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

List of Tables

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 view a secret text message that 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.

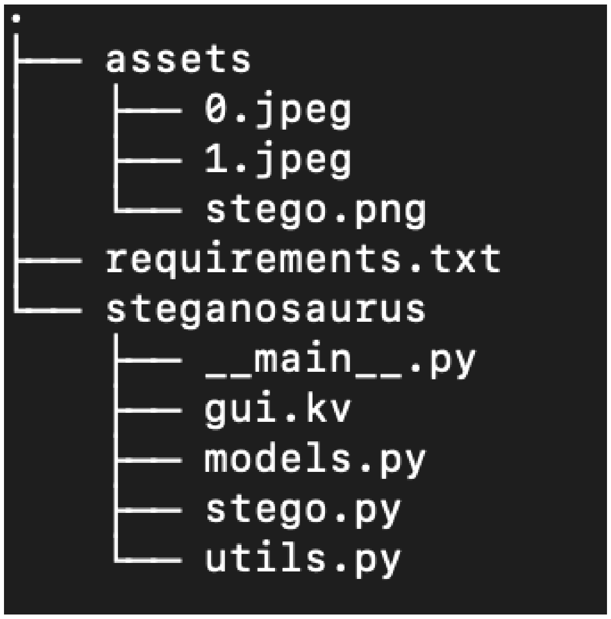
Alert the user to any problems reading or writing files from within the application.

Design Concept

The application will contain two main classes: MainFrame and ImageObject. The MainFrame class will contain the GUI 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.

<u>File Structure</u>

The application files will be structured as shown in figure 1.



```
.
├── assets
│   ├── 0.jpeg
│   ├── 1.jpeg
│   └── stego.png
├── requirements.txt
└── steganosaurus
    ├── __main__.py
    ├── gui.kv
    ├── models.py
    ├── stego.py
    └── utils.py
```

Figure 1. File structure

<u>File Access</u>

The application will require read access to image files on the user's system to enable the user to encode and decode messages inside of the pixel data of the images.

<u>Data Structures</u>

An instance of an ImageObject will be declared at runtime. This object will use Python base types of String, list, 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 2 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.

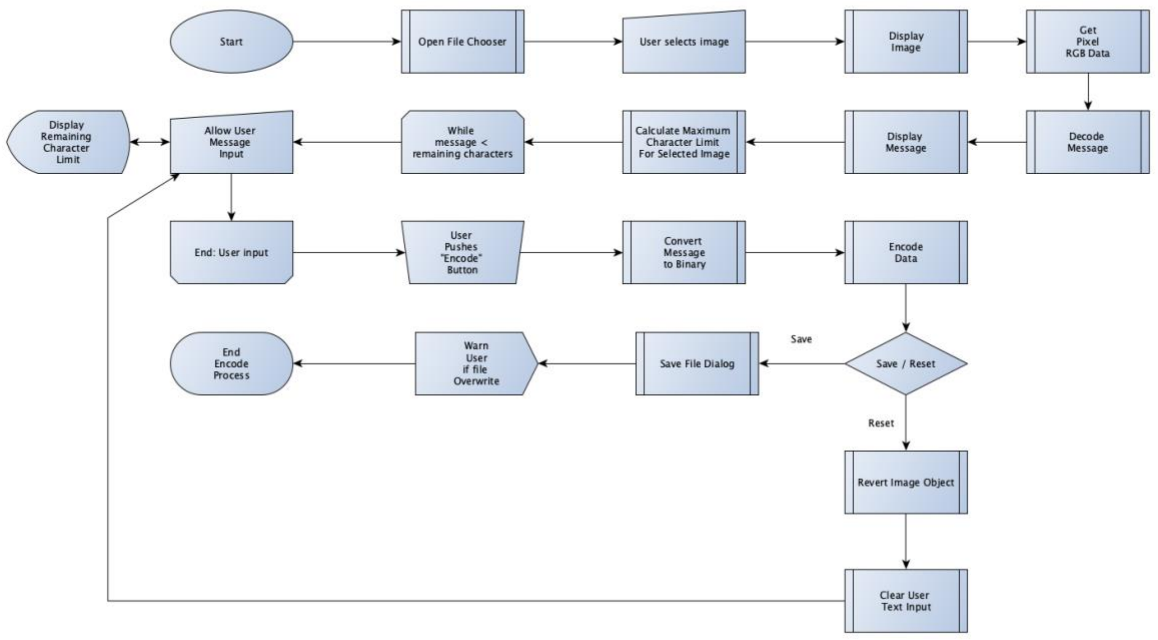


Figure 2. Logic Control Flow of a typical user interaction

User Interface

GUI Overview

The Graphical User Interface (GUI) of the application shown in figure 3 will have an image display area and four function buttons on the right side of the image in the upper half of the window. 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 input 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 one label between the text field and the image display area to display warning messages regarding the text field (e.g., Maximum chars limit is reached.) and a label below the text field displaying a ratio expressing the number of characters entered to the maximum allowed.

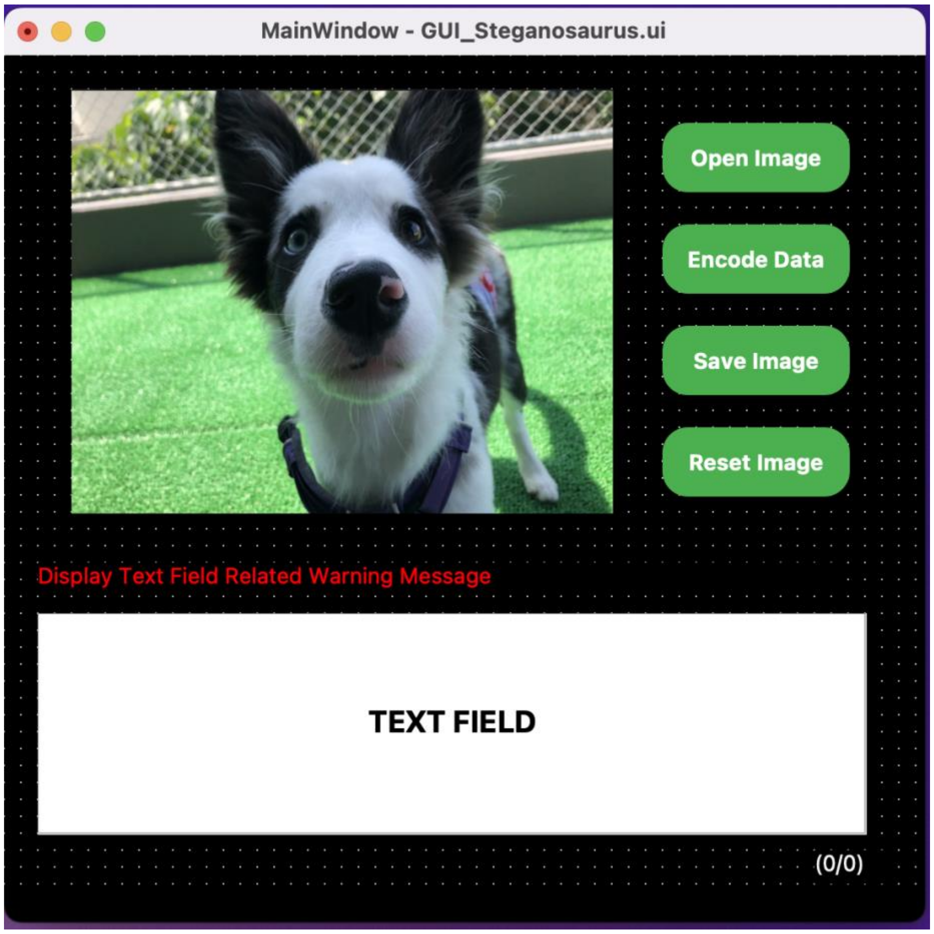
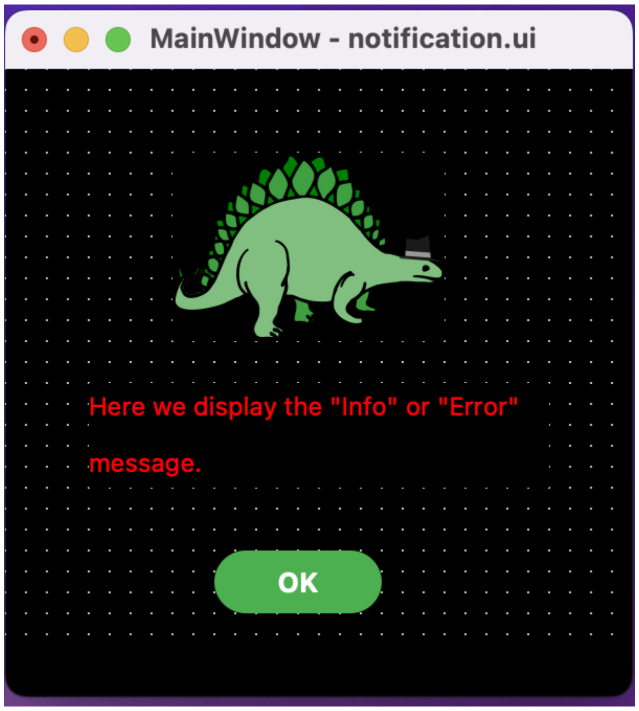

MainWindow - GUI_Steganosaurus.ui
Open Image
Encode Data
Save Image
Reset Image
Display Text Field Related Warning Message
TEXT FIELD
(0/0)

Figure 3. Graphical User Interface

User Notification Dialog

User messages resulting from errors and runtime exceptions will be displayed in a popup dialog as shown in figures 4 and 5. The application icon will be displayed in the upper center of the popup dialog and the message will be displayed under the icon. At the bottom of the popup dialog there will be an "OK" button for the "info" and "error" type message that enables the user to click the button and close the popup dialog. For the "warning" type message, there will be two buttons under the message display label, that gives the user an option to click either "Yes" to continue or "No" to cancel.



Here we display the "Info" or "Error" message.

Figure 4. Info and error message popup dialog

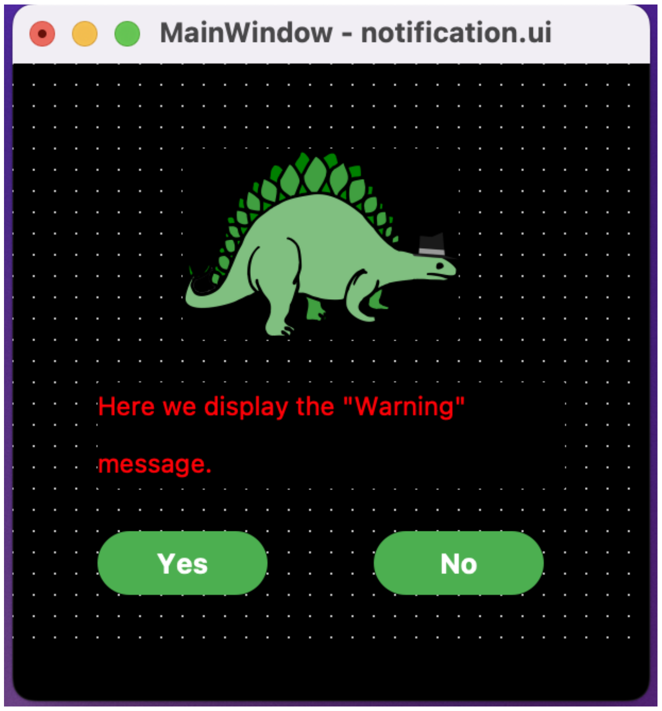

MainWindow - notification.ui
Here we display the "Warning" message.
Yes
No

Figure 5. Warning 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) when the message is later decoded.

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 will 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_user_notification	self, String: message, String: message_type	N/A

Table 2: gui.kv – MainFrame template

Element Name	Type	Properites	Action
MainFrame	Main Interface	background-color: black size: 550x500	N/A
DisplayImage	Image	source: display_image.filename size: 330x250 position: (30, 20)	N/A
OpenImageButton	Button	background-color: # 4CAF50 text color: white text: Open Image size: 110x40 position: (390, 40) enabled by default: True	on_open_button_click()
EncodeImageButton	Button	background-color: # 4CAF50 text color: white text: Encode Image size: 110x40 position: (390, 100) enabled by default: True	on_encode_button_click() enables when file opened
ResetImageButton	Button	background-color: enabled - # 4CAF50 disabled – grey text color: white text: Reset Image size: 110x40 position: (390, 160) enabled by default: False	on_reset_button_click() enables when message encoded
SaveImageButton	Button	background-color: # 4CAF50 text color: white text: Save Image size: 110x40 position: (390, 220) enabled by default: True	on_save_button_click() enables when file opened
TextInfoLabel	Label	default text: warning message size: 20x280 position: (470, 20) text color: red	N/A
TextCounterLabel	Label	default text: (0/0) No Image Loaded text: (n/p) characters remaining text: Maximum (p) characters entered size: 50x20 position: (460, 450)	N/A

		text color: white	
GUITextInput	TextInput	default text: decoded message from loaded image text: User Input size: 490x130 position: (20, 310)	N/A

Table 3: dialog.kv – PopupDialogWidget template

Element Name	Type	Properites	Action
DialogIconLabel	Label	background-color: black size: 300x300	N/A
DialogNotificationLabel	Label	text color: red size: 220x50 position: (40, 150)	N/A
DialogButtonYes	Button	background-color: # 4CAF50 text color: white size: 100x230 position: (390, 40) enabled by default: True	N/A
DialogButtonNo	Button	background-color: # 4CAF50 text color: white size: 80x30 position: (40, 220) enabled by default: True	N/A
DialogButtonOK	Button	background-color: # 4CAF50 text color: white size: 80x30 position: (170, 220) 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

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