Appendix:

How to program in Croquet

Author: S.M. Wolff
Project: Cell-Forum Project

Table of Contents

Introduction

This is a tutorial for programming in the Croquet environment. This tutorial will not go into details, but cover some basics which are needed to give beginning Croquet developers a kick start. First an “hello world!” application will be shown and described, then the topics ‘variables’, ‘graphical interface’, ‘control structures’, ‘data structures’ and ‘methods and classes’ will be covered. This tutorial will be concluded by an example of how to create your first working collaborative Croquet virtual world.

The tutorial was written for those that already have some experience in any other programming language(s).

Hello World

Like always with introductions to new languages, let us start with a simple ‘Hello World!’ program. Although we will be using the Croquet SDK, the first few tutorials will not be inside a working Croquet World, instead the output will be displayed within a ‘transcript’ window, comparable to a console window.

To start with Squeak, first we are going use the workspace to write some simple functions and let those functions print something in the transcript.

```
| a |

a := String new.
a := ‘Hello World!’.
Transcript show: a.
```

```
| a |
```

The first line of code in the example contains the declaration of variables. You do not specify the type of the variable yet here, only the name. In this case we have created a variable called ‘a’. The system doesn’t know if it is going to be an integer, double, string or any other object yet.

```
a := String new.
```

At the next line of code we declare that the variable with name ‘a’ is an object of the kind String. At this example I have explicitly declared ‘a’ as a String, however, for the basic variables this isn’t really necessary. In the next examples the type of the variable will be made clear at the moment of assigning a value to it. You see that the command is followed by a point, as in most other languages the semicolon ‘;’ is used, in Smalltalk the point is used for this. Also note that the symbol for assigning is ‘:=’ instead of ‘=’.

```
a := ‘Hello World!’.
```

Next we assign a value to the variable ‘a’. It now contains the string of characters: ‘Hello World!’.

Next we are going to print the contents of the variable ‘a’ to the screen, or better said, the Transcript window.

So now, when you have typed this in the workspace (or copy/pasted it there), select this piece of code by using your mouse and right-click somewhere in the Workspace to bring up a menu and select ‘Do it’ [Fig 1].

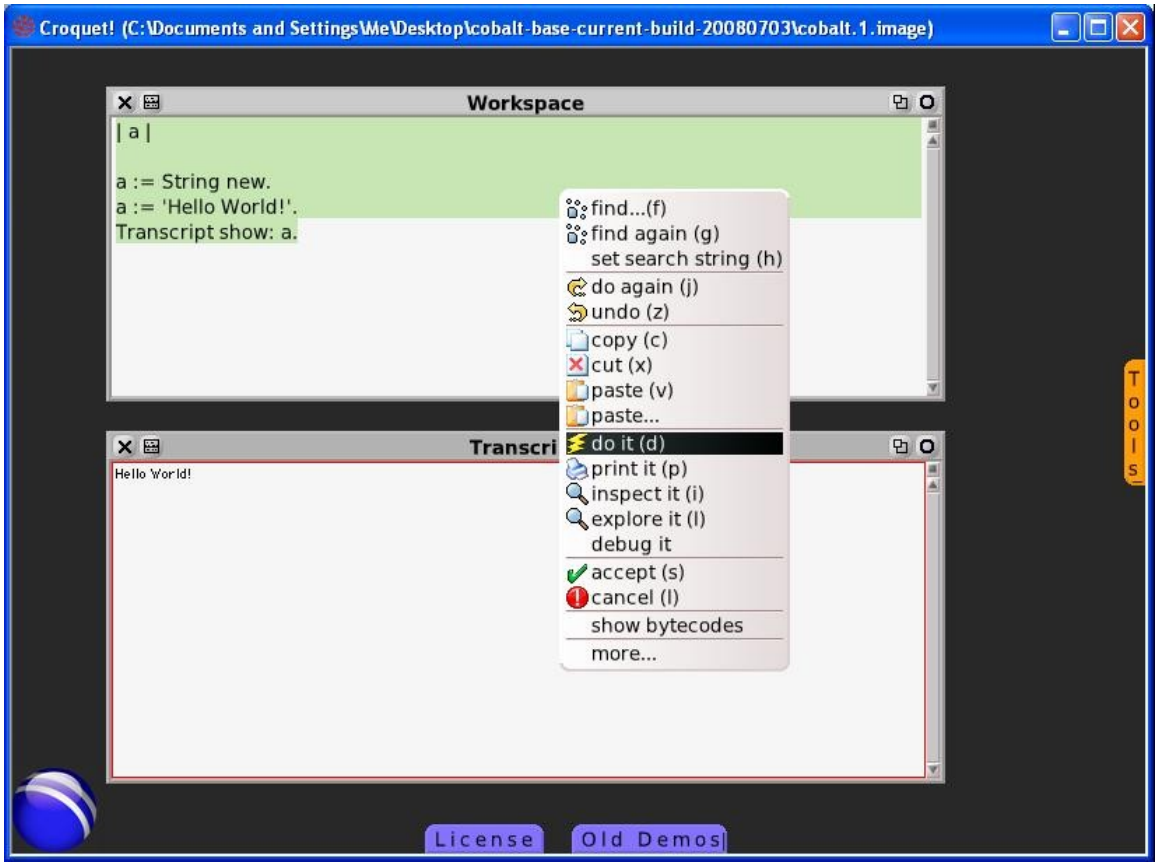

Fig 1. Execute a piece of code

This action will either execute, point out syntax errors or bring up a debug window of the selected piece of code. In this case the contents of the variable ‘a’ will be displayed in the Transcript.

The next section will cover the most frequently used graphical interfaces of the programming environment.

Graphical Interface

To start programming in Croquet, we will use Cobalt. Cobalt is a free and open source multi-platform metaverse browser and toolkit application built on top of the Croquet SDK Cobalt comes with a graphical programming environment which will be introduced in this chapter by using screenshots of the several most frequently used parts.

When you downloaded and extracted the archive containing Cobalt and executed the 'Croquet.bat' file, a start screen similar to the following screenshot will be visible.

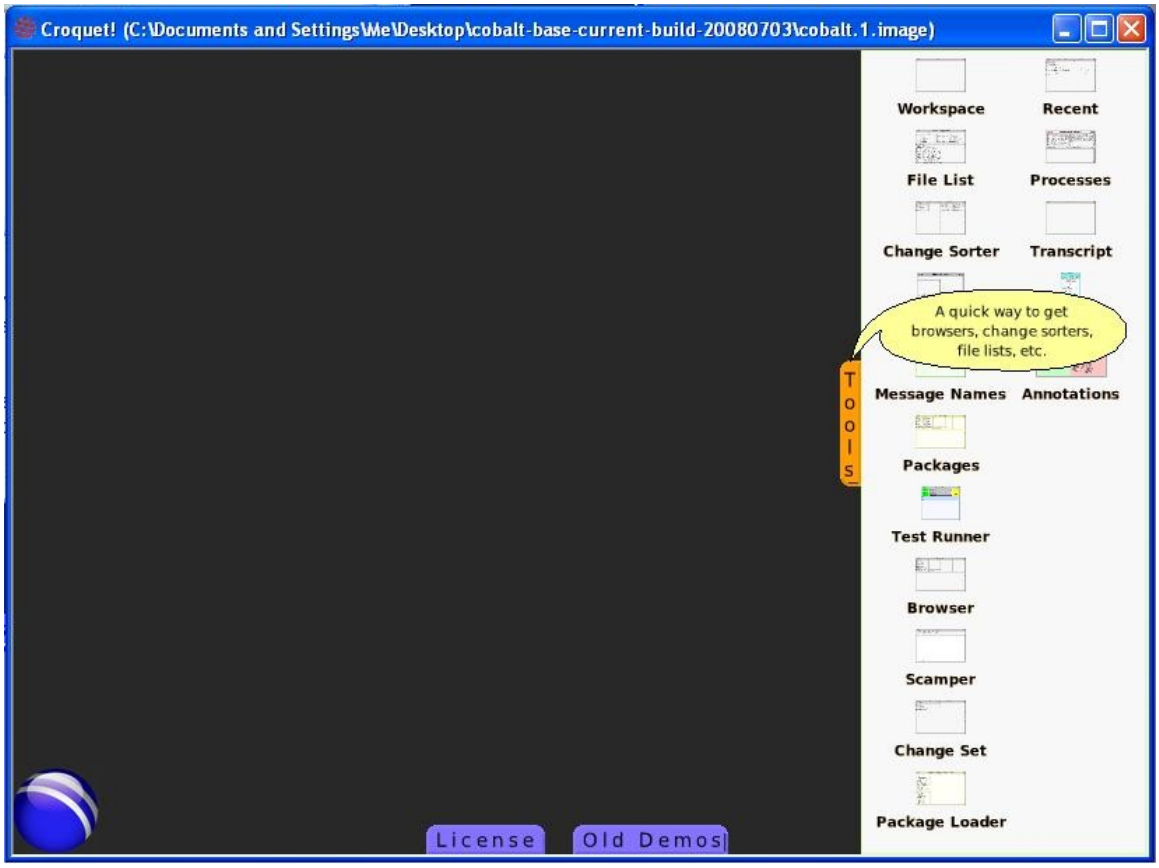

Fig 2. The 'tools' tab dragged out

The 'Tools tab' can be dragged from the right side of the screen to view the available tools to assist you in your Croquet programming sessions.

The most of the time you will be working inside the ‘Package Browser’, here you can browse all the available packages, class-categories within those packages, classes, method-categories and methods.

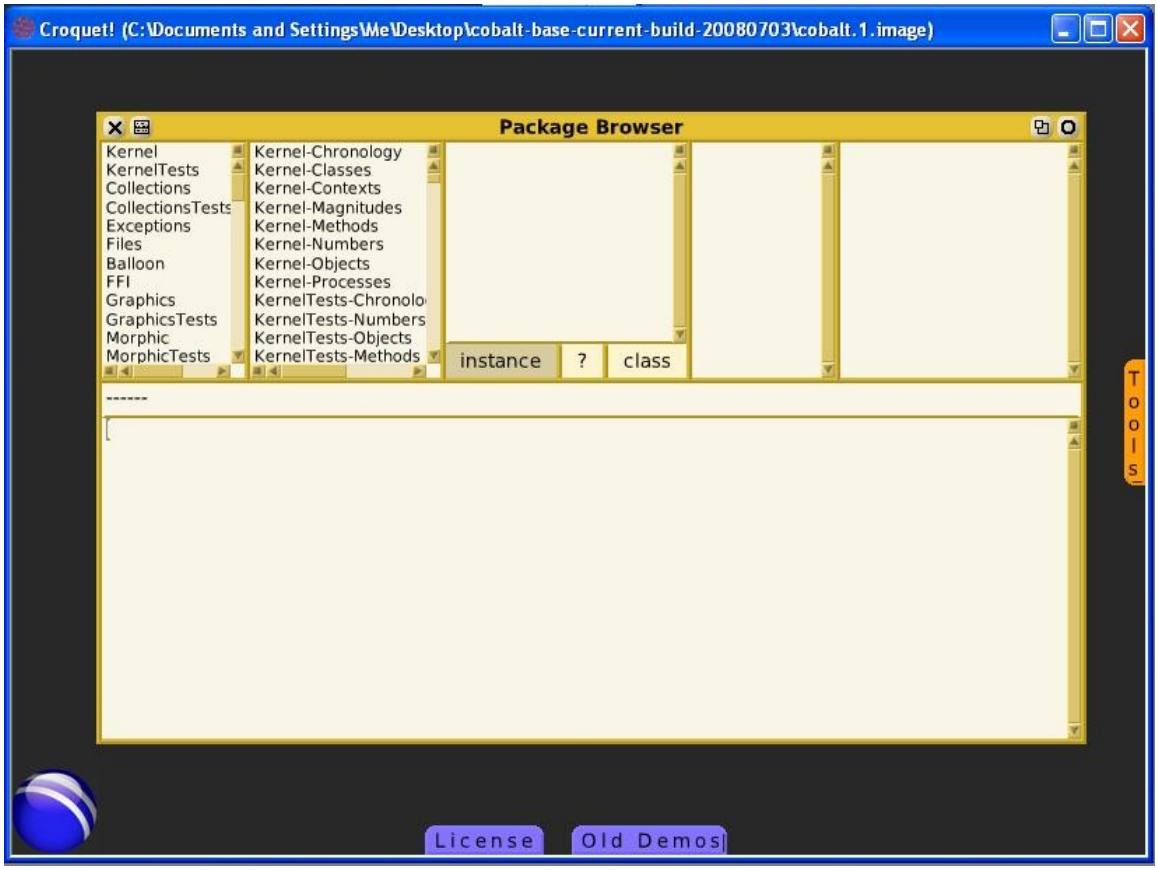

Fig 3. The package browser

Declarations and operators

Declaring variables, or object-references, instances, pointers, will be done at the top part of a Squeak scripts.

```
| var1  var2  var3  |
```

These variables will be local for the script that they're declared in. How to declare more global variables will be explained at the 'Class' section of this tutorial.

At the declaration of the variables, no types are specified yet. The most primitive variables will even never have to be specified, think of integer, string etc. Although in my opinion it will definitely improve the readability of most of your scripts.

The symbol ":=" is the operator for assignment in Squeak. All statements finish by a "point". So for example, to declare a string variable, and assign a value to it, the following lines of code will get you what you want.

```
| var1 |

var1 := String new.

var1 := 'This is the contents of the variable var1'.
```

Here an object of the class 'String' is defined. For other Classes it works exactly the same.

```
var1 := someClass new.
```

Control structures

The control structures known from every other programming language exist in squeak as well. The syntax however, is little different again. The syntaxes of the standard control structures (if-then-else, while, for-loop) will be explained in this section, if necessary by means of some example pseudo codes and their squeak representations.

The workings of the control statements will not be explained as it is considered you already know how they what they do, it are just the syntaxes that will be covered.

Testing variables

To be able to write conditional statements for the control structures, you first have to know how to test variables/objects for certain conditions. For comparison of variables or objects in Squeak, the following symbols are used:

>	greater than
<	less than
=	equal to in value
~=	not equal in value
>=	greater than or equal to
<=	less than or equal to
==	check if two objects are of the same type, rather than equal in value

And the signals for AND and OR are:

(a > 0) & (b < 0)	Returns true if *a* is positive and *b* is negative, otherwise false.
(a > 0) | (b < 0)	Returns true if either *a* is positive or *b* is negative.

If then else

The Squeak if-then-else structure syntax is the following.

```
| x y |

x := 1.
y := 2.

(x > y)
  ifTrue: [
    "do something".
  ]
  ifFalse: [
    "do something".
  ].
```

Switch

The switch control structure has exactly the same functionality as the if-then-else structure. Only the switch control structure offers significantly better readability compared to the if-then-else structure when a large number of conditions and associated actions are being listed.

```
aValue switch
  case: [matchCode1] then: [actionCode1];
  case: [matchCode2] then: [actionCode2];
  ...
  default: [otherCode].
```

While

The Squeak *while loop* control structure syntax is the following:

```
    x := 5.
    y := 0.

    [y <= x] whileTrue: [

    "do something".
      y := y + 1

    ].
```

For

The usual *for loop* is declared in Squeak like this:

```
1 to: last do: [
:i | i+1.

"do something"

 ]
```

This can be a little confusing for C(++) or Java programmers at first. Below is an example of how the *for loop* is used.

The *for loop* that you know would be:

```
for ( i = 1; I <= 4; i++ ) {
     x = i * 5;
}
```

In Squeak you will have:

```
1 to:4 do: [
   : i | i+1.
   x := i*5.
]
```

Data structures

With the term “Data Structures” we mean the different manners of organizing and accessing your data used for your computations. Here you can think of structures like Arrays, Lists and Heaps.

Ordered Collection

The ordered collection is best described as a composite of an array and a linked list. The data inside the ordered collection can be accessed by simply providing the location of the element inside the collection in which you are interested, much like you are used to with arrays. It is also possible to insert items in a certain position or just “at the end/beginning” of the collection. Below some of the basic actions on ordered collections are listed.

add: 'x'	To add the string x after the last element of the collection.
addFirst: 'x'	To add the string x at the first position of the collection.
size	Returns the quantity of elements of the collection.
asString	Convert an element of the collection to String.
at:n	Returns the string in the position n.
removeAt, RemoveFirst, RemoveLast	used to remove elements from the collection.

Classes, Methods and Shared Variables

Now the ‘how’ about programming in squeak has been covered, the ‘where’ about programming in squeak will be covered in this section. Squeak is a pure Object Oriented language and therefore everything you code will be in the form of Classes and Methods. This tutorial will not describe everything about Object Oriented programming here as there are numerous books about this subject. If you have programmed before you probably already have some experience with this in a language like C++ or Java.

To state it simply, a Class is a group of methods/functions and variables. A programmer can create, or instantiate, an object of such a class and execute the associated functions.

Everything inside of Croquet is a class and therefore it is very easy to use existing objects and adjust them to your own needs, this makes it possible to create functional collaborative worlds in a very shot period of time. One thing that you should keep in mind when programming in Croquet, is that you can have variables local to a method within a class, variables local to a object of a certain class and variables local to all instantiated objects of a class together [Fig 4, 5]. If you want a function to make use of parameters, you have to specify it in the function declaration, like in the following example code:

The declaration of the method, at the top of the method code, states the method ‘linkspaces’ with 4 parameters ‘a’, ‘b’, ‘ap’ and ‘bp’.

```
linkSpaces:a and: b p1: ap p2: bp

        | p1 p2 win1 win2 |

        "portal from a to b"

        p1 := TPortal new.
        p1 postcardLink: bp.

        …
```

This is how an instance of the class, called ‘cube’, is executing this method with the required parameters:

```
cube linkSpace:spaceA to: spaceB p2: postcardB.
```

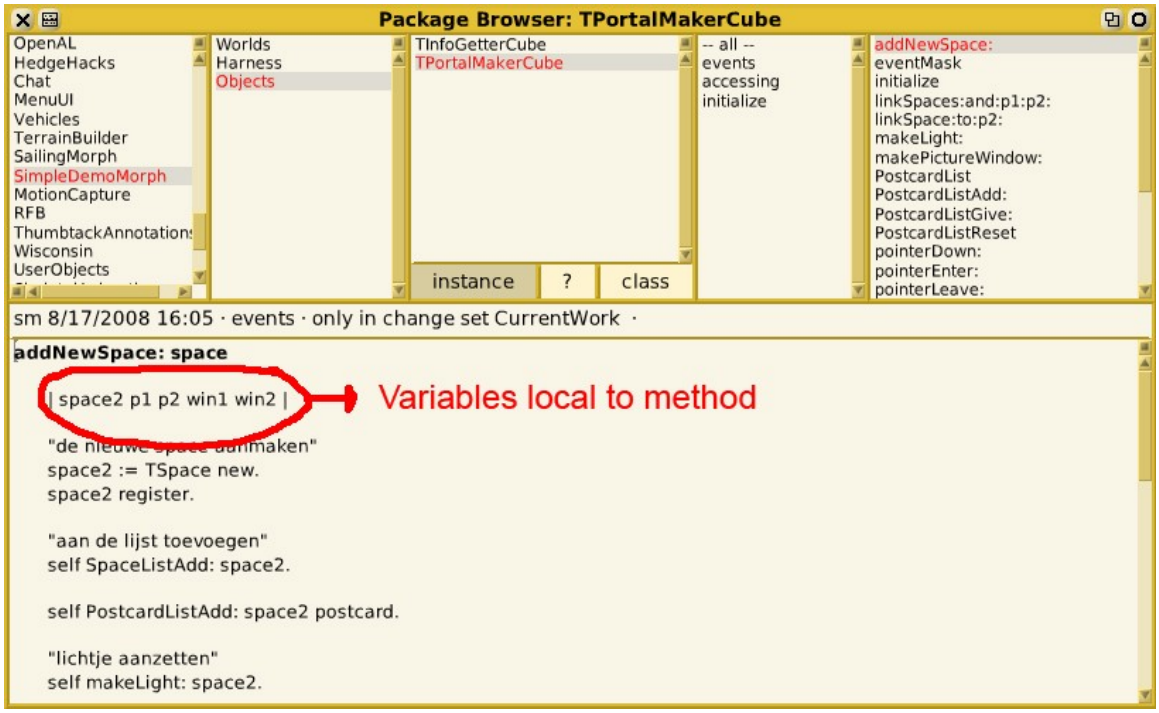

Fig 4. Variables local to method

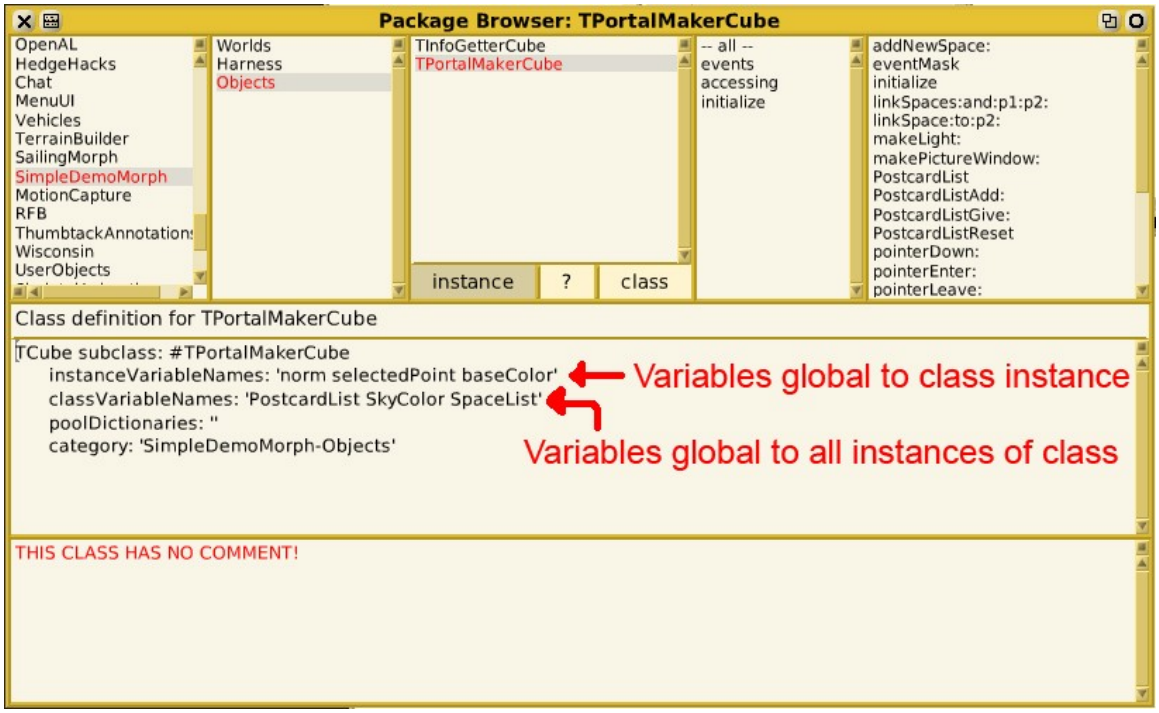

Fig 5. Variables global to single object / all objects of a certain class

Packages

Packages are a collection of classes that belong to a certain group/application/subject. However it is still possible to use a class somewhere which is not in the same package. It is just a handy method of grouping classes together.

Bringing it all together, a simple Croquet World

Let's start at the beginning, start up Cobalt and open a package browser window by clicking the 'Tools' tab from the starting screen and drag out a package browser to you workspace.

Now you are going to create a new package for your Croquet World. Left click your mouse on the square icon above the scrollbar and choose "add item", as shown on [Fig 6, 7, 8]

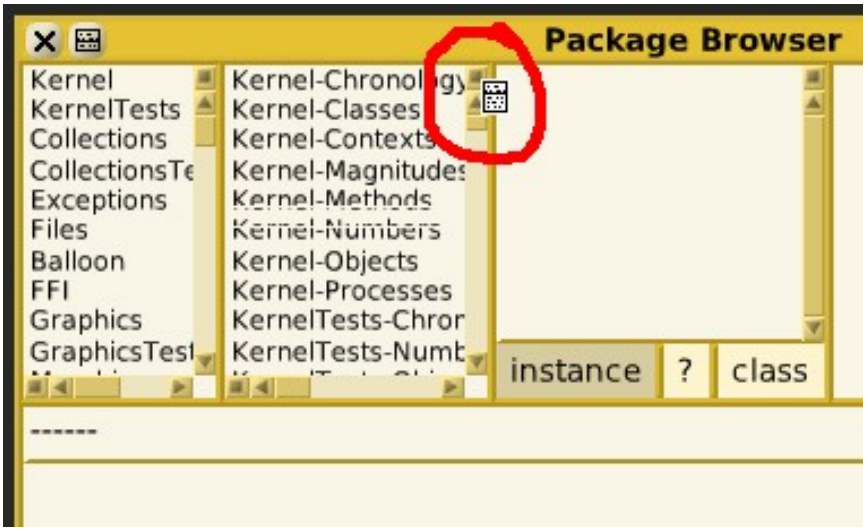

Fig 6. Click the square

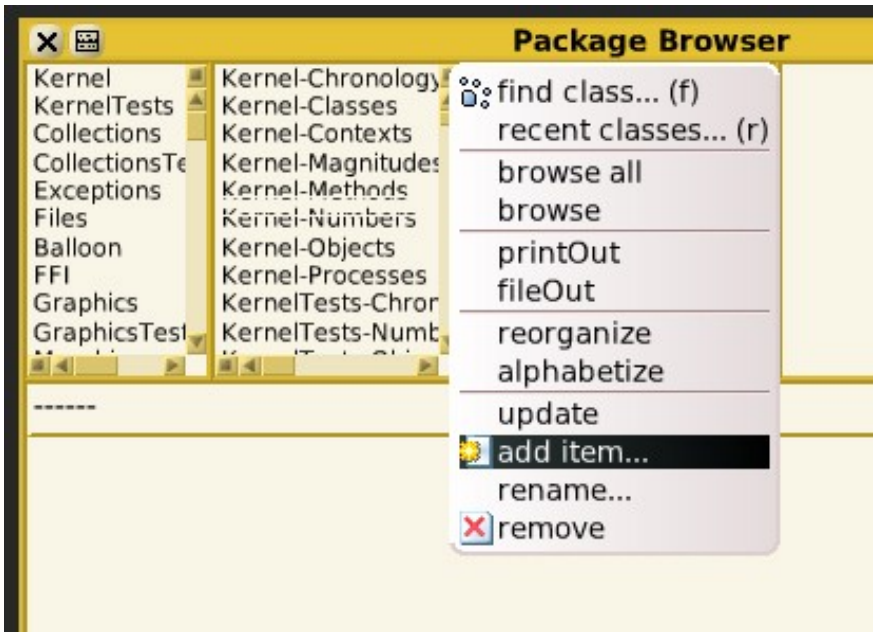

Fig 7. Choose "add item"

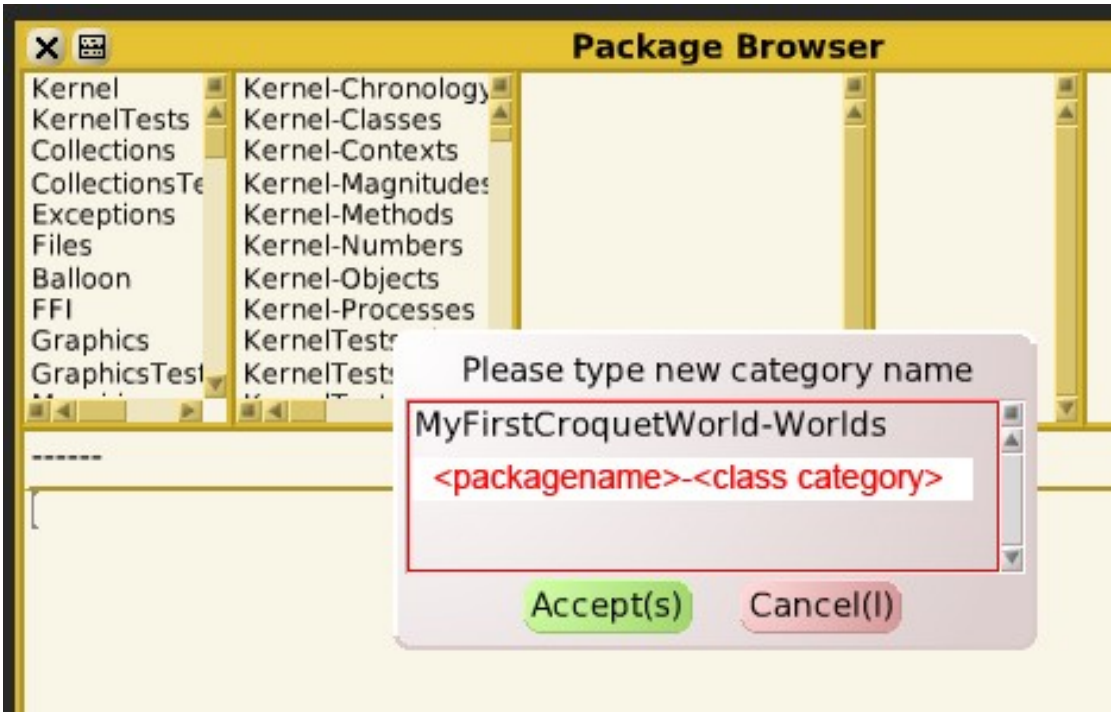

Fig 8. Type new package-category

The name on the left side of the dash (“-“) symbol is the package name and the name on the right side of the dash symbol is the class category name. Add a package containing the class categories “Worlds” and “Harness” [Fig 9].

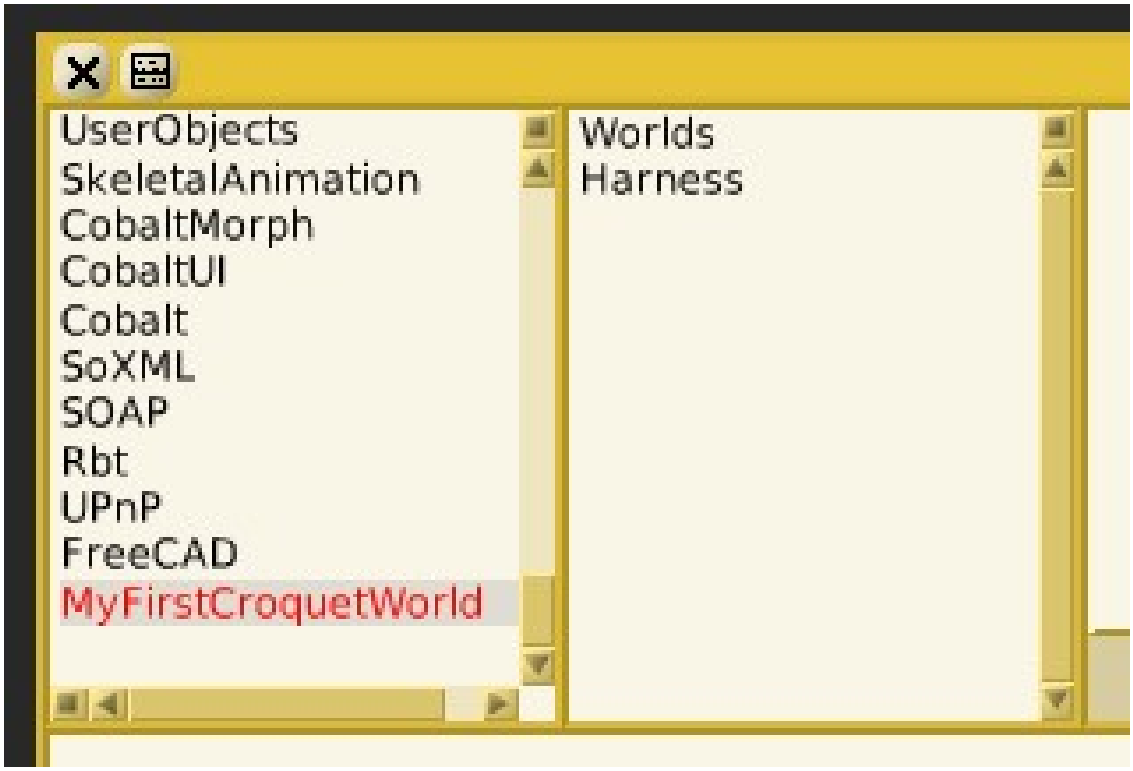

Fig 9. Resulting in your first own package

Now we are going to use one of the demo worlds to create a world of our own. Locate the package “SimpleDemoMorph” in the package browser. Click on it and select the class category “Worlds”. Right-click on the class “SimpleDemoWorld” and choose the option ‘copy’ [Fig 10].

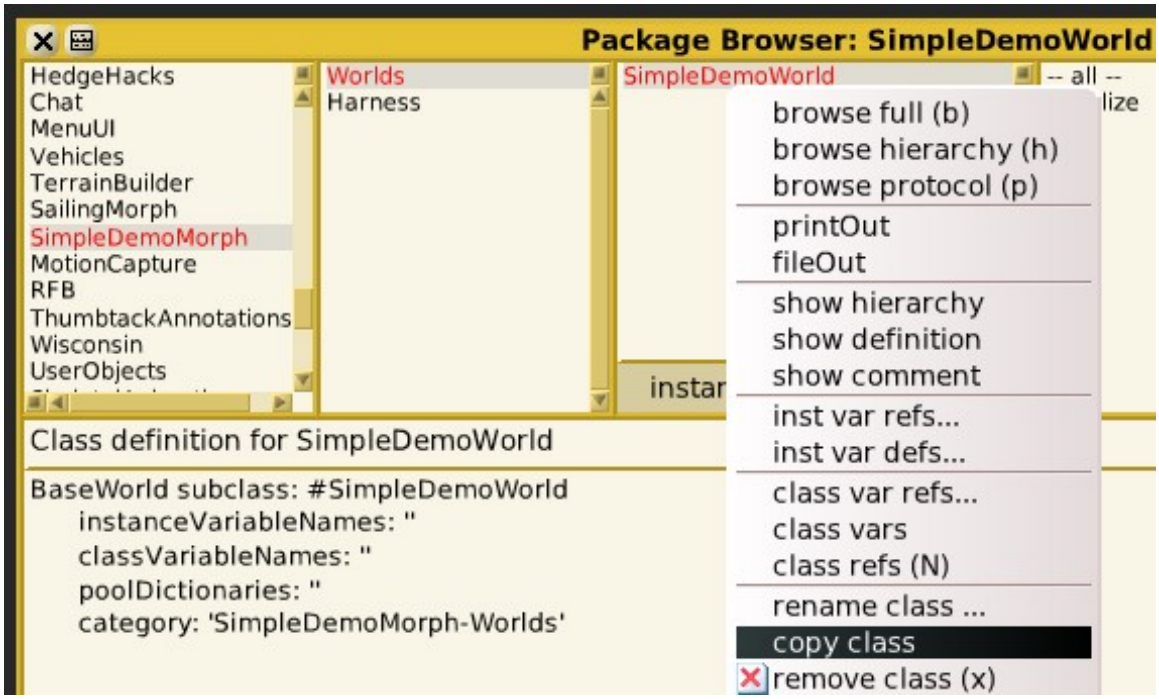

Fig 10. Choose option ‘copy class’

Copy the classes “SimpleDemoWorld” and “SimpleDemoHarness” of the “SimpleDemoMorph” package, give them a unique name and move them to your newly created package. How to move a class from one package to another is shown in [Fig 11, 12, 13].

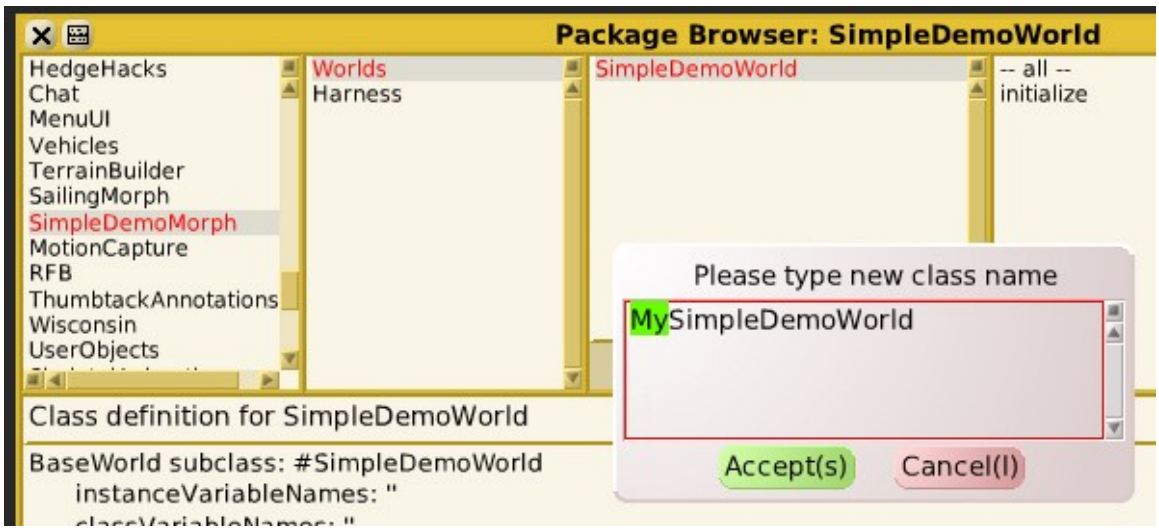

Fig 11. Give the copy of the class a unique name

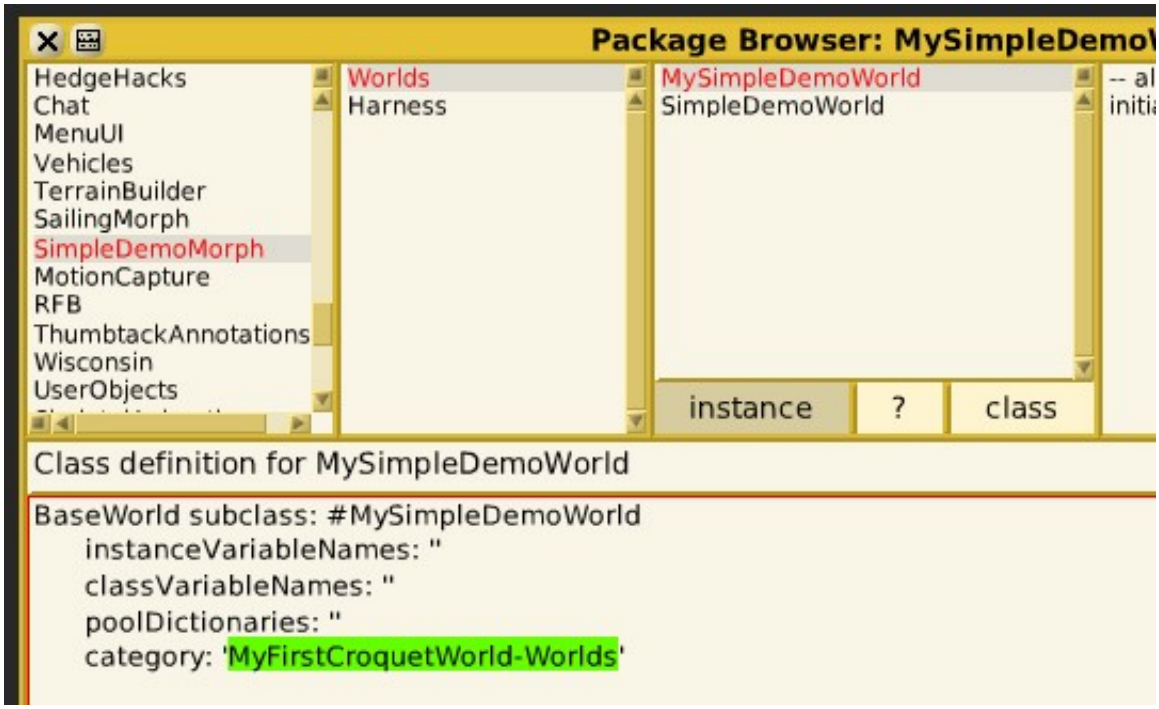

Fig 12. Change package location of the copied class

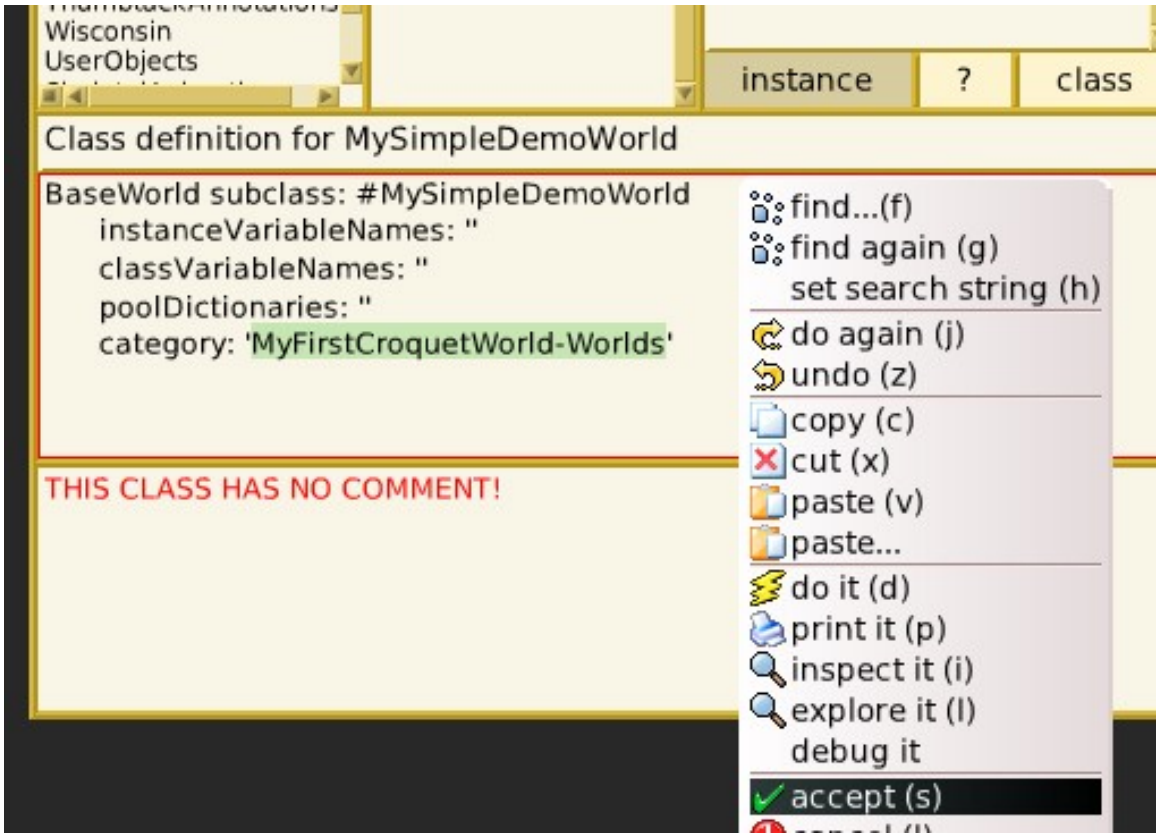

Fig 13. Don't forget to 'accept' the code, else nothing happens

Now you are going to set the Island ID for your Croquet World. Select the “MySimpleDemoWorld” class from the “Worlds” class category in your newly created package en press the “class” button [fig 14].

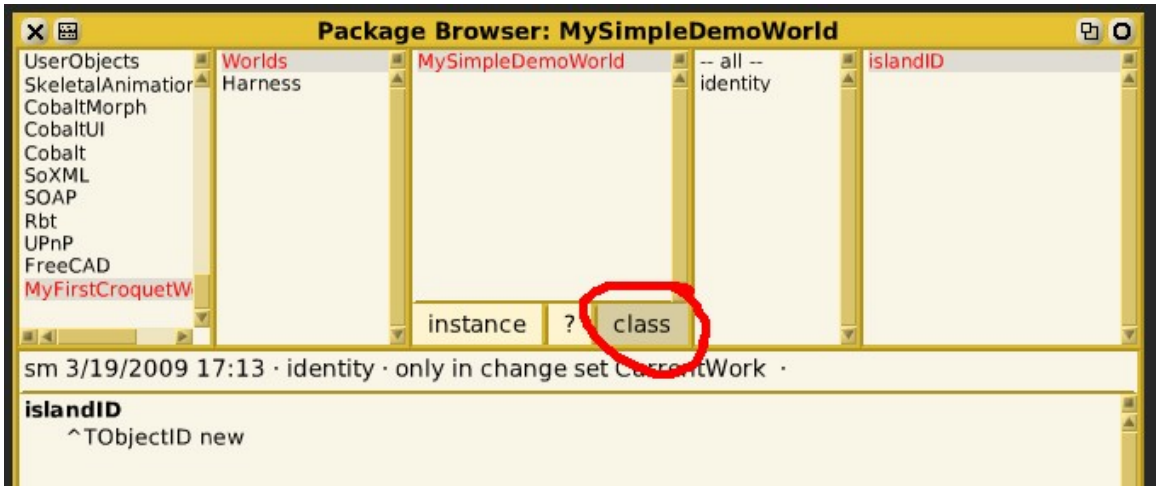

Fig 14. The ‘class’ button

Here you have to change to code from:

```
islandID
        ^TobjectID new
```

to:

```
IslandID
        ^TobjectID for: ‘MySimpleDemoWorld’
```

Where “MySimpleDemoWorld” is the same as the classname. Do not forget to right-click and ‘accept the code!

You are actually done now! But to make it easy to launch your Croquet World, you are going to make it appear in the “object” window. In the “MySimpleDemoMaster” class from the Harness class category, click on the descriptionForPartsBin’ method [fig 15].

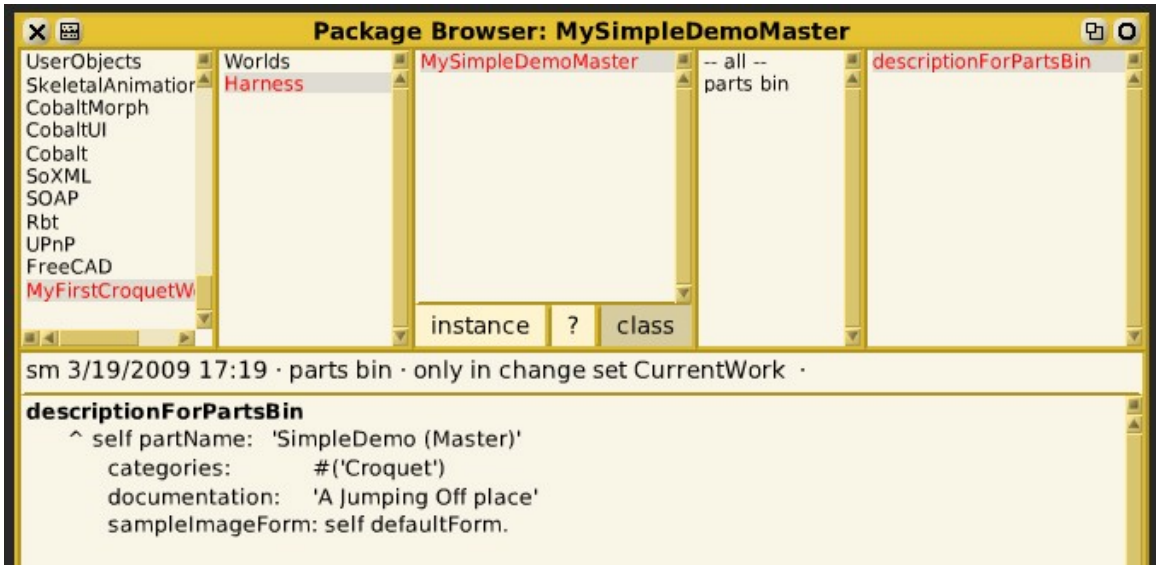

Fig 15. 'class' methods of 'MySimpleDemoMaster'

Now change to code to the following:

```
descriptionForPartsBin
        ^ self partName:      'My Croquet World (Master)'
        categories:           #('Croquet')
        documentation:        'My first Croquet World'
        sampleImageForm: self defaultForm.
```

Now select and 'accept' the code.

Now save your work by clicking on an open area on your workspace to bring up the menu. Click 'save as' and save your work as a new image. If you want other people to be able to run/join your Croquet World, you have to copy the [name].image and [name].changes files to computer where you want to run it.

Now let's open your World. Open up the Objects pane and drag your world to the workspace, your first Croquet world is born, congratulations!

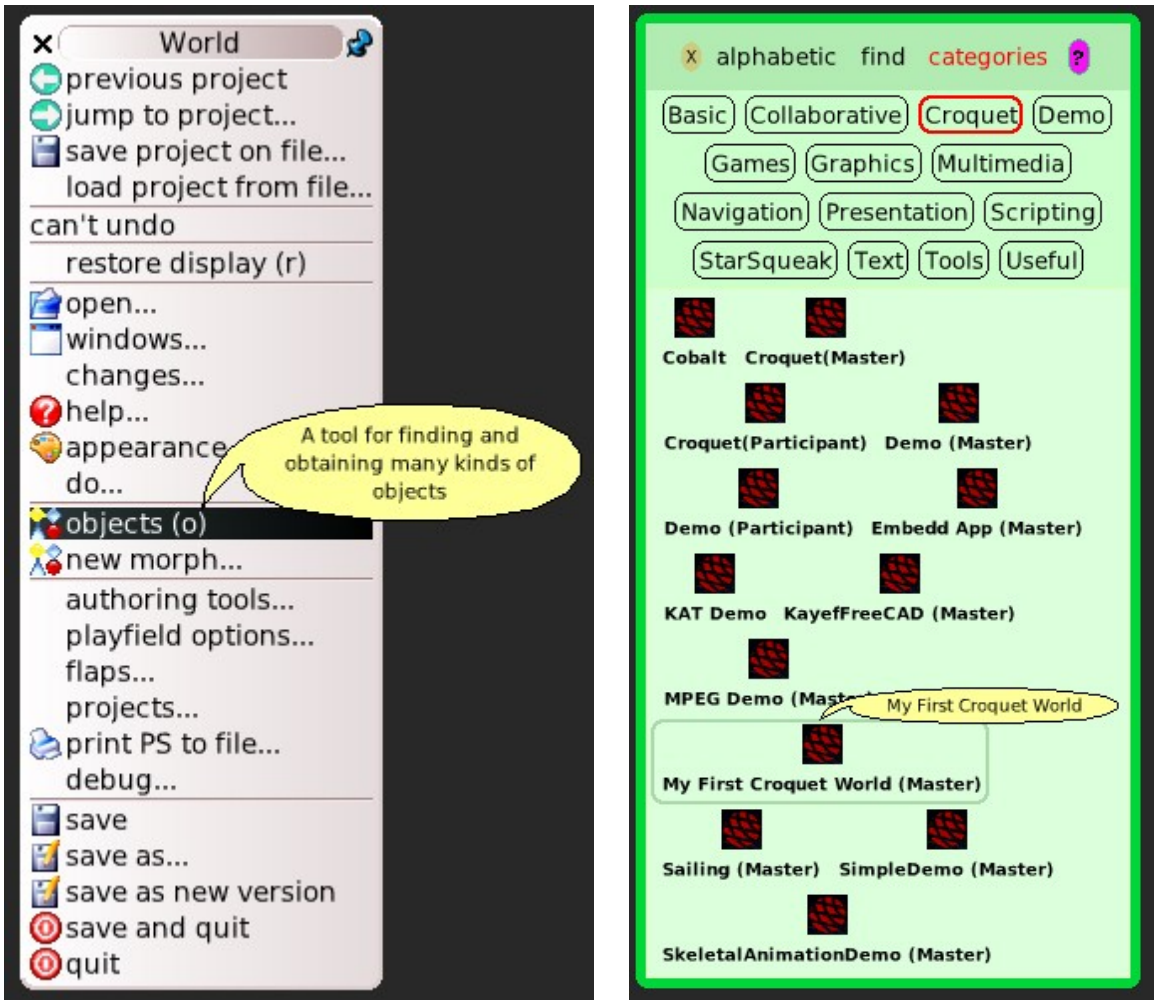

Fig 16. Opening the 'objects' panel and dragging your Croquet World to the workspace

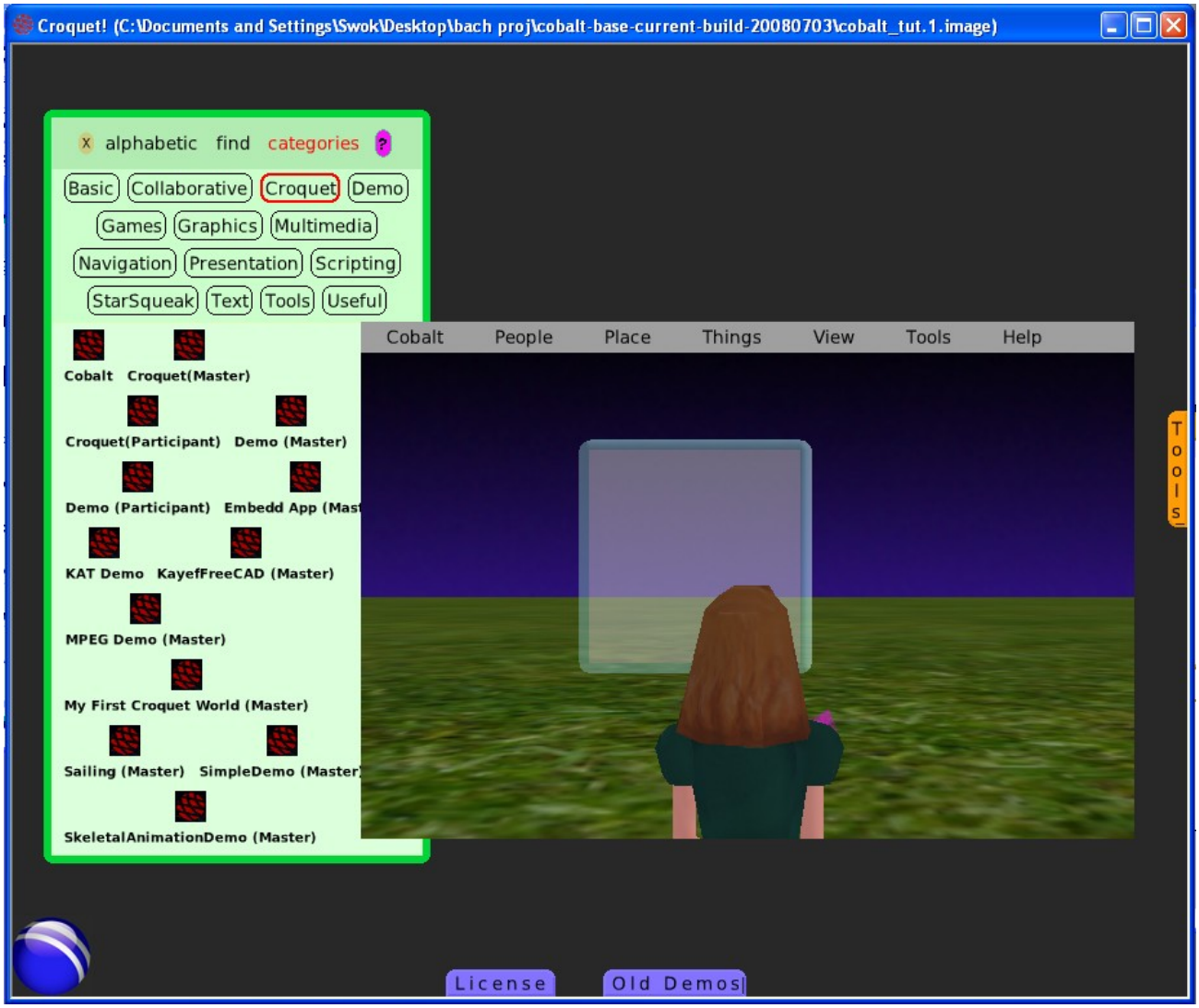

Fig 17. Your first Croquet World is born!