ECE 364 Project: Steganography
Phase II

Due: April 20, 2015

Steganography Browser

Completing this project phase will satisfy course objectives CO2, CO3 and CO4

Instructions

- Work in your Lab11 directory.
- There are no files to copy for this lab.
- This is the second of two phases for the course project for ECE 364. Each phase is worth 12% of your overall course grade.
- This is an **individual** project. All submissions will be checked for plagiarism using an online service.
- Remember to add and commit all files to SVN. Also remember to use your header file to start all scripts. **We will grade the version of the file that is in SVN!**
- Name and spell your scripts exactly as instructed. When you are required to generate output, make sure it matches the specifications exactly, since your scripts will be graded by a computer.
- The use of PyCharm is strongly encouraged, but *not* mandatory. If you have not used PyCharm before this lab, you might want to use a regular text editor, as the TAs are not responsible for teaching you how to use the PyCharm IDE.
- If you are using PyCharm, make sure you are using Python 3.X for your project. Otherwise, the IDE will give you pain. To verify/modify the Python version, go to:

  File Menu → Settings → Project Interpreter

  And make sure that Python 3.X is selected.

Introduction

By now, you should have completed the first phase of the project, where you have implemented embedding and extracting of different message types within an image medium, without affecting its visual appearance. In this phase, you are going to implemented a simple File Browser Application, using PySide and the Qt Framework, that uses your "Phase I" code. The purpose of the Browser Application is to quickly identify what images have messages in them, and what these messages are. Furthermore, it needs to offer a facility to remove the messages from their mediums as a mechanism for clean-up. For this phase, all of the functionality that you will implement must be accessible from the GUI, and not from the command-line.

The GUI Design

You are given the UI file `SteganographyGUI.ui` that has been designed using the Qt Designer. Convert it into `SteganographyGUI.py` to be usable by Python, and create a file called `SteganographyBrowser.py` and do your work in it. The GUI main window is shown in Figure 1. Aside from the Qt widgets that you have used before in the lab, in this application you will utilize: the `Tree Widget`, the `Graphics View` and the `Stacked Widget`. You should check the PySide documentation to understand the functionality of these widgets.

The `Stacked Widget`, and similarly the more popular `Tab Widget`, allows you to create multiple widgets in your window, but only display a subset of them at a time. To do so, the widget uses the concept of "pages", where it can contain multiple pages, with any number of widgets in each one. The application, or the user, selects what page to display at any point in time. In this application, we are going to use the `Stacked Widget` to switch between two pages: Page 1 contains a `Graphics View` for displaying images, and Page 2 contains a `Line Edit` for displaying text. Note that, while this is acceptable for this project, this is not always a good practice; because the undisplayed widgets still consume some resources.

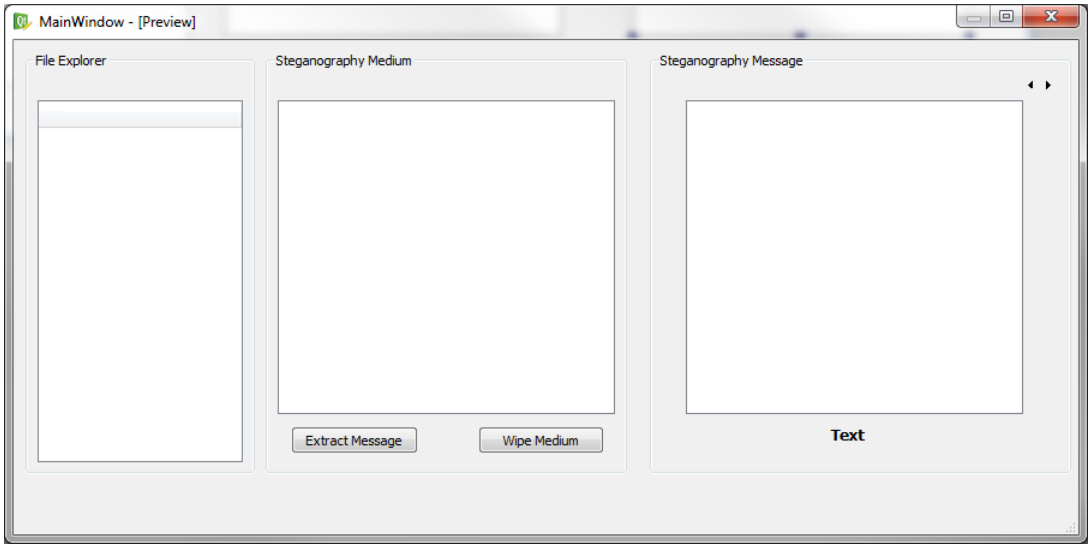


Figure 1: The File Browser GUI.

"NewSteganography" Class

You will need to extend your `Steganography` class to include some additional functionality. Create a new class named `NewSteganography` that inherits from `Steganography` and adds the following two new functions:

1. `wipeMedium(self)`:

   Erase the message from the medium by resetting the Least Significant Bit in every pixel to 0, and save the medium image to the same location.

2. `checkIfMessageExists(self)`:

   Check if the current medium contains a message or not, *without* performing a full extraction. If a message exists, return the tuple (`True, messageType`) where `messageType` is `Text`, `GrayImage` or `ColorImage`. If a message does not exist, return the tuple (`False, None`). This function should give an instantaneous answer, in order to have a responsive UI.

Application Behavior

Initial State

Before displaying the main window, the application should start by popping up a Dialog Box asking the user to select a "Folder" that contains some images. (If the user does *not* select any folder, then the application should *not* start.) The application will, then, examine all of the image files in that folder to determine which ones have messages embedded in them (along with the messages' types,) and which ones do not. Once all the images are examined, the main window can then be displayed, with only the `Tree Widget` enabled. This will be the application's "Initial State", which will help drive the rest of application behavior.

An example of an initial-state window is shown in Figure 2, where the images that do not contain any messages are shown in Blue (with no sub-items), while the ones that do are shown in **Bold Red**, with the message type shown as a sub-item in Green (You may choose a different set of colors other than those three shown, and you also may choose to give a different color for each message type.) Note how all of the widgets have been disabled to force the user to select an image from the list.

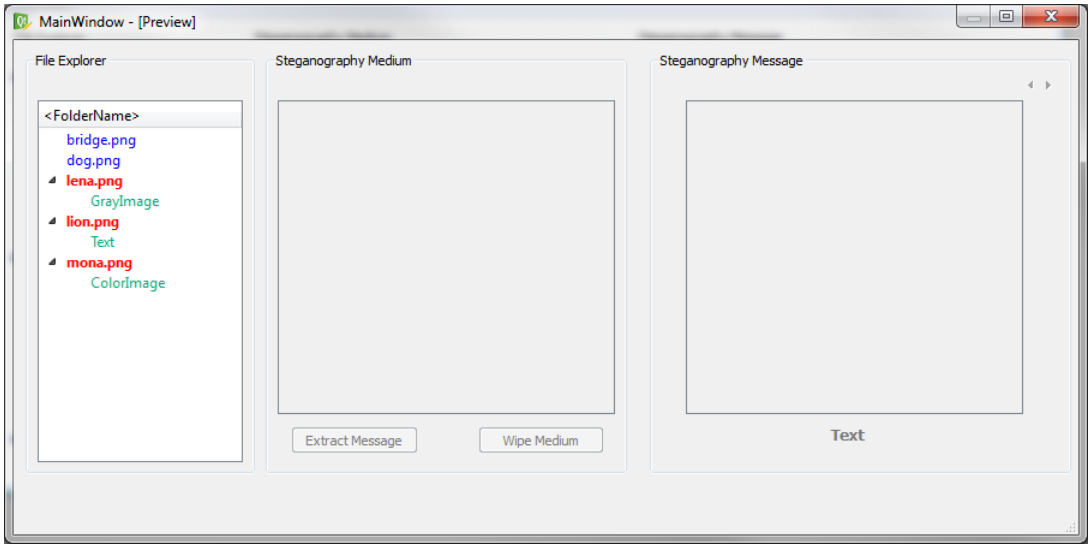


Figure 2: The Initial State of the Application.

Image Selection & Message Extraction

From the initial state, the user can select an image name from the `Tree Widget` to display in the "Steganography Medium" `Graphics View`. The application state will depend on the type of the image selected, and

should be as follows:

- If the image selected does *not* contain any message, the image is only displayed, but no other changes should occur.
- If the image selected **does** contain a message, the image is displayed and all of the other widgets are enabled. Additionally:
  - If the message type is "GrayImage" or "ColorImage", the `Stacked Widget` is set to display the page with the `Graphics View`.
  - If the message type is "Text", the widget is set to display the page with the `Line Edit`.

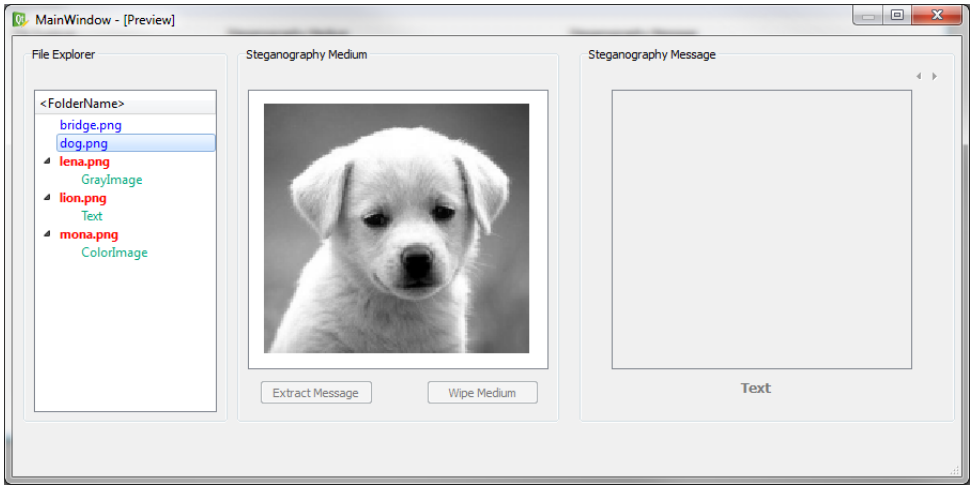
  Figure 3 shows examples of the application's state at each image type. Note that, at this point, the message is *not* yet extracted.
- To extract a message, the user must click on "Extract Message" to perform the extraction, and display the message, be it a text or an image, in the appropriate widget. Figure 4 shows the application state after extraction. (Note how the "Extract Message" button has been disabled.)

Wiping the Medium Image

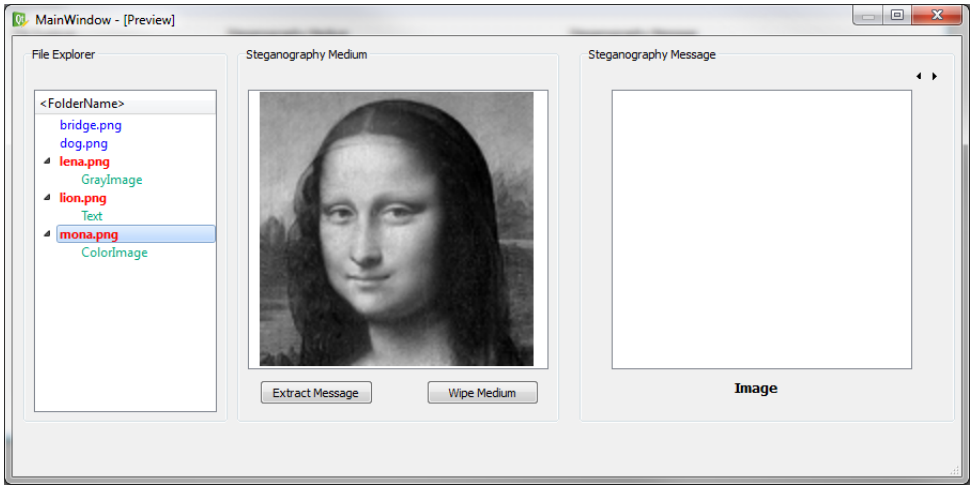
At any point in the application lifetime, when a medium containing a message is selected, if the user decides to erase the message from a medium and clicks on the button "Wipe Medium", the application should perform the following actions:

- Pop up a `QMessageBox` asking the user to confirm an irreversible action. Continue only if the user agrees.
- Clear the message display widget (i.e. the `Graphics View` or the `Line Edit`) if the message has been extracted, and disable the "Steganography Message" widget.
- Erase the message from the medium by reseting the LSB to 0 in all pixels.
- Save the wiped medium image in the same location with the same name.
- Update the `Tree Widget` to reflect that the medium does not contain a message anymore.

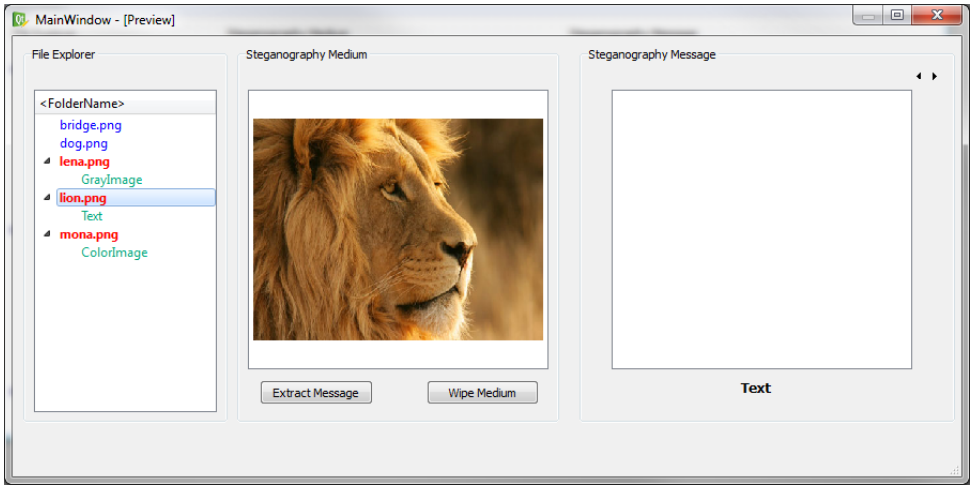
Figure 5 shows an example of the application state before and after wiping a medium.



(a) Medium with no message

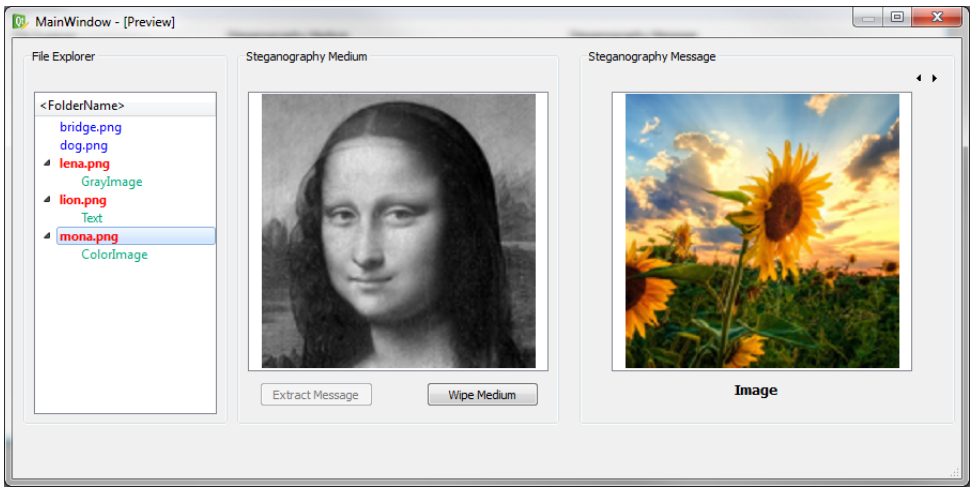


(b) Medium with an image message



(c) Medium with a text message

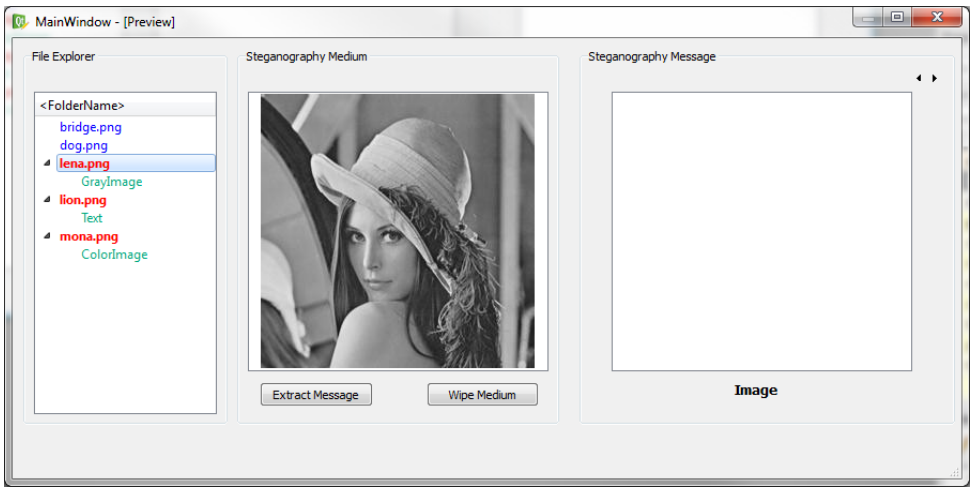
Figure 3: Application state when an image is selected.



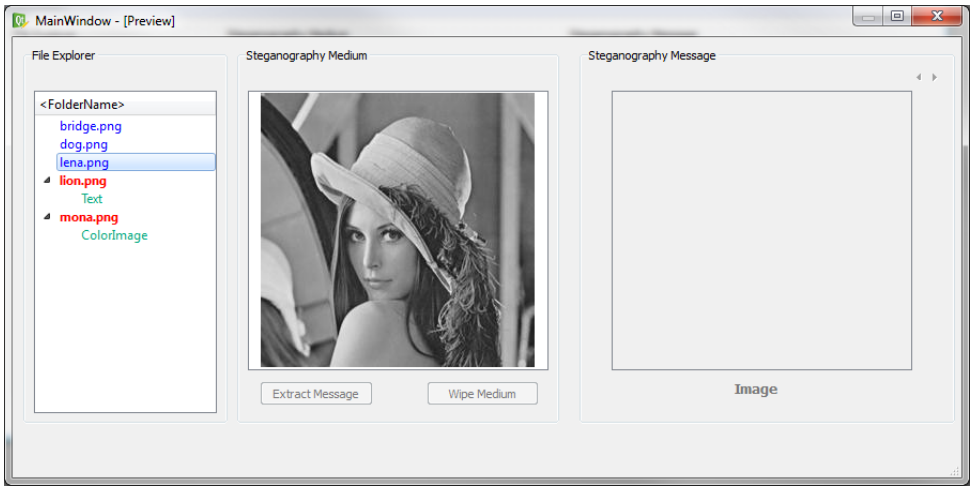
(a) Image message extracted

(b) Text message extracted

Figure 4: Application state when a message has been extracted.



(a) Before wiping the medium



(b) After wiping the medium

Figure 5: Application state when a medium is wiped.

Extra Credit

Decryption

Extend the capability of your Browser Application to include decryption. When the user requests message extraction, if the message detected in a medium is encrypted, you will need to perform the following:

- Pop up a Message Box requesting a password. Make sure you "mask" the user input in the text widget so that the password is not displayed.
- Perform the decryption only if the user clicks "OK". Otherwise, return to the application without decryption, and do not display the message content.
- If the message type is "Text", it is possible to verify whether the password entered is correct or not. If the password is correct, perform the decryption and display the data in the `Line Edit` widget. Otherwise, pop up a Message Box indicating that the password entered is wrong and return without displaying the message.

Grading Requirements

In addition to checking in your code files, you will need to demonstrate the functionality of your program to your TA to complete this Phase. Perform the following tasks to your TA:

1. Start the application → Do *not* select a folder.
   Expected: Form will not display, and application will exit.
2. Start the application → Select a folder.
   Expected: Form is shown in the initial state **within reasonable time**, and with the right image information.
3. From Initial State → Click on an empty medium.
   Expected: Medium image is displayed with no other widgets enabled.
4. Click on a medium with a text message.
   Expected: Medium image is displayed with widgets enabled. No extraction yet.
   (Repeat with gray & color images.)
5. Click on a medium with a text message → Click on "Extract Message".
   Expected: Medium image is displayed. Message is displayed. "Extract Message" is disabled.
   (Repeat with gray & color images.)
6. Click on a medium with a text message → Click on "Wipe Medium".
   Expected: Message is wiped. `Tree Widget` updated to reflect empty medium. Widgets are disabled.
   (Repeat with gray & color images.)
7. Optional: Click on a medium with an encrypted text message → Click on "Extract Message" → Enter correct password.
   Expected: Medium image is displayed. Password is requested. Message is displayed. "Extract Message" is disabled.
   (Password must be masked.)
8. Optional: Click on a medium with an encrypted text message → Click on "Extract Message" → Enter wrong password.
   Expected: Medium image is displayed. Password is requested. Message is *not* displayed. "Extract Message" is still enabled.
   (Password must be masked.)

Comments

- While some examples in the document show a color medium, only gray-scale mediums will be used for the demo of this phase. The color medium is just for illustration, and you do not have to implement/use the ColorSteganography; since it was optional.
- When you display the image in the Graphics View Widget, please note the following:
  - It is acceptable for the image to be smaller and not fill the whole widget.
  - It is *not* acceptable for the image to be larger than the widget, which will cause scrollbars to be visible in the widget.
  - It is *not* acceptable for the image to be deformed. In other words, the aspect ratio of the image must be maintained.
- Note that selecting the scanning direction has not been offered to the user. You will need to handle both directions internally in your program. The test images will have a combination of horizontal and vertical embedding.
- All of the functions **must be commented.** You will lose points if any function does not contain any documentation/comments.
- Your grade for this phase is based on your demo to the TA. **Failing to demonstrate your work to your TA will result in a zero credit for this phase**. Moreover, please note that the final judgment for passing the test cases rest with the TAs. If you have any doubt about how any case should work, it pays to ask first before you find out at demo time.
- Your submission must consist of the files:
  1. `SteganographyGUI.ui`.
  2. `SteganographyGUI.py`.
  3. `SteganographyBrowser.py`.
  4. `NewSteganography.py`.