated. The new turtle appears at the position [0 0].

Example:
```
newturtle "Shelly
```

`distance turtle-name`
Reports the distance between the current turtle and the turtle indicated.

Example:
In this example, there are two turtles on the page. t1,
```
show distance "t2
122
```

Your answer will be different.

```
towards "t2
fd distance "t2 T1 meets t2.
```
Set t1 to go Many Times and define the go procedure as follows. T1 will be "trapped" around t2:

```
to go
fd 1
if 100 < distance "t2 [towards "t2]
end
```

towards turtle-name
Sets the heading of the current turtle to aim towards the one whose name is given as input.
See `distance .`

Example:

```
t1,
towards "t2 T1 faces t2.
fd distance "t2 T1 meets t2.
```

`butfirst (bf) word-or-list`
Reports all but the first component of a word or list.

Examples:

```
show butfirst [0 1 2 3]
1 2 3
show butfirst [hello there]
there
```

`make word word-or-list`
Creates a global variable and gives it a value, either a number, word or list.

Example:

```
make "class [Peter Dennis Geni]
show :class
Peter Dennis Geni
```