Summary	Neocortex version 1.4 Developer's Guide	
Document Control	The document is identified by the version and date/time of saving.	
Status	Draft	
Author	David Green	Version Date 14/12/2008
Reviewed by		Date
File Version	NeoV1.4_Developers Guide.Version 0_1.doc	
Main Changes at this version	Changed images and nomenclature for version 1.4.	
Change forecast	More sequence diagrams to be added as required. E.g. for general framework and for recognition.	

Table of Contents

Table of Figures

1 General Document Information

1.1 References

1	C++ GUI Programming with Qt 4, Jasmin Blanchette and Mark Summerfield, Prentice Hall 2006.
2	Visual Assist version 10, Whole Tomato (software)
3	Enterprise Architect version 7, Sparx (Software)
4	NeoV1.4 User Guide 1_0

1.2 Abbreviations and Glossary

EA	Enterprise Architect (Sparx Systems' UML CASE tool)

1.3 Document Cross References

Ref	Name Version and Date	Title

1.4 Subject and Audience

This document relates to the Source Forge Neocortex project
http://sourceforge.net/projects/neocortex/
This document is intended for individuals who are knowledgeable in software development.
Basic knowledge of C++, software design techniques and the Qt C++ GUI is assumed.

The following web sites are of value to qt developers
http://trolltech.com/
http://www.qtforum.org/
http://www.qtcentre.org/
http://qtnode.net/wiki?title=Main_Page
For running qmake
http://web.mit.edu/qt-dynamic_v4.2.1/www/qmake-running.html

A viewer for Enterprise Architect is available from
http://www.sparxsystems.com.au/products/ea_downloads.html

1.5 Acknowledgements

The core model, along with the original interface using Borland Studio, was released by Saulius Garalevicius in 2005.
David Green contributed a new partial framework of Neocortex, a pixel editor based on [1] and the multi-platform user interface. This interface uses the open source (GPL) version of the cross platform toolkit Qt (now owned by Nokia).

The Neocortex development team thanks Dileep George for publishing the Pictionary project (available at http://www.stanford.edu/~dil/invariance/), which inspired several important ideas used in the Neocortex.

2 Neocortex Project Structure

2.1 Source Formatting

2.1.1 File Names

File names are regarded as case sensitive. The reason is that whereas for Windows the case is ignored, for Linux and other known UNIX variants the case is relevant and files will not be found if this rule is not observed.

2.1.2 Braces

Source formatting within MPF uses the convention that the opening curly braces go at the end of the leading statement. E.g. `if (a==b){`
This is in line with the style of the original MPF code.
In the case of new code connected with the Gui the alternative convention is in use.
The reason for this is simply that I (DG) write in this style and there is no clear winner for 'correctness' so I thought I might as well leave it as it is. That way we may please some of the people all of the time.
It is not very hard to convert between the conventions. I tend to convert MPF to my own style when I'm working on it then back to the original style for consistency within that project directory.

2.1.3 Variable naming

I have found it useful to have a convention for naming variables.
Class members are prefixed 'c' e.g. `cIImageSourceContoller`
Local variable are prefixed 'l'
Parameters are prefixed 'p'.
There are quite a few places where this could be tidied up and I have not changed much in MPF.

2.2 Source tree

The source tree has five branches for the main program.
A separate iconeditorplugin program, on its own branch, adds a library to the Designer plugin directory so that iconeditor is available at design time.
The necessary part of the CImg template library is also present as a branch in the directory, CImg-1.2.5, but does not have a project of its own because it is only a header file of templates.

2.3 Visual Studio projects and QMake projects

The 'Precompile' branch holds the QT Designer files for the GUI. The reason for the file being in a separate directory is that the QT uic processor creates, from the `.ui` file, a header that is needed in the other projects. If the header is generated in any one project then, in some circumstances, circular dependencies are created between compilation units.

2.3.1 Visual Studio 2005 projects

The project and solution files are provided in the source tree so everything should build with Visual Studio 2005. Make sure Neo is set to the startup project.

When you build the **IconEditorPlugin** project, the plugin will be placed into `$(QTDIR)\plugins\designer\`.
This plugin enables the Icon Editor program to be available in the Qt Designer Widget Box from where it can be dragged onto a form.
In order for the plugin to appear in the Qt Designer Widget box it must be a Release build. (See the release note for the current status of the Designer widget).

2.3.2 QMake projects

For Linux and for MinGw on Windows, six QMake projects are provided. Five are for Neo; the remaining one is for the iconeditorplugin project.

Please see the release note for the current method of building under Linux..

The iconeditorplugin project enables the iconeditor to be seen in Designer.
The main .pro file (For Neo) is in the root.

The QMake files have been checked with MinGW compiler (Release build only) and the g++ compiler (GNU) compiler under SUSE 10.x.

2.3.3 Creating new QMake projects

If you want to regenerate the project files you must ensure that the iconeditorplugin directory is not present. Otherwise qmake may not work properly.
* In most of the files you will need LIBS += -lgdi32
* It is best to use the existing QMake files as a model.

2.4 The CImg template library

Just the main template file CImg.h is included in the source tree.
Because CImg.h is a template file, no specific compilation is needed – including the header in the program is sufficient.

Some elements of the CImg library need to be separately compiled but these are not currently used by Neocortex.

The CImg library is available in full from:
http://cimg.sourceforge.net/

The CImg examples, documentation, plugins and compilation are not present in the Neocortex source tree. If you have downloaded the entire CImg archive the examples will appear in your .Pro file (should you create one) and can be removed (unless you do want to compile the examples.)

Any Windows program that uses the CImg templates needs to be linked to the library gdi32. With Suse Linux, at least, no specific steps are necessary.

The main reason for using this library is that the image rotation function in qt proved very hard to manage. Rotating a qt image gave inconsistent results with the image being translated as well as rotated. Attempts to correct the translation effect failed. The CImg rotate function does just what it says on the packet. So this was used instead of the qt function.
The reason for failure when using qt may be that I was attempting to use the qt framework for image processing in a rather naïve way. Qt image processing seems to assume you will change the frame coordinates rather than moving image data around. That would be a more efficient way of doing it and could be explored later especially as it could provide a noticeable speedup when performing translations for saccades.

The secondary reason for using CImg is that the library is specifically targeted at image processing. As such the library should be explored for functions useful to Neocortex.

3 Packages

An attempt has been made to separate the Graphical and other interfaces from the core code.
This is an on-going effort and the future gains will have to be weighed against the cost, in time an effort, of further separation.

Roughly speaking each directory in the source tree can be seen as a package although these 'packages' are not separately useable at the moment.
The main packages are described in the following sections.

Please note:

- The term Event corresponds to a Qt signal.
- Most of the components correspond to individual classes. But in a true component model there is no necessary correlation with classes. Components can have an internal structure.
- A Component should be characterised as an autonomous replaceable part of the system. So the use of the term Component is a statement of intent – most 'components' here are not easily replaceable. Amongst the exceptions are the Saccade classes and controllers in the Framework package.
- The use of Qt signals and slots gives us a convenient way to define the interfaces and hide the inner workings of components. Ref [1] is strongly recommended for anyone who intends to carry out Qt development using signals and slots.
- The term package refers to a branch of the source tree. It is just a useful grouping of classes.

3.1 Neo/Gui

This package contains only the MainWindow code. It is linked to GuiUtils, in the Framework package, as shown in Figure 1. It is also linked to Thinker.

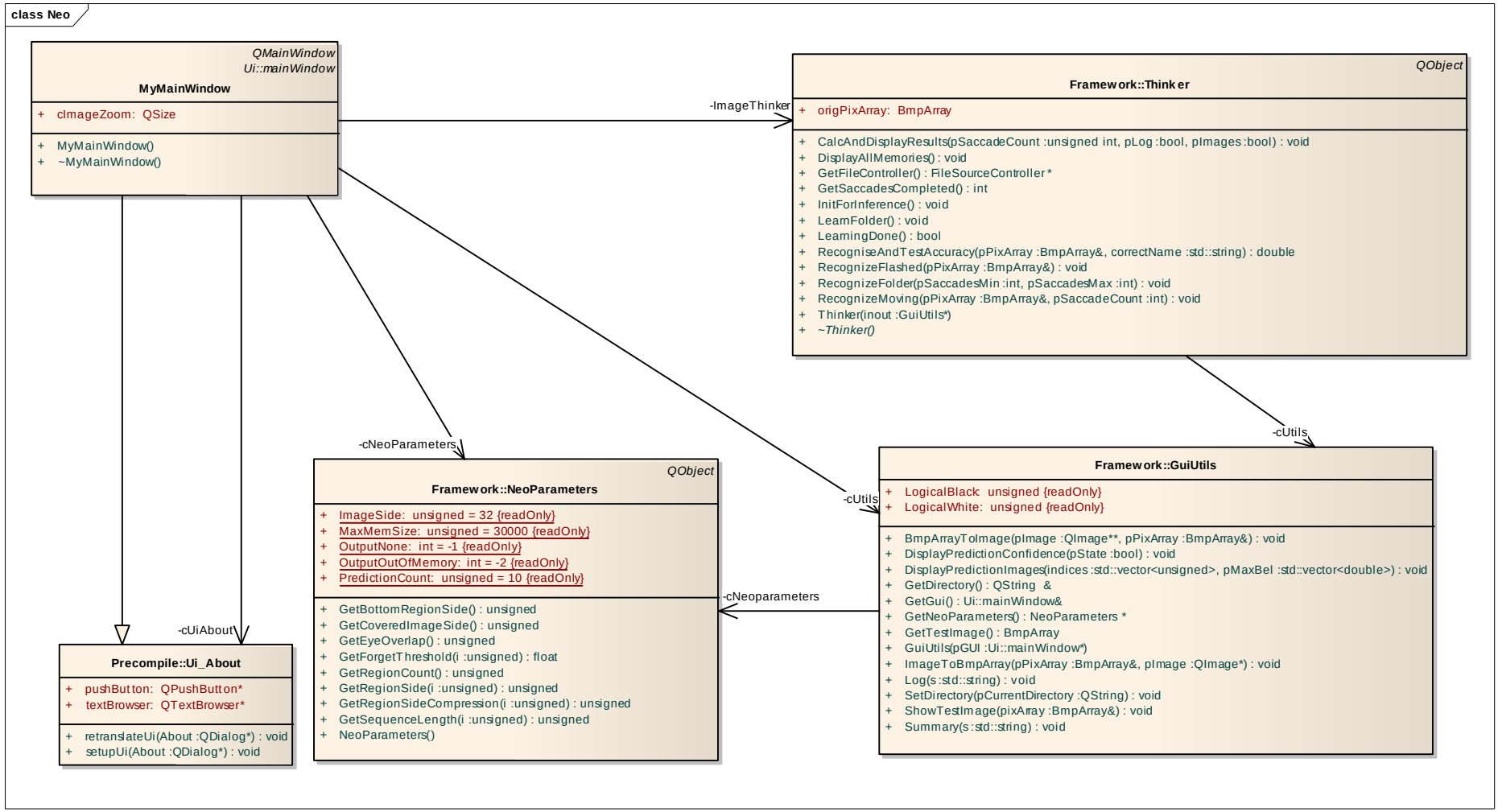

Figure 1 Gui Class Relationships in context.

MyMainWindow is the point at which the GUI controls are made visible.
The controls in the GUI trigger Qt Slots which appear as private members of `MyMainWindow`.
The slots are:

```
void OpenImage();
void LearnFolder();
void RecogniseFlashed();
void RecogniseMoving();
void RecogniseFolder();
void ExitApp();
void closeEvent(QCloseEvent *pEvent);
void DisplayPredictionConfidence(bool pState);
void Interrupt();
```

Here is the connect instruction to the Qt preprocessor:

```
QObject::connect(ui.NormaliseResults,
SIGNAL(toggled(bool)),  this,
SLOT(DisplayPredictionConfidence(bool)));
```

Thus the signal triggers the required member function to carry out processing
For further details of signals and slots see [1] early chapters.
`ReadSettings()` and `writeSettings()` retrieve and store the position of the main window on the screen when the program is started and ended. These functions are overrides to `QSettings`. The functions are available on all supported platforms so they can operate with the Windows registry and with other means, usually files, in the non-Windows environment.

3.2 Framework

The following class diagram shows the Framework classes in the context of Thinker and the 'Gui interface' GuiUtils.

Thinker makes use of the source controllers to obtain files and images for processing. ImageSourceController must use ISaccade – derived classes to create a stream of callbacks that it can relay back to tThinker

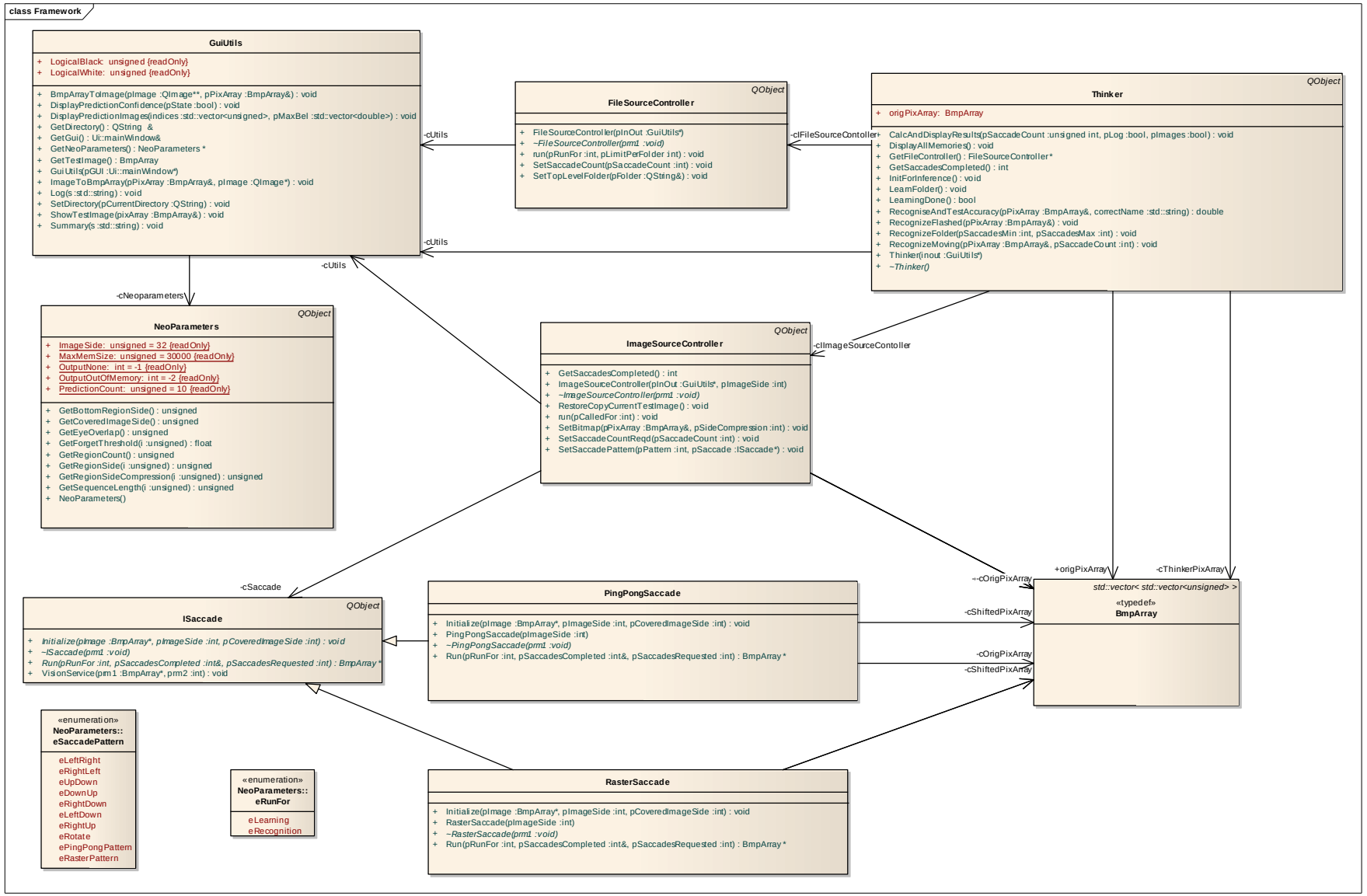

Figure 2 Framework Classes in context

3.3 IconEditor

This package contains the editor that allows our pixel array example images to be displayed and edited within the GUI. All the image manipulation code is held within iconeditor and it includes the CImg library mentioned earlier.

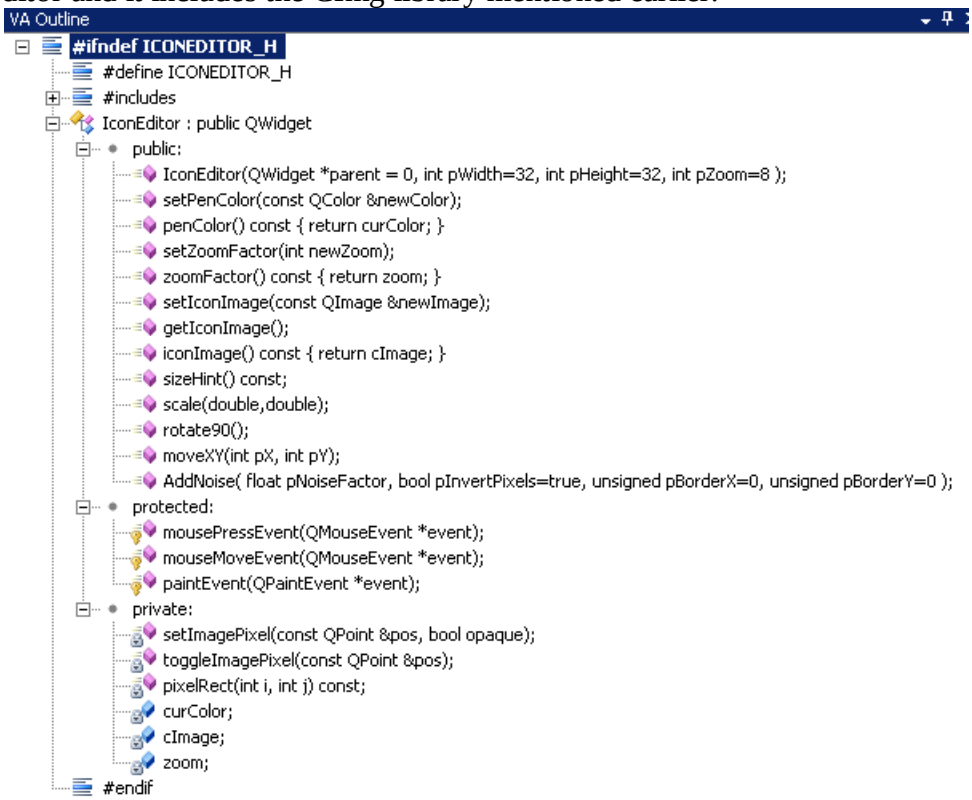

Figure 3 The Iconeditor Class

The Qt events from the application-related Gui controls are handled by the `moveXY(int, int)` etc. members which are called from the ImageSourceController
IconEditor also handles its own Gui events which are:
`mousePressEvent(…)`
`mouseMoveEvent(…)`
`paintEvent(…)`

3.4 MPF

This is the main core of learning and recognition.
It is here where the algorithms are run.

3.5 Precompile

This directory contains the GUI sources, `Neo.ui, Parameters.ui` and `About.ui` which are created by the Qt Designer utility.
The Qt pre-compiler uic.exe converts, for example, `Neo.ui` to `ui_MainWindow.h` which is required by many other parts of Neocortex.
This is an area where further re-factoring could be attempted. The links to the ui from diverse parts of the program are undesirable dependencies to the Gui.

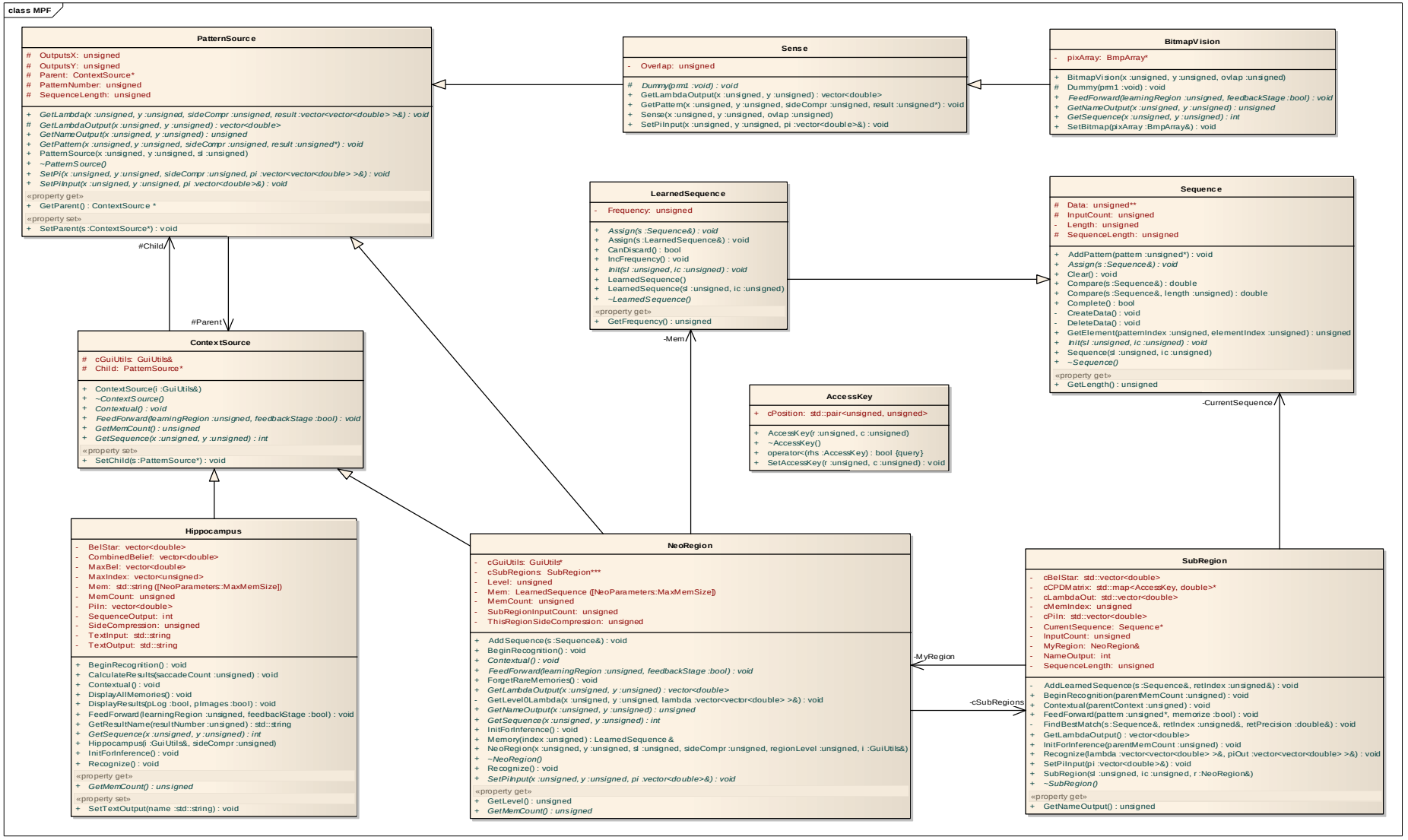

Figure 4 MPF Core classes

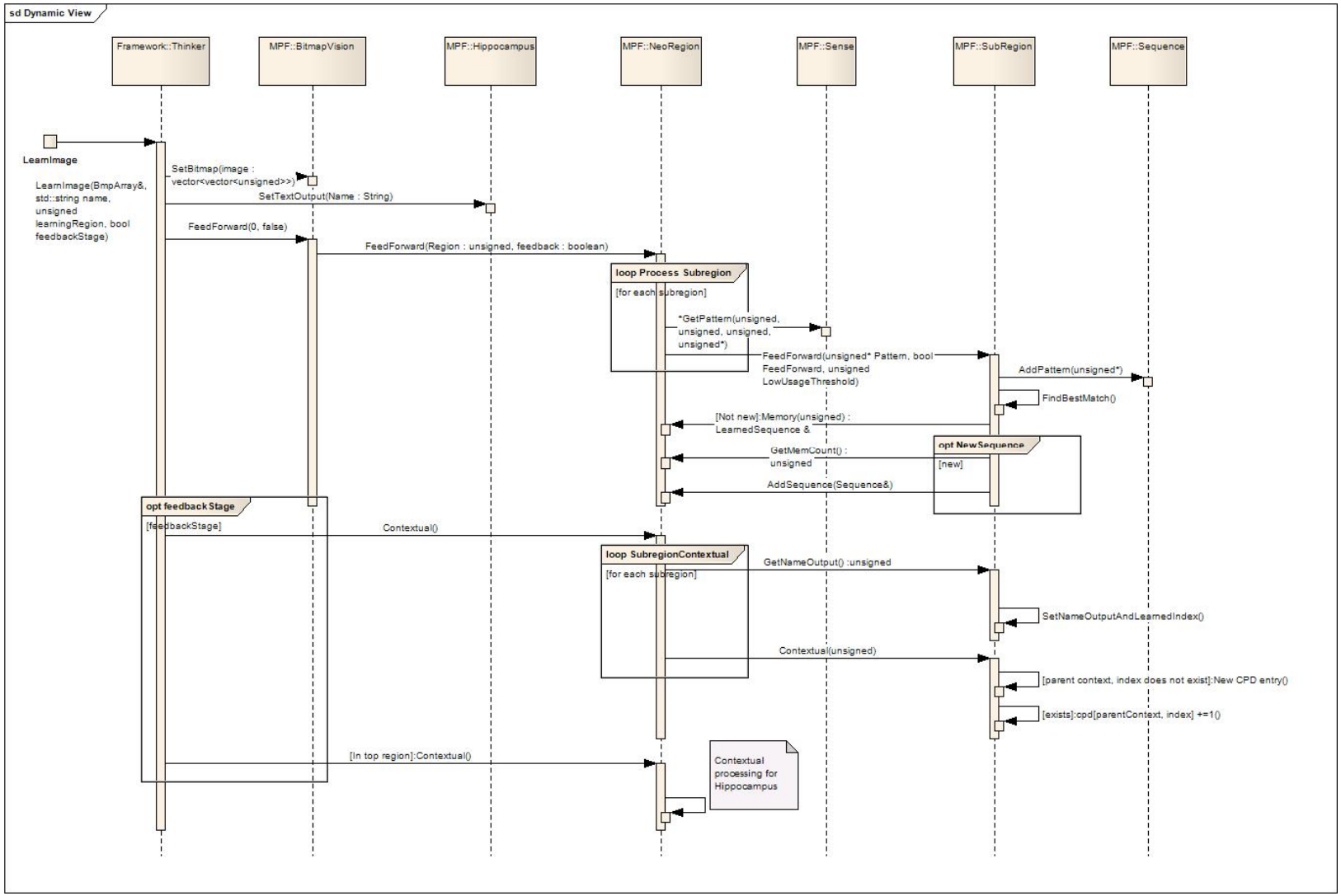

Figure 5 MPF Sequence Diagram for LearnImage

4 Event processing and Callbacks

4.1 GUI Components

The Gui Components are responsible for control of the model by the user.
Events that are emitted from the Gui in the form of signals. These signals are caught in MyMainWindow and/or any other component that needs to process them.

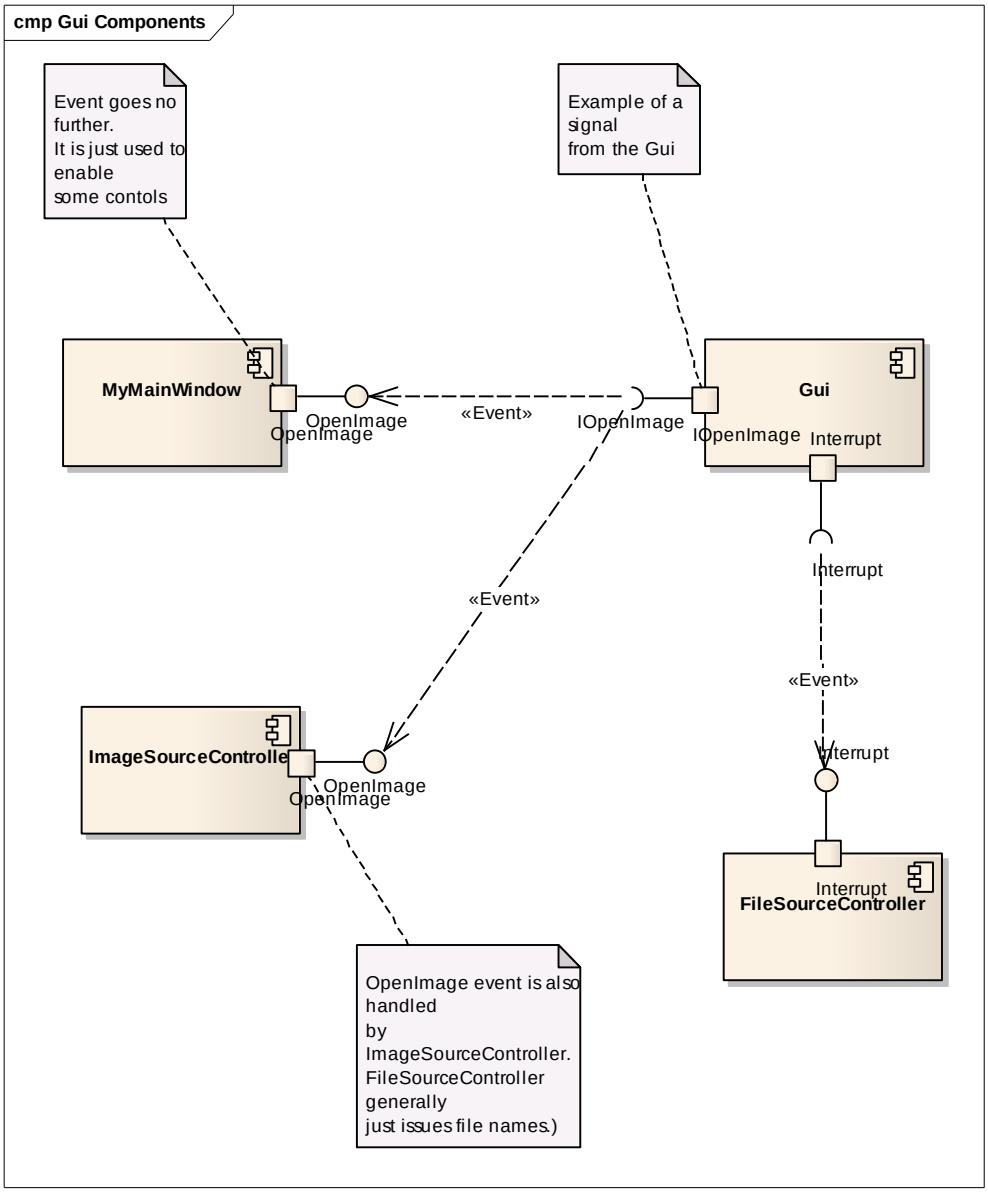

Figure 6 Example Gui events

4.2 Framework Components

Thinker uses the VisionService callback to obtain images for learning or recognition. The FileService callback is used to obtain file names.

The ImageSourceController uses one of the ISaccade-derived objects (represented as SaccadeImpl below) to obtain a stream of images (one per callback to IVisionService) that are re-emitted as events/signals back to Thinker.

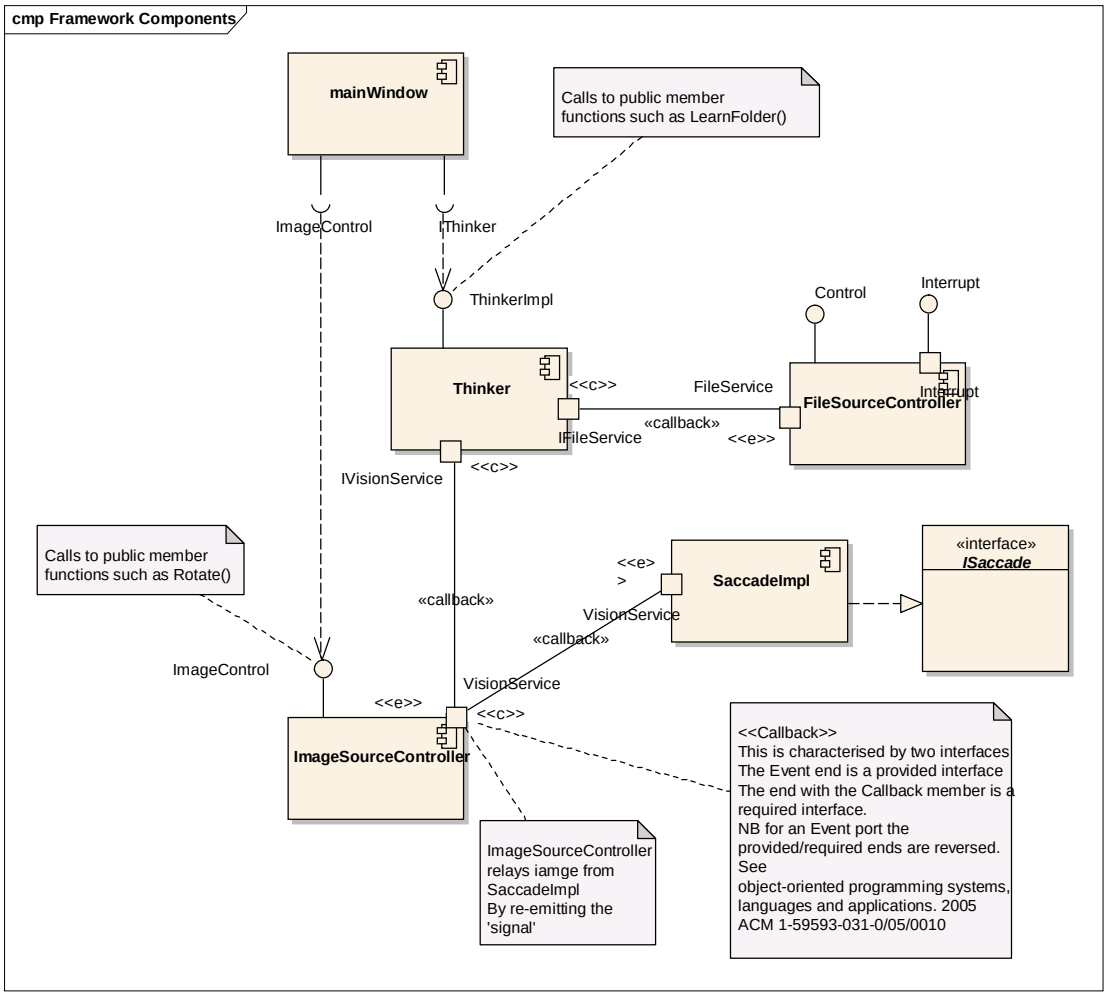

Figure 7 Main events for recognition and learning

Note <<c>> is the callback receiver stereotype for events <<e>>

4.3 Test Data

The following results were obtained with Test set Medium and default parameters. The Original Code version is 'MPF 4 Levels.exe'.

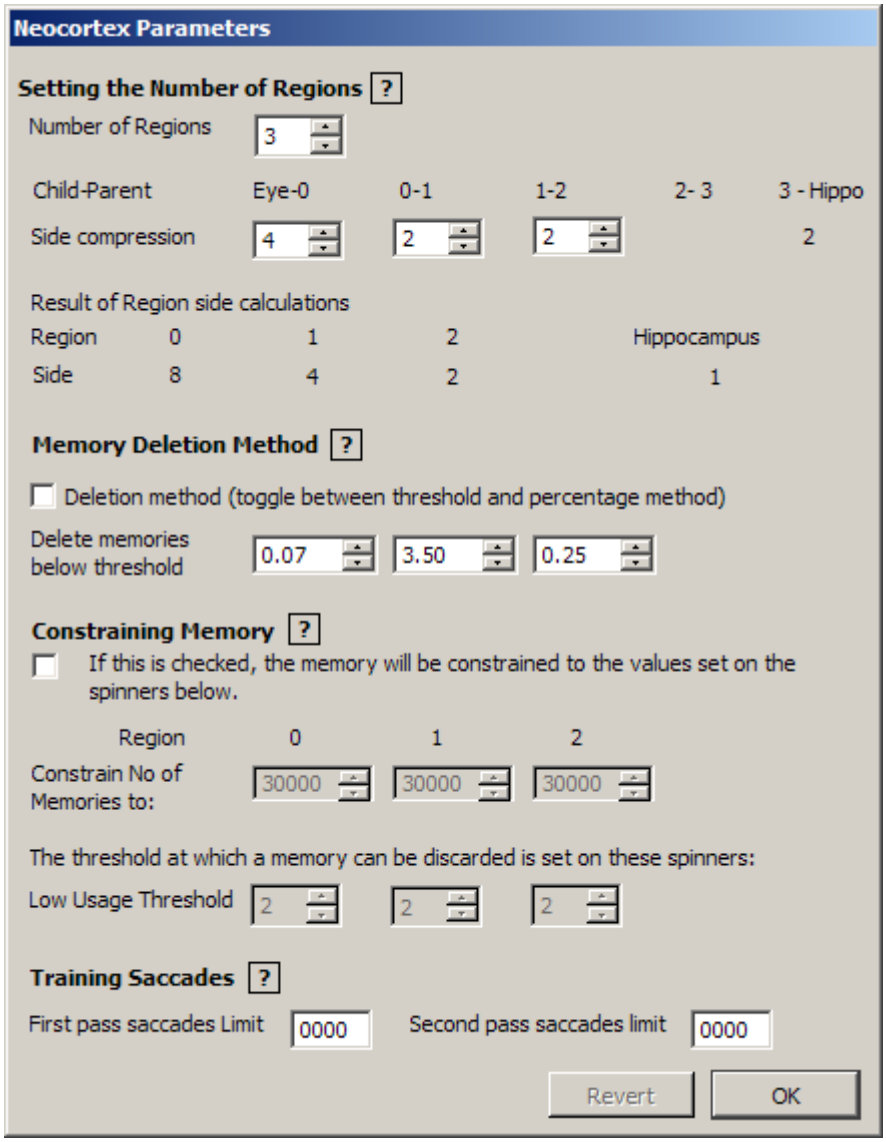

Figure 8 'Standard' parameters to test Neo

4.3.1 Results

4.3.1.1 Neo Program using Studio 2005

```
Region 0 total observations: 2037760. Memories before forgetting: 134. Memories after
forgetting : 77
Region 1 total observations: 509440. Memories before forgetting: 2984. Memories after
forgetting : 19
Region 2 total observations: 127360. Memories before forgetting: 3733. Memories after
forgetting : 913
```

4.3.1.2 Original code using Turbo C++

```
Region 0 observed: 2037760, memories before forgetting: 134, after: 77
Region 1 observed: 509440, memories before forgetting: 2984, after: 19
Region 2 observed: 127360, memories before forgetting: 3733, after: 913
```

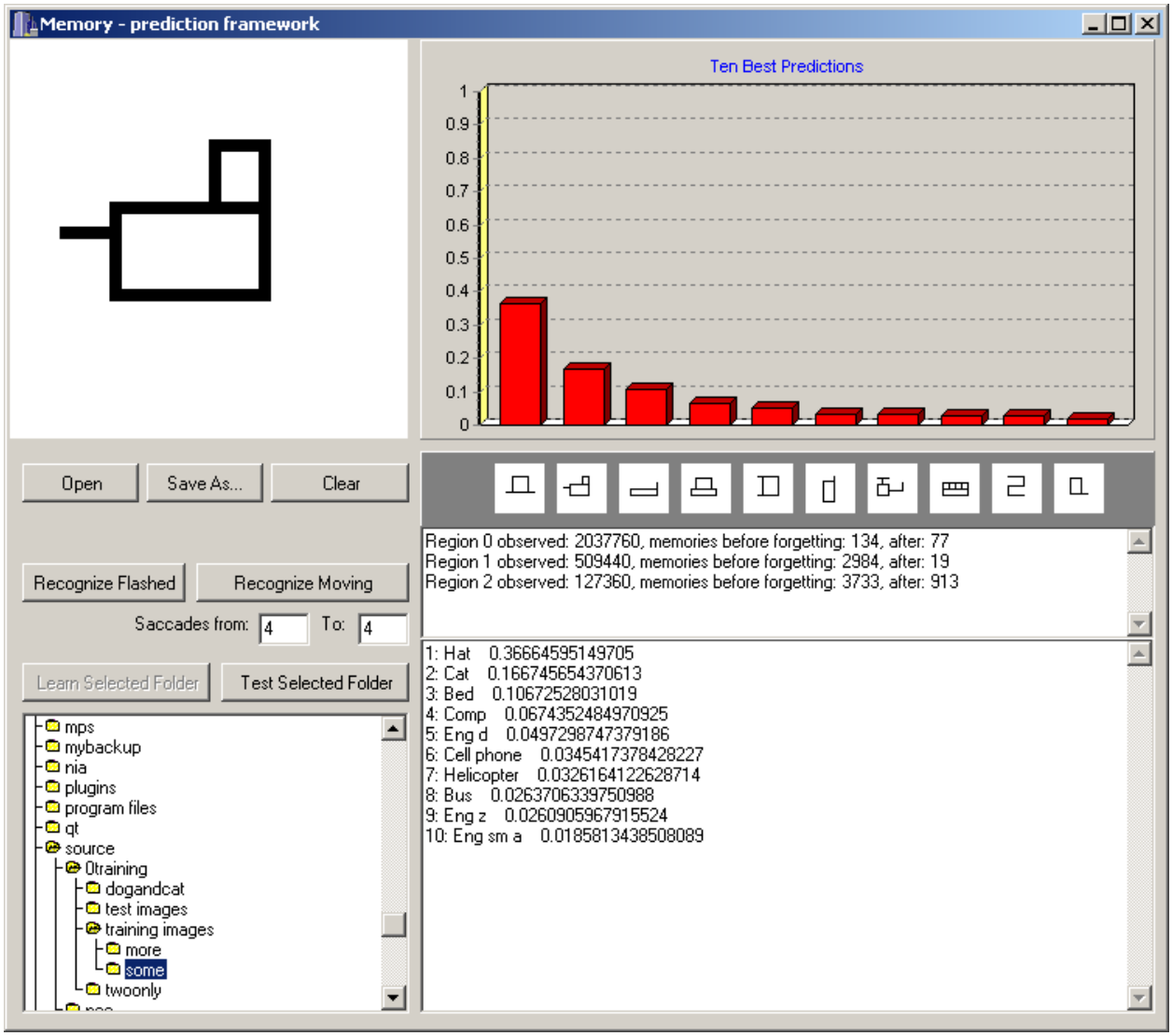

Figure 9 Results with Original 3 layer program. (Un-normalized)

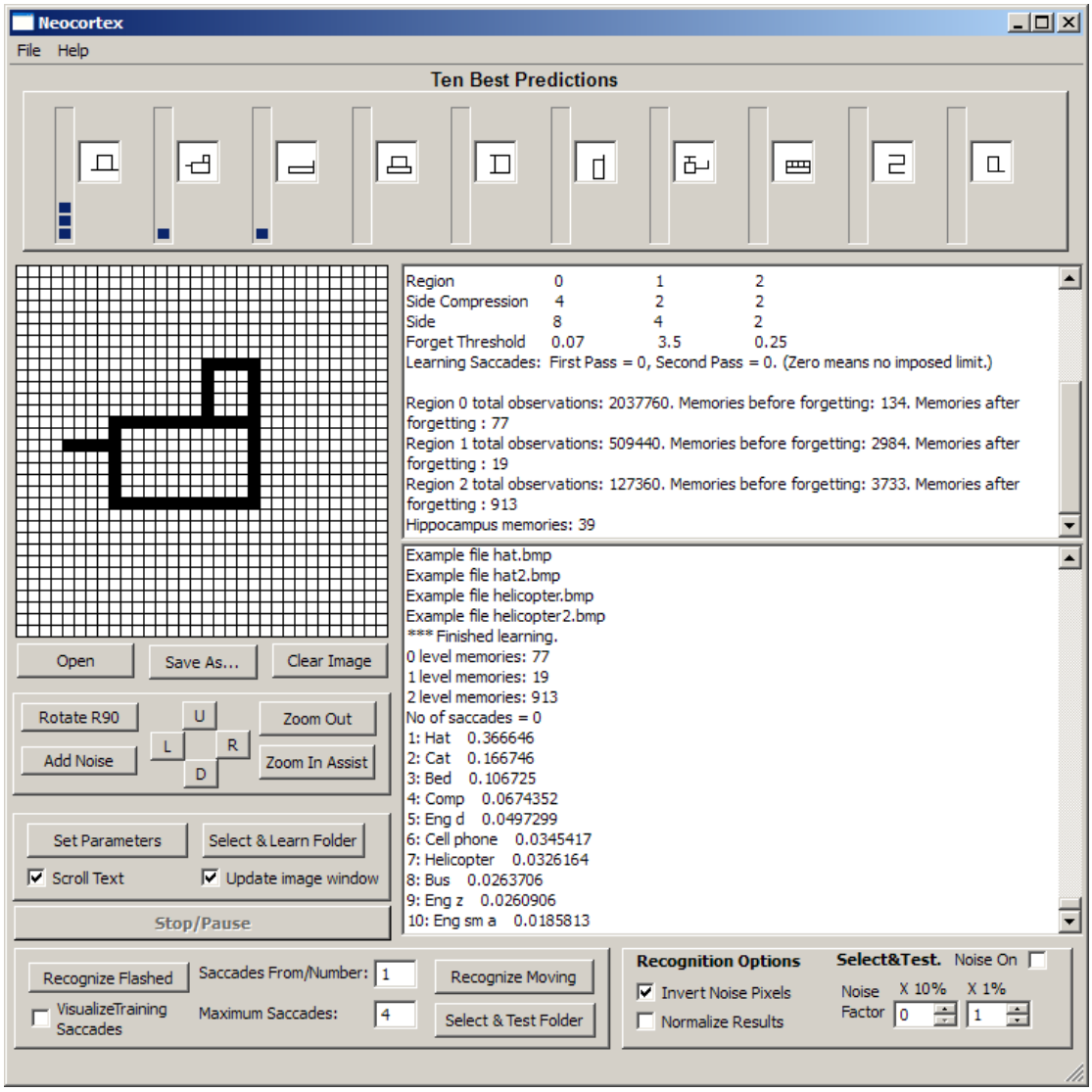

Figure 10 Results with Neo version 1.4. (Un-Normalized)