Resurrecting Dead Images

Pierre Misse-Chanabier

August 14, 2022

Abstract

Using images to develop software has proven very useful. However an image may break for multiple reasons and not being able to open anymore. For example when using a breakpoint in a method used by the system may prevent it to open. We explain here how we were able to recover such an image.

1 Introduction

Have you ever had a Pharo image that does not open anymore ? Hours or even days of work lost forever because you forgot to save your code outside of the image ? It happens to all Pharo developers at some point. And we all hate the existing solutions when a lot of code is involved.

Epicea. Epicea is pretty much the best option we have. Quite frankly, it saved me multiple times. It is awesome in a number of cases, because it allows us to reapply the latest changes on the current image. One image may be dead, but we are able to reconstruct it ! However Epicea is not as great when you have to apply back multiple sessions, ones that you forgot, and particularly when it touches sensitive code such as the Compiler.

2 Real World Case Description

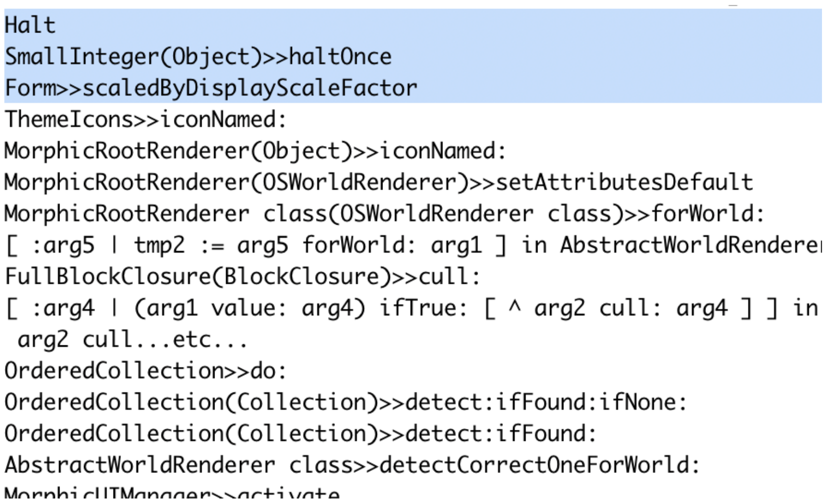
2.1 The Murder

In the RMOD team, we currently have a student that is working on having multiple OS windows, rather than only one (ref). While working on this project, he was modifying very sensitive user interface code. At some point, he added a breakpoint to explore a method, and saved his image to be sure he would be able to recover if something went wrong. He then closed the image. This image still had the breakpoint in the method.
Later, he tries to open the image.
Nothing.
The Pharo Launcher stays silent.

2.2 The Investigation

We followed advice from a colleague.

”If you can run images on your laptop but not from Pharo Launcher, I suggest you open Pharo Launcher, select an image you know it can be opened, right-click on it and click on ’copy launch command’. You will get the command PL use to run the image. Then you could try it directly from a shell and see what happens. It is often the best way to understand what is wrong when an image does not open.”
Following his advice, we found that the breakpoint is triggered by the startup of his image (Figure **??**).
Therefore, the image cannot open the UI.
Therefore, the image cannot open.



```
Halt
SmallInteger(Object)>>haltOnce
Form>>scaledByDisplayScaleFactor
ThemeIcons>>iconNamed:
MorphicRootRenderer(Object)>>iconNamed:
MorphicRootRenderer(OSWorldRenderer)>>setAttributesDefault
MorphicRootRenderer class(OSWorldRenderer class)>>forWorld:
[ :arg5 | tmp2 := arg5 forWorld: arg1 ] in AbstractWorldRendere
FullBlockClosure(BlockClosure)>>cull:
[ :arg4 | (arg1 value: arg4) ifTrue: [ ^ arg2 cull: arg4 ] ] in
 arg2 cull...etc...
OrderedCollection>>do:
OrderedCollection(Collection)>>detect:ifFound:ifNone:
OrderedCollection(Collection)>>detect:ifFound:
AbstractWorldRenderer class>>detectCorrectOneForWorld:
```

Figure 1: Debug stack printed by the VM when trying to open the image from a command line. This clearly shows that (1) the error preventing to open is indeed the halt and (2) the method containing the halt.

2.3 The Resurrection

To fix this, we used a recent tool we wrote based on the VM simulator: **Polyphemus** [1]. We opened the dead image inside a VM simulator, running inside a clean Pharo image. We then explored the classes to find the class with the breakpoint method (Figure **??**).

Once we had the method object (Figure **??**), we were able to compare the breakpoint version of this method, with one from a fresh Pharo 11, and analyse what we had to fix.

Of course, we first had to change the bytecodes. We take the bytecode from the Pharo version and inject it into the image. We replaced manually the content of this object with bytecodes that would execute the correct behavior. We replaced the last extra bytecodes with 0.

Listing 1: Using Polyphemus to replace the bytecodes. **replacementsBytecodes** are the ones extracted from the correct method. We replaced the last few bytecodes with 0

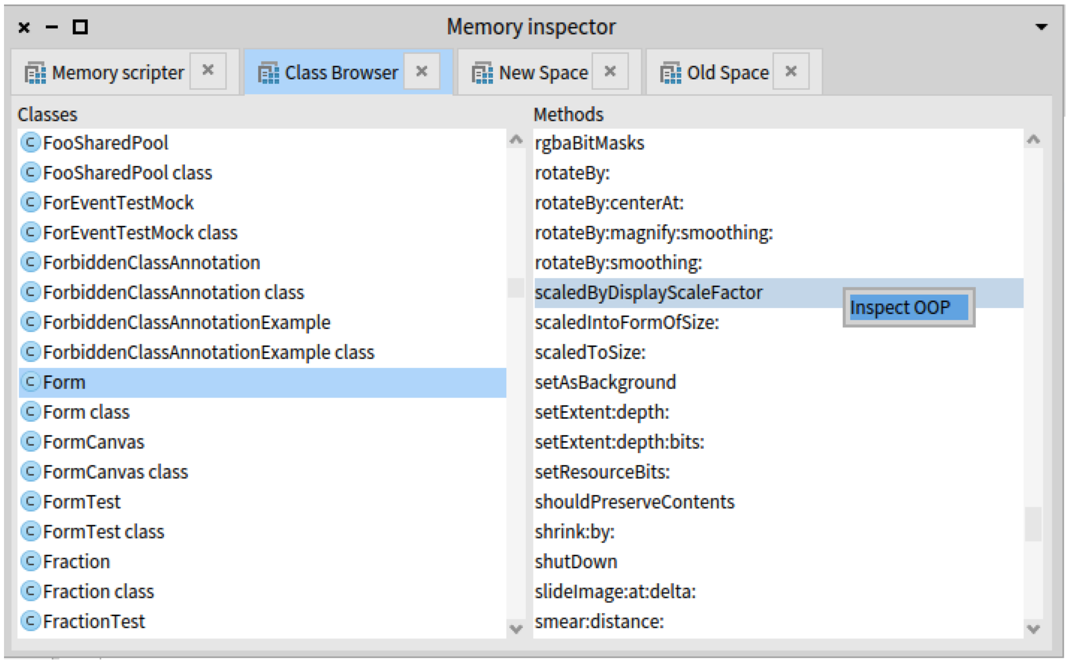
```
replacementBytecodes := #(76 76 128 76 129 130 104 147 92 16 152 150 252 0 0 0).
replacementBytecodes doWithIndex: [ :aBytecode :anIndex |
    self bytecodeAt: anIndex put: aBytecode
]
```

Listing 2: Using Polyphemus to replace the literals. We replace each bytecode with the next one.

```
0 to: self numberOfLiterals - 1
    do: [ :aLiteralIndex | | nextLiteral |
        nextLiteral := self literalAt: aLiteralIndex + 1.
        self literalAt: aLiteralIndex put: nextLiteral
    ]
```

This also required fixing the method literals, which are used to express the breakpoint. To send a message, the bytecode pushes the selector of that message on the execution stack. To have this selector available, it is stored inside the method literals. In this case, we can see that the *haltOnce* selector is stored at index one (ref). Therefore, the literals were off by one, and the indexes used by the bytecodes were wrong. To fix that, we replace each literal by the next one. This leaves a duplicated literal in the last index. This is not really a problem, because it is not used in the bytecode. This could be a problem because the last and penultimate literals have particular properties. They hold the class and the selector of the method. Therefore we advised the student to immediately recompile the method for safety !

[1] https://github.com/hogoww/Polyphemus



FooSharedPool
FooSharedPool class
ForEventTestMock
ForEventTestMock class
ForbiddenClassAnnotation
ForbiddenClassAnnotation class
ForbiddenClassAnnotationExample
ForbiddenClassAnnotationExample class
Form
Form class
FormCanvas
FormCanvas class
FormTest
FormTest class
Fraction
Fraction class
FractionTest

rgbaBitMasks
rotateBy:
rotateBy:centerAt:
rotateBy:magnify:smoothing:
rotateBy:smoothing:
scaledByDisplayScaleFactor
scaledIntoFormOfSize:
scaledToSize:
setAsBackground
setExtent:depth:
setExtent:depth:bits:
setResourceBits:
shouldPreserveContents
shrink:by:
shutDown
slideImage:at:delta:
smear:distance:

Figure 2: Class browser. Allows to browse the classes and methods of the dead image. This allows to bypass the metacircularity of the environment and to browse the methods without executing any of them.

Listing 3: Using the simulator to snapshot the memory onto the hard-drive.

```
reifiedMemory simulator coInterpreter primitiveSnapshot.
```

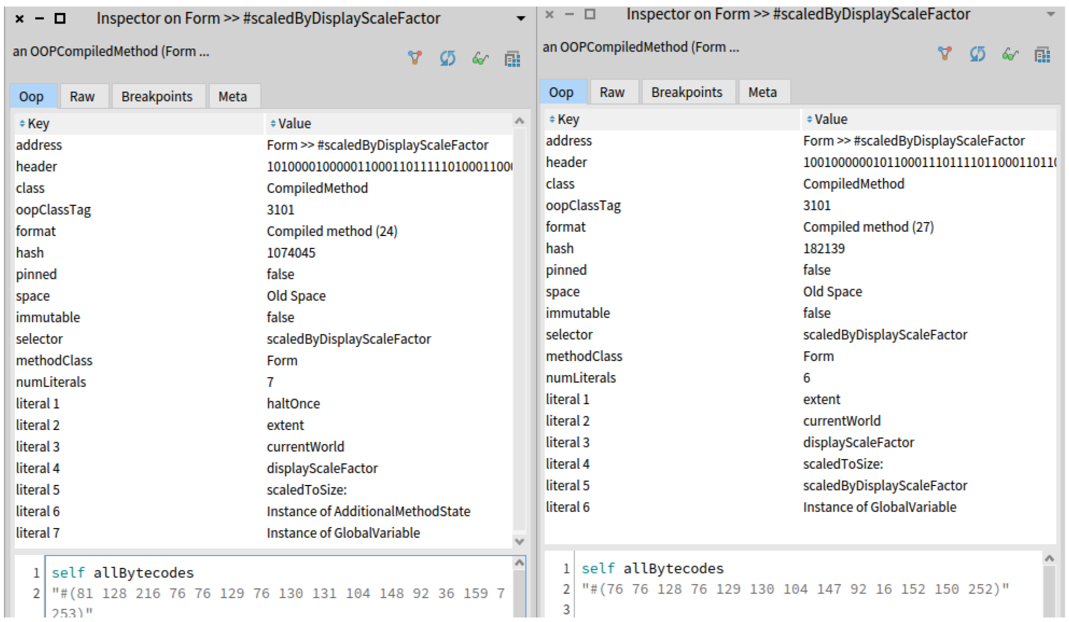
Finally, we snapshot this image ??. And poof, dead image is now resurrected !

With the abstractions provided by Polyphemus, it took us less than an hour to fix this image and recover days of work.

Note that this solution was possible because the method with the breakpoint has both more bytecodes and literals than the clean one. Therefore we were able to keep it simple and just patch it. More complex solutions would be required for more complex cases.

3 Closing Words

Although rudimentary, these solutions were achievable because of the initial effort of reifying the objects inside the VM simulator. We are also investigating how to investigate and repair images suffering from memory corruption. At this time, Polyphemus is just a base. A base that took a lot of thinking and effort to create, but a base nonetheless. The tools are yet to come !


Inspector on Form >> #scaledByDisplayScaleFactor
an OOPCompiledMethod (Form ...
Oop
Raw
Breakpoints
Meta

Key	Value
address	Form >> #scaledByDisplayScaleFactor
header	10100001000001100011011111010001100
class	CompiledMethod
oopClassTag	3101
format	Compiled method (24)
hash	1074045
pinned	false
space	Old Space
immutable	false
selector	scaledByDisplayScaleFactor
methodClass	Form
numLiterals	7
literal 1	haltOnce
literal 2	extent
literal 3	currentWorld
literal 4	displayScaleFactor
literal 5	scaledToSize:
literal 6	Instance of AdditionalMethodState
literal 7	Instance of GlobalVariable

```
self allBytecodes
"#(81 128 216 76 76 129 76 130 131 104 148 92 36 159 7
253)"
```

Inspector on Form >> #scaledByDisplayScaleFactor
an OOPCompiledMethod (Form ...
Oop
Raw
Breakpoints
Meta

Key	Value
address	Form >> #scaledByDisplayScaleFactor
header	1001000000101100011101111011000110111
class	CompiledMethod
oopClassTag	3101
format	Compiled method (27)
hash	182139
pinned	false
space	Old Space
immutable	false
selector	scaledByDisplayScaleFactor
methodClass	Form
numLiterals	6
literal 1	extent
literal 2	currentWorld
literal 3	displayScaleFactor
literal 4	scaledToSize:
literal 5	scaledByDisplayScaleFactor
literal 6	Instance of GlobalVariable

```
self allBytecodes
"#(76 76 128 76 129 130 104 147 92 16 152 150 252)"
```

Figure 3: Inspection of the breakpoint compiled method object on the left and the clean compiled method object on the right. The breakpoint method has 3 more bytecodes sending the haltOnce message. It also has an extra literal for the selector *#haltOnce* in the first literal.