**Steganography Project Design**

Linden Crandall, Jonathan Mainhart, Zhihua Zheng

University of Maryland Global Campus

CMIS 495: Current Trends and Projects in Computer Science

Prof. Majid Shaalan

April 12, 2022

Table of Contents

[List of Figures 3](#_Toc100260287)

[Project Overview 4](#_Toc100260288)

[Main Goals 4](#_Toc100260289)

[Design Concept 4](#_Toc100260290)

[File Access 4](#_Toc100260291)

[Data Structures 4](#_Toc100260292)

[Input and Output 4](#_Toc100260293)

[Typical User Interaction 5](#_Toc100260294)

[User Interface 6](#_Toc100260295)

[Application GUI Overview 6](#_Toc100260296)

[Error Message Popup Dialog 6](#_Toc100260297)

[Message Encoding and Decoding 7](#_Toc100260298)

[Encoding an Image 7](#_Toc100260299)

[Decoding an Image 7](#_Toc100260300)

[Resetting an Image 8](#_Toc100260301)

[Class, Attributes, and Methods 8](#_Toc100260302)

[stego.py – MainFrame class 8](#_Toc100260303)

[gui.kv – MainFrame file 8](#_Toc100260304)

[dialog.kv – PopupDialogWidget file 9](#_Toc100260305)

[models.py – ImageObject class 10](#_Toc100260306)

[utils.py – Utility Functions class 10](#_Toc100260307)

# List of Figures

[Figure 1. Logic Control Flow of a typical user interaction. 5](#_Toc100259759)

[Figure 2. Application Graphical User Interface 6](#_Toc100259760)

[Figure 3. Error Message Popup Dialog 7](#_Toc100259761)

# Project Overview

## Main Goals

Provide a GUI application which runs locally on any computer which meets the following minimum system requirements:

* Python 3.9 or greater installed
* A keyboard or similar text input device
* A mouse or similar pointing device
* A monitor or similar display capable of rendering images

Allow a single user to open an image file and determine whether a secret text message has been encoded in the pixel data.

Allow a user to enter a text message into a text input area and encode that text into the pixel data of an image.

Allow a user to save an encoded image file with the same or different filename.

Alert the user to any problems reading or writing files from within the application.

## Design Concept

The application will contain two main classes: The GUI, and the ImageObject. The GUI class will contain the event handlers and runtime variables. The ImageObject class will contain the image attributes such as filename and pixel data, as well as methods to encode and decode embedded text messages. Several utility functions will be created to open files, save files, and perform other tasks as required that do not fit into the object class paradigm.

## File Access

The application will require read access to image files on the user’s system to provide visual feedback to the user.

## Data Structures

An instance of an ImageObject will be declared at runtime. This object will use Python base types of String, list, Boolean, and int to record data about a user selected image.

User input will UTF-8 characters stored as a String.

## Input and Output

User text input will be entered using stdin.

User visual input will be entered using a mouse or similar pointing device controlled by the operating system.

Image files will be read from and written directly to the filesystem.

# Typical User Interaction

Figure 1 shows a typical user interaction with the application.

First, the user starts the program and chooses an image via the file selection interface. The image is then displayed to the user in the window.

Next, the image pixel data is then automatically extracted and sent to the message decoding algorithm. If a message can be decoded, it will be displayed inside the text input area of the window, otherwise a default message will be displayed.

Next, the maximum allowed character limit will be calculated and displayed to the user as a ratio of characters entered to characters allowed.

The user may then delete, edit, or append the text within the text input area until the character limit is reached. When the user is ready, they can push the “encode” button.

When the “encode” button is pressed, the user’s input will be converted to its binary value. The binary value of the message will then be used to encode the message in the RGB value of the image’s pixels.

Upon completion of the encoding process, the user may then choose to reset the pixel data to its original state, or to save the image for later use. The user will be alerted if they attempt to overwrite a file.

If the image is reset, the original message (if any) will be displayed in the text input area.

Diagram

Description automatically generated

Figure . Logic Control Flow of a typical user interaction.

# User Interface

## Application GUI Overview

The Graphical User Interface (GUI) of the application shown in figure 2 will have an image display area and four function buttons on the right side of the image. The function buttons include “Open Image”, “Encode Image”, “Reset Image” and “Save Image”. The “Reset Image” button will be initially disabled until an image is successfully encoded. There will be a text field at the bottom of the GUI to enable the user to enter the secret data to be encoded and to read messages that have been decoded from images. There will be two labels between the text field and the image display area to display warning messages regarding the text field (e.g., The image has been reset!) and a ratio expressing the number of characters entered to the maximum allowed.

A screenshot of a dog

Description automatically generated

Figure 2. Application Graphical User Interface

## Error Message Popup Dialog

All the error messages of the application will be displayed in a popup dialog as shown in figure 3. The application icon will be displayed in the up center of the popup dialog. Then the error message will be displayed under the icon. Finally, there will be an “OK” button, that enables the user to click the button and close the popup dialog.

A screenshot of a computer

Description automatically generated with medium confidence

Figure . Error Message Popup Dialog

# Message Encoding and Decoding

## Encoding an Image

Messages will be encoded into the target image by first extracting the image pixel data into a list of tuples which represent the red, green, and blue (RGB) color values of the image.

Next, each character of the user’s message will be converted to its 8-bit binary ASCII value. The 8-bit values will then be used to shift the RGB values to be either an even number to represent a binary zero, or an odd number to represent a binary one. Pixels will be read 3 at a time which will allow one 8-bit character to be encoded. The remaining bit of each set will tell the decoder to either continue reading (binary one), or to stop (binary zero).

## Decoding an Image

Messages will be decoded by first extracting the image pixel data into a list of tuples which represent the RGB color values of the image.

Next, the color value will be determined to be either even or odd. An even number will result in a binary 0 being appended to a string, an odd value will result in a binary 1 being appended to the same string. The ninth color value will determine whether to continue reading (binary 1), or to stop reading (binary 0) and be discarded.

Next, the string of binary digits will be converted to a string of characters and returned.

## Resetting an Image

Images may be reset to their original state by copying the values of the backup\_pixel\_data to the rgb\_pixel\_data attribute. The backup\_pixel\_data attribute shall be immutable.

# Class, Attributes, and Methods

The following tables list the classes, attributes, and methods.

**Table 1**

## stego.py – MainFrame class

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Source** |
| display\_image | ImageObject | open\_image() |
| **Method** | **Input** | **Output** |
| on\_open\_button\_click | self | filename |
| on\_encode\_button\_click | self | encode\_image(rgb\_pixel\_data, gui\_text\_input) |
| on\_reset\_button\_click | self | reset\_image() |
| on\_save\_button\_click | self | save\_image(encoded\_rgb\_pixel\_data) |
| popup\_error\_dialog | self, String | N/A |

**Table 2**

## gui.kv – MainFrame file

|  |  |  |  |
| --- | --- | --- | --- |
| **Element Name** | **Type** | **Properites** | **Action** |
| MainFrame | Main Interface | background-color: black  size: 550x500 | N/A |
| Image | Image | source: default or filename returned from on\_open\_button\_click()  size: 330x250  position: (30, 20) | N/A |
| Open Image | Button | background-color: # 4CAF50  text color: white  size: 110x40  position: (390, 40)  enabled by default: True | on\_open\_button\_click() |
| Encode Image | Button | background-color: # 4CAF50  text color: white  size: 110x40  position: (390, 100)  enabled by default: True | on\_encode\_button\_click()  enables when file opened |
| Reset Image | Button | background-color:  enabled - # 4CAF50  disabled – grey  text color: white  size: 110x40  position: (390, 160)  enabled by default: False | on\_reset\_button\_click()  enables when message encoded |
| Save Image | Button | background-color: # 4CAF50  text color: white  size: 110x40  position: (390, 220)  enabled by default: True | on\_save\_button\_click()  enables when file opened |
| gui\_label\_1 | Label | default text: warning message  size: 20x280  position: (470, 20)  text color: red | N/A |
| gui\_label\_2 | Label | default text: (0/0) No Image Loaded  text: (n/p) characters remaining  text: Maximum (p) characters entered  size: 50x20  position: (460, 450)  text color: white | N/A |
| gui\_text\_input | TextInput | default text: decoded message from loaded image  text: user input text  size: 490x130  position: (20, 310) | N/A |

**Table 3**

## dialog.kv – PopupDialogWidget file

|  |  |  |  |
| --- | --- | --- | --- |
| **Element Name** | **Type** | **Properites** | **Action** |
| dialog\_label\_1 | Label | background-color: black  size: 300x300 | N/A |
| dialog\_label\_2 | Label | text color: red  size: 220x50  position: (40, 150) | N/A |
| dialog\_button\_1 | Button | background-color: # 4CAF50  text color: white  size: 100x230  position: (390, 40)  enabled by default: True | N/A |

**Table 4**

## models.py – ImageObject class

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Source** |
| filename | String | open\_image() |
| rgb\_pixel\_data | list | self.extract\_pixel\_data() |
| backup\_pixel\_data | list | rgb\_pixel\_data |
| max\_available\_chars | int | self.calculate\_max\_chars() |
| decoded\_message | String | self.decode\_image() |
| **Method** | **Input** | **Output** |
| \_\_init\_\_() | self, filename | ImageObject |
| extract\_pixel\_data() | self | list |
| calculate\_max\_chars() | self | int |
| encode\_image() | self, String | list |
| decode\_image() | self | String |
| reset\_image() | self | list |

**Table 5**

## utils.py – Utility Functions class

|  |  |  |
| --- | --- | --- |
| **Function** | **Input Parameters** | **Return Value** |
| convert\_message() | String: user\_input | list: converted\_message(08b) |
| open\_image() | None | String: filename |
| save\_image() | List: rgb\_pixel\_data | None |
| user\_notification() | String: message | None |