Chapter 1

Various tips

Due to the Dr. Geo integration in the Pharo Smalltalk environments, there are a few genies we can invoke. Most of them are hidden to the user. It is not we want to restraint its use, instead we don't want to overload the user when discovering Dr. Geo. As we will show in the following sections, these genies can be invoked from menus, keyboard shortcuts or Smalltalk codes.

1.1 Programming

In this section we present a few tools to use when writing Smalltalk script or sketch: the workspace, the debugger, the inspector, etc.

1.1.1 Workspace

To show it, click on the Dr. Geo environment background then do Ctrl-k[1].

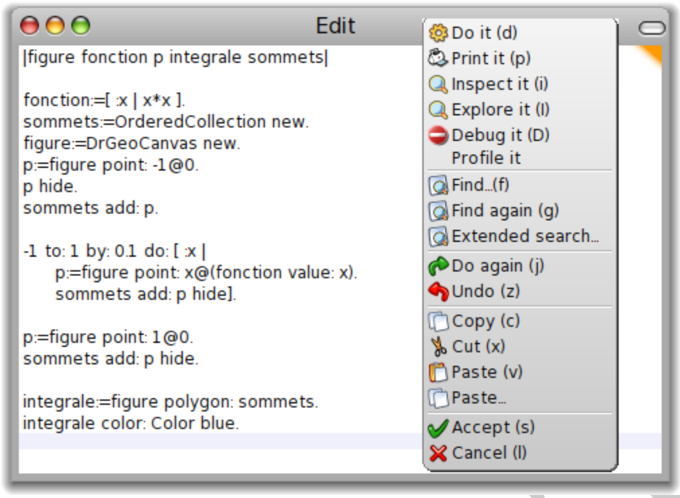
A workspace, at a first glance, is like a text editor. But it is in fact a console to edit Smalltalk code: to write it, to compile and to execute it; it is of course possible to paste code copied somewhere else.

After its invocation, paste the source code of the Smalltalk sketch bellow[2] :

```
| sketch function p integral summits |

function := [:x | x * x ].
summits := OrderedCollection new.
sketch := DrGeoCanvas new.
p := sketch point: -1@0.
p hide.
summits add: p.

-1 to: 1 by: 0.1 do: [:x |
   p := sketch point: x @ (function value: x).
   summits add: p hide].

p := sketch point: 1@0.
summits add: p hide.

integral := sketch polygon: summits.
integral color: Color blue.
```

[1]Depending one your system, replace Ctrl by Alt.

[2]To paste a text, try with the shortcut Ctrl-v or from its contextual menu (right click).



```
|figure fonction p integrale sommets|

fonction:=[ :x | x*x ].
sommets:=OrderedCollection new.
figure:=DrGeoCanvas new.
p:=figure point: -1@0.
p hide.
sommets add: p.

-1 to: 1 by: 0.1 do: [ :x |
     p:=figure point: x@(fonction value: x).
     sommets add: p hide].

p:=figure point: 1@0.
sommets add: p hide.

integrale:=figure polygon: sommets.
integrale color: Color blue.
```

Figure 1.1: Your workspace with the pasted source code and its contextual menu

To compile and to execute this source code, just select it with the mouse and invoke *Do it(d)* in the contextual menu. These two operations can also be done at the keyboard: $\boxed{\text{CTRL-A}}$ to select all the source code then $\boxed{\text{CTRL-D}}$ to execute it. You immediately get the result of this code, an interactive programmed sketch.

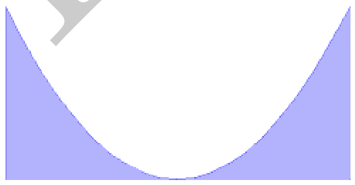
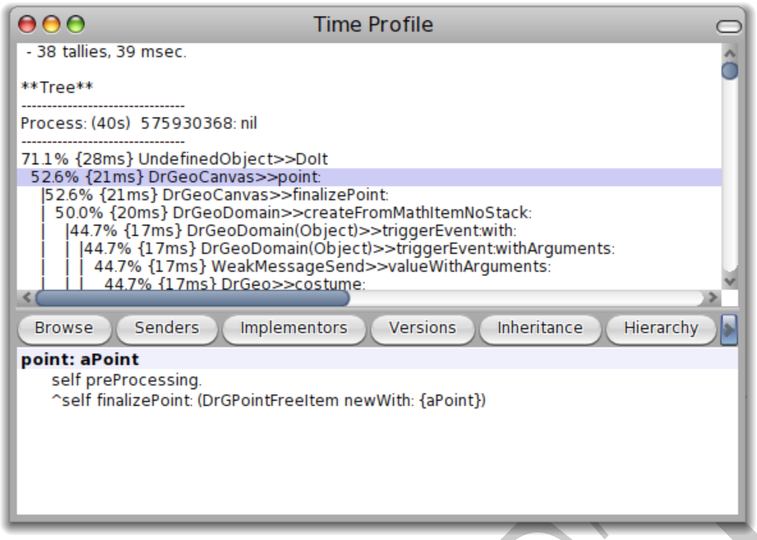


Figure 1.2: Result after executing the source code: the integral of the function in [-1 ; 1]

When executing a complex source code, running it with a profiling option let you see its bottlenecks. To do so, in the contextual menu invoke the command *Profile it*. The source code is executed, the sketch is built then a profile windows informs you about the execution time in different part of the code and the various invoked methods. It is a wonderful to navigate in execution tree of the code and look at the method consuming to much cycle.

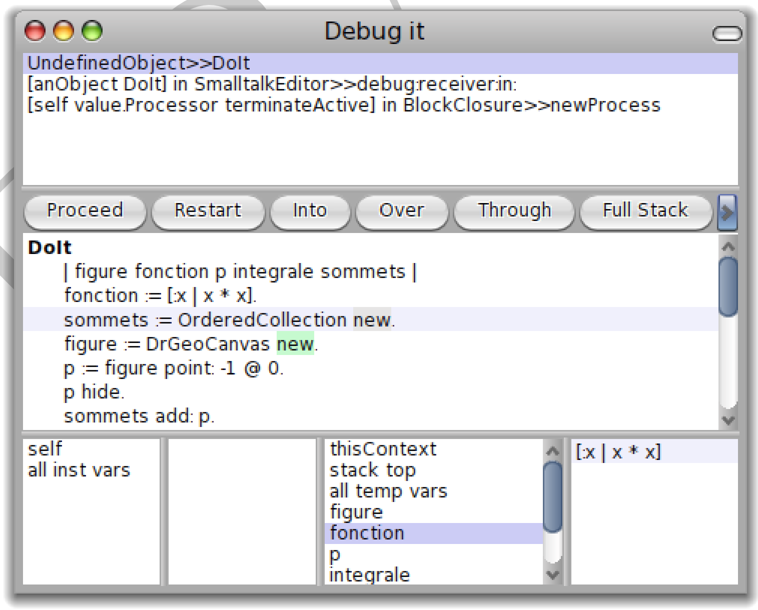


```
- 38 tallies, 39 msec.

**Tree**
--------------------------------
Process: (40s)  575930368: nil
--------------------------------
71.1% {28ms} UndefinedObject>>DoIt
  52.6% {21ms} DrGeoCanvas>>point:
    |52.6% {21ms} DrGeoCanvas>>finalizePoint:
    |  50.0% {20ms} DrGeoDomain>>createFromMathItemNoStack:
    |    |44.7% {17ms} DrGeoDomain(Object)>>triggerEvent:with:
    |    |  |44.7% {17ms} DrGeoDomain(Object)>>triggerEvent:withArguments:
    |    |  |  44.7% {17ms} WeakMessageSend>>valueWithArguments:
    |    |  |    44.7% {17ms} DrGeo>>costume:
```

```
point: aPoint
	self preProcessing.
	^self finalizePoint: (DrGPointFreeItem newWith: {aPoint})
```

Figure 1.3: Dr. Geo profiling

Last refinement: executing code step by step. It is done through the debugger, in the contextual menu, choose the command *Debug it*. The debugger is invoked on the first line of the source code and it is executed step by step with the button *Over*. In the bottom area of the debugger window, at the right, the local variables. The other buttons allow other refinements in the step by step execution, it is up to you to explore it!



```
UndefinedObject>>DoIt
[anObject DoIt] in SmalltalkEditor>>debug:receiver:in:
[self value.Processor terminateActive] in BlockClosure>>newProcess
```

```
DoIt
	| figure fonction p integrale sommets |
	fonction := [:x | x * x].
	sommets := OrderedCollection new.
	figure := DrGeoCanvas new.
	p := figure point: -1 @ 0.
	p hide.
	sommets add: p.
```

```
self            thisContext        [:x | x * x]
all inst vars   stack top
                all temp vars
                figure
                fonction
                p
                integrale
```

Figure 1.4: The Dr. Geo debugger

1.1.2 Debugger

As shown in the previous section, the debugger lets you execute the code step by step. In the whole Dr. Geo environment you can invoke it any time with the keyboard shortcut

Alt-. .

Moreover, the debugger can be invoked in the source code by adding a line `self halt.` – like a breakpoint. In our previous example we modify the source code as follow:

```
...
p := sketch point: -1@0.
p hide.
summits add: p.

self halt.

-1 to: 1 by: 0.1 do: [:x |
   p := sketch point: x @ (function value: x).
   summits add: p hide].
...
```

1.1.3 Inspector

With the inspector dialogue, the user consults the attributes of an instance or a variable, it is updated automatically whenever the inspected object change.

In our previous example, suppose we want to see the content of the `summits` collection. In this case, it is very simple, we add a line of code where we send the message `inspect` to `summits`. The place in the code where we put it is not very important[3], it can be at the beginning or the end because we do not have breakpoint or step by step execution:

```
...
p := sketch point: -1@0.
p hide.
summits add: p.

summits inspect.

-1 to: 1 by: 0.1 do: [:x |
   p := sketch point: x @ (function value: x).
   summits add: p hide].
...
```

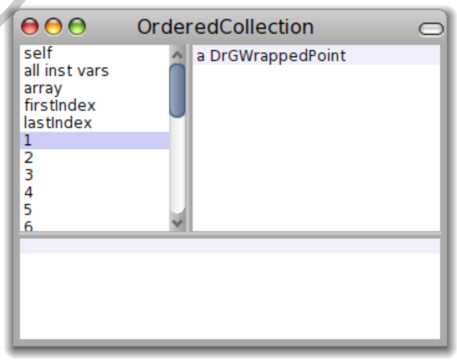


Figure 1.5: The inspector on the variable `summits`

[3]For obvious reason, we must avoid to add it in the loop.