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
- Modules listed in requirements.txt 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. It relies on several sub-classes to implement different visual features such as pop-up warnings and file chooser windows. The ImageObject class will contain the image attributes such as filename and image data, as well as methods to encode and decode embedded text messages. There are utility functions which perform required tasks that do not fit into the object class paradigm.

File Structure

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

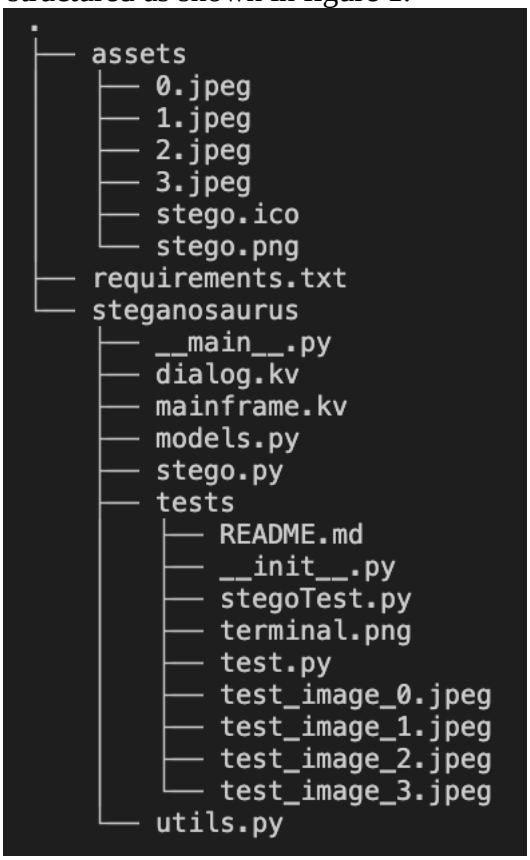

Figure 1. File structure

File Access

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.

Data Structures

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.

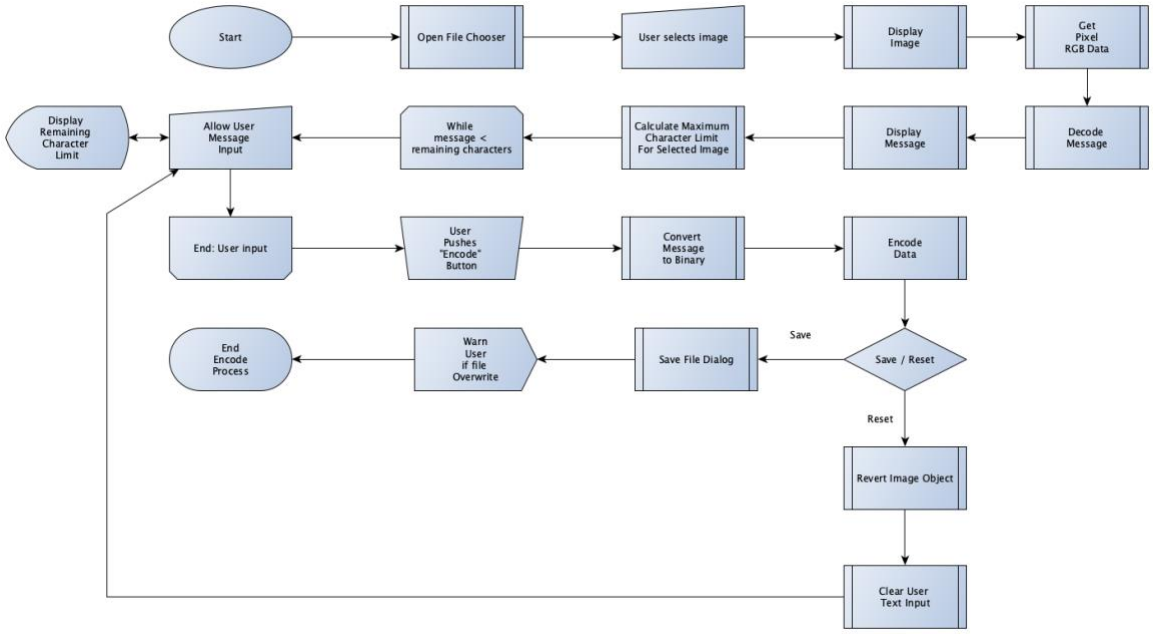

Figure 2. Logic Control Flow of a typical user interaction

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., Warning: Maximum encode character number has been reached) and a label below the text field displaying a ratio expressing the remaining number of characters to the maximum allowed.

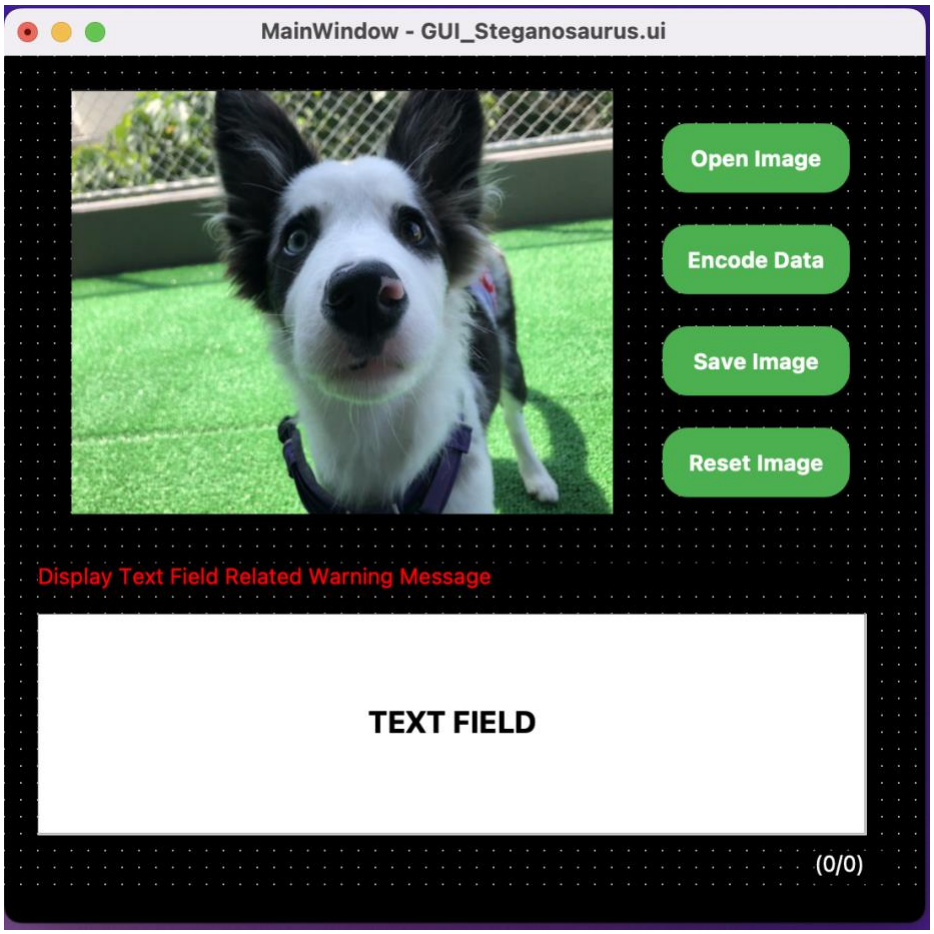

Figure 3. Graphical User Interface

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.

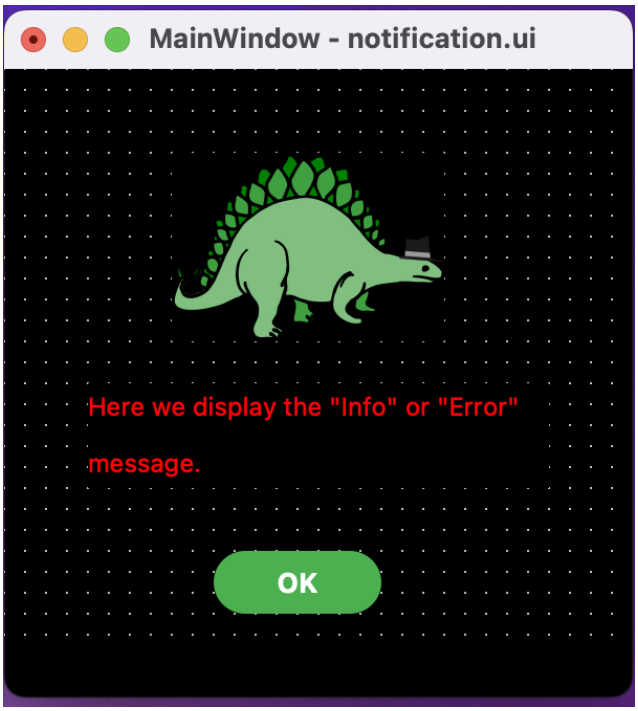

Figure 4. Info and error message popup dialog

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
main_title	String	MainFrame App
message	String	MainFrame App
message_type	String	MainFrame App
file_spliter	String	MainFrame App
current_filename	String	MainFrame App
LEGO_PATH	String	MainFrame App
reset_btn_disabled	BooleanProperty	MainFrame App, default value: True on_encode_button_click()
textfield_disabled	BooleanProperty	MainFrame App, default value: False
image_saver_dismiss	BooleanProperty	MainFrame App, default value: False
valid_image_name	BooleanProperty	MainFrame App, default value: True
MESSAGE_TYPE	Enum	MainWidget class
WARNING_TYPE	Enum	MainWidget class
display_image	ImageObject	MainWidget class
new_filename	String	MainWidget class
warning_type	String	MainWidget class
new_filepath	String	MainWidget class
encodable_bool	Boolean	MainWidget class
user_notification_msg	StringProperty	MainWidget class, Set up value dynamically
textfield_str	StringProperty	MainWidget class, GUI TextInput
maximum_char_count	NumericProperty	display_image.max_available_chars

Method	Input	Output
on_open_button_click	self	ImageChooserPopup().show_load_list()
on_encode_button_click	self	MainWidget.display_image.encode_image()
on_reset_button_click	self	self.popup_user_notification()
execute_reset	self	MainWidget.display_image.reset_image()
on_save_button_click	self	ImageSaverPopup().show_filechooser()
save	self, String: new_filepath String: new_filename	self.update_warning_btn_yes() self.popup_user_notification()
execute_save	self	MainWidget.display_image.save_image() MainWidget.display_image.decode_image()
validate_image_name	self, String: image_name	self.popup_user_notification(), Boolean
popup_user_notification	self, String: message, String: message_type	WarningPopup().open() InfoAndErrorPopup().open()
update_warning_btn_yes	self, Boolean: image_warning_btn_yes	self.update_textfield_input() self.execute_reset() self.execute_save() self.update_widgets_status()
update_widgets_status	self, String: reset_btn_disabled, String: textfield_disabled, String: image_saver_dismiss	N/A
update_main_widgets	self, *args	N/A
update_textfield_input	self	N/A

Table 2: stego.py – ImageChooserPopup class

Attribute	Type	Source
file_path	String	selected()
Method	**Input**	**Output**
__init__	self, **kwargs	Cache.remove()
show_load_list	self	self.open()
selected	self, String: filename	file_path MainWidget.popup_user_notification()
load_list	self	MainWidget.display_image self.dismiss()

Table 3: stego.py – ImageSaverPopup class

Method	Input	Output

__init__	self, **kwargs	Clock.schedule_interval()
show_filechooser	self	self.open()
dismiss_popup	self, *args	MainWidget.display_image self.dismiss()

Table 4: mainframe.kv – MainFrame template

Element Name	Type	Properites	Action
MainFrame	Main Interface	id: main_widget background-color: black size: 550x500	N/A
DisplayImage	Image	id: main_image source: display_image.filename size: 330x250	N/A
OpenImageButton	Button	id: open_btn background-color: # 4CAF50 text color: white text: Open Image size: 110x40 enabled by default: True	on_open_button_click()
EncodeImageButton	Button	id: encode_btn background-color: # 4CAF50 text color: white text: Encode Image size: 110x40 enabled by default: True	on_encode_button_click() enables when file opened
ResetImageButton	Button	id: reset_btn background-color: enabled - # 4CAF50 disabled – grey text color: white text: Reset Image size: 110x40 enabled by default: False	on_reset_button_click() enables when message encoded
SaveImageButton	Button	id: save_btn background-color: # 4CAF50 text color: white text: Save Image size: 110x40 enabled by default: True	on_save_button_click() enables when file opened

MessageLabel	Label	default text: warning message size: 550x40 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: 500x30 text color: white	N/A
GUITextInput	TextInput	default text: decoded message from loaded image text: User Input size: 500x150	N/A
RoundedCornerButton	Button	on_press: set background color to #3E8E41 on_release: set color back to # 4CAF50 canvas.before: set rounded rectangle to the button	N/A
WarningPopup	Popup	id: warning_dialog size: 300x300 WarningDialog	title: app.message_type dynamically display message type
InfoAndErrorPopup	Popup	id: warning_dialog size: 300x300 InforAndErrorDialog	title: app.message_type dynamically display message type
ImageChooserPopup	Popup	Size: 550x500 FileImage FileChooserIconView CancelButton LoadButton	N/A
FileImage	Image	id: file_image source: “”	N/A
FileChooserIconView	FileChooserIconView	id: file_chooser	on_selection: root.selected(file_chooser.selection) Pass the file path of selected image to stego.py
CancelButton	Button	Not Defined	On_release: root.dismiss()

LoadButton	Button	Not Defined	on_press: app.root.update_textfield_input() on_release: root.load_list()
ImageSaverPopup	Popup	Size: 550x500 FileImage FileChooserIconView TextInput CancelButton SaveButton	N/A
FileChooserIconView	FileChooserIconView	id: file_chooser	on_selection: new_image_name.text
NewImageName	TextInput	id: new_image_name	
CancelButton	Button	Not Defined	On_release: root.dismiss()
SaveButton	Button	Not Defined	on_release: app.root.save()

Table 5: dialog.kv – PopupDialogWidget template

Element Name	Type	Properites	Action
WarningDialog	Widget	DialogIconLabel DialogNotificationLabel DialogButtonYes DialogButtonNo	N/A
InforAndErrorDialog	Widget	DialogIconLabel DialogNotificationLabel DialogButtonOK	N/A
DialogIconLabel	Label	background-color: black size: 220x50	N/A
DialogNotificationLabel	Label	text color: red size: 220x60 position: BoxLayout default spacing: 20 padding: 5	N/A
DialogButtonYes	Button	background-color: #4CAF50 text color: white size: 110x40 position: BoxLayout default enabled by default: True	on_press: app.root.update_warning_btn_yes(True)

DialogButtonNo	Button	background-color: #4CAF50 text color: white size: 110x40 position: BoxLayout default enabled by default: True	on_press: app.root.update_warning_btn_yes(False)
DialogButtonOK	Button	background-color: #4CAF50 text color: white size: 110x40 position: BoxLayout center enabled by default: True	N/A

Table 6: models.py – ImageObject class

Attribute	**Type**	**Source**
filename	String	open_image()
max_available_chars	int	self.calculate_max_chars()
_backup_image	Image	Image.open(self.filename, 'r')
_image	Image	_backup_image.copy()
Method	**Input**	**Output**
__init__()	self, filename	ImageObject
_calculate_max_chars()	self	int
_modify_pixels()	self, data: list	tuple generator
encode_image()	self, String	Image
decode_image()	self	String
reset_image()	self	Image

Table 7: utils.py – Utility Functions

Function	**Input Parameters**	**Return Value**
convert_message()	String: user_input	list: converted_message(08b)
random_img()	None	str: random file from assets folder