Steganography Project Final Report

Linden Crandall, Jonathan Mainhart, Zhihua Zheng

University of Maryland Global Campus

CMIS 495: Current Trends and Projects in Computer Science

Prof. Majid Shaalan

May 8, 2022

Table of Contents

List of Figures

List of Tables

Overview

Steganography is the practice of hiding secret information inside of something that is not secret. There are many applications of digital steganography, but one of the most common implementations is the practice of embedding a secret message within an image file. Steganosaurus is an image-based digital steganography application written in the Python language. The application allows users to decode, read, create, and encode secret messges to and from images. The result of an encoded image is a seemingly exact replica of the orginial image imperceptibly changed to contain a secret message. The purpose of this software is to allow users to have fun and experience steganography in a user-friendly way that anyone can enjoy.

Individual Contributions:

Linden Crandall – Designed and implemented File I/O open and save filechooser, Designed and was the main contributor to the User's Guide, led completion of Phase II, provided documentation inputs and updates, performed manual and automated testing, and many bug fixes.

Jonathan Mainhart – Designed and implemented the ImageObject model, methods for encoding and decoding messages, utility functions, unit tests for Image methods and utility functions, designed and performed manual tests, implemented bug fixes, managed the GitHub repository, and edited documentation. Submitted Phase III.

Zhihua Zheng – Project Manager, designed GUI, implemented interaction between GUI elements and ImageObject model, unit tests for GUI elements, designed and performed manual tests, and provided bug fixes. General documentation input, submission of project proposal and specifications, Phase I.

Project Plan

The project plan was completed in 5 phases over 8 weeks. The phases were:

- Group Formation and Project Decision (Week 1)
- Design (Weeks 2-4)
- Implementation (Weeks 5 and 6)
- Testing and Bug Fixes (Week 7)
- Release and Retrospective (Week 8)

Each phase was tracked on a spreadsheet which listed the person responsible, planned start and end dates, actual start and end dates, and person responsible for reviewing each milestone. Our plan was not overly ambitious and allowed flexibility. We were ahead of schedule during most of the project which allowed us time to slightly refactor our code for readability.

Requirements Specification

The application is designed to run 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

The application will perform the following functions:

- 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.

User Guide

The user guide in Appendix A has been changed from the original document submitted with the following updates:

- An updated folder layout
- Addition of class objects, methods, and other variables needed to implement functionality of the application
- Removal of all references to the Delete feature
- Removal of any reference to double click an icon to launch the application
- Minor clarifications and grammar corrections

Test Plan and Results

The application was tested using the Pytest automated testing framework which is an extension of the built-in unittest module. Manual tests were completed in addition to the automated tests to ensure the application was fully operational prior to release. A detailed test plan and test results are listed in Appendix B.

Design and Alternate Designs

The project design in Appendix C has been updated from the original document submitted to account for differences in implementation than originally planned. These include:

- Updated design concept which includes references to other window class elements such as pop-up warnings and file selection windows
- An updated folder layout which includes additional assets and test files
- Updated class, method, and attribute definitions
- Minor clarifications and grammar corrections

Development History

Phase I

The focus was on foundational function implementation. A GUI was developed which can open and load images. The functionality of the encode and decode features was built but was not yet integrated into the front end. There were version control issues that were overcome, but ultimately ended this phase ahead of schedule. Specific milestones and their status are listed in Table 1 below.

Table 1: Phase I Milestone Status Table

Scope	**Milestone**	**Status**	**Notes**
Week5-6	Main GUI	incomplete	GUI Display is done. Will need to dynamically display input and output from other py. classes.
Week5-6	Popup windows	incomplete	Window Display is done. Will need to dynamically display input and output from other py. classes.
Week5-6	ImageObject Class	complete	Window Display is done. Will need to dynamically display input and output from other py. classes.
Week5	File Input	complete	Exploring a simplified method to open files
Week5-6	File Output	not started	
Week5	Message conversion	complete	
Week6	Image encoding	complete	Will need to refactor to simplify some of the code.
Week6	Image decoding	complete	
Week6	Image reset	complete	
Week6	Image save	not started	

Phase II

This phase marked the completion of preliminary testing of the software and efforts toward image file saving error handling which proved to be more complicated than originally thought. The GUI can be used to select images via filechooser and rendering it in the window. The encoding, reset, and save buttons have been implemented. The loaded image can be encoded and decoded, the maximum number of character allowance for the user's secret message can be calculated and displayed but does not update when a new image is loaded, and the image can be saved to the user's computer. The group is working on small bugs and fixes like overwriting files on save but are still ahead of schedule. Specific milestones and their status are listed in Table 2 below.

Table 2: Phase II Milestone Status Table

Scope	**Milestone**	**Status**	**Notes**
Week5-6	Main GUI	incomplete	GUI Display is done. Will need to dynamically display input and output from other py. classes.
Week5-6	Popup windows	incomplete	Still implementing popups for specific actions like saving image file to machine, resetting image to original state, etc.
Week5-6	ImageObject Class	complete	Window Display is done. Will need to dynamically display input and output from other py. classes.
Week5	File Input	complete	Successfully opens files of image type only, throws error in the form of popup dialog box if user attempts to open a non-image file
Week5-6	File Output	complete	Successfully opens files of image type only, throws error in the form of popup dialog box if user attempts to open a non-image file
Week5	Message conversion	complete	
Week6	Image encoding	complete	Will need to refactor to simplify some of the code. Button enabled
Week6	Image decoding	complete	
Week6	Image reset	complete	
Week6	Image save	incomplete	Filechooser popup implemented, image file isolated and can now successfully save to user's machine. Need to implement overwriting functionality.

Phase III

Testing, minor bug fixes, and minor improvements to the design were completed during this phase. All features have been implemented and tested. This phase ended on schedule. Specific milestones and their status are listed in Table 3 below.

Table 3: Phase III (Week 7) Milestone Status Table

Scope	**Milestone**	**Status**	**Notes**
Week5-6	Main GUI	complete	
Week5-6	Popup windows	complete	
Week5-6	ImageObject Class	complete	
Week5	File Input	complete	
Week5-6	File Output	complete	JPEG compression disabled due to message corruption when images are compressed. JPEG images can be saved with the .jpeg extension but will not be compressed.
Week5	Message conversion	complete	
Week6	Image encoding	complete	Will need to refactor to simplify some of the code.
Week6	Image decoding	complete	
Week6	Image reset	complete	
Week6	Image save	complete	
Week7	Test plan updated		
Week7	ImageObject unit tests	complete	
Week7	Message conversion utility unit test	complete	
Week7	Front end unit tests	complete	
Week7	Manual Integration tests	complete	

Final Submission

This week focused on final submission of documentation, cleaning up the GitHub repository, and minor updates to comments in the code.

Conclusion

While we did have a few minor setbacks during development, we feel this project was successful because we created working software with the features that were promised and delivered the application on time. Our application allows users to encode secret messages into images, to save them, and decode messages which use the same encoding technique with an easy to use GUI.

Lessons Learned

Our group faced and overcame many obsticles during the conception, planning, documentation, development and completion of our application. One of the most important lessons learned was ensuring that our software was compatible with and able to run natively on each of the three

major operating systems without any limitations or preference to one OS over another. One specific roadblock we ran into in the early stages of development was implementing a filechooser that worked in tandem with our GUI framework, Kivy. The initial filehooser implementation worked well on Windows, but not on macOS due to resource conflicts on the latter. We attempted to to fix this but ultimately decided to use the Kivy framework's built-in filechooser even though it prevented us from using the default file explorer that users would be most used to seeing.

Other lessons learned include:

- How to effectively collaborate and develop our application using a single git repository.
- Working through different levels of experience with the Python language and learning other new technologies during production (Pytest, Kivy, and Git).
- Working through the challenges of different time zones and different schedules.
- Having open, honest, and frequent communication ensured group members were always on schedule and working toward the same goals.

Design Strengths

Using an Oject Oriented Design ensured that the front end was effectively decoupled from the backend. The ImageObject model could easily be repurposed to create a web application or a CLI application with little or no adjustments. The use of the Kivy framework also ensures that the design of the application is able to be easily updated to refresh the color scheme or overall look and feel of the application.

Design Limitations

Design Limitations for this project include the program performance check, incomplete automated testing implementation, and deprecation warnings for a few methods used in the unit test. Considering the 8-week time limitation and the small-scale of the project, the performance check, such as the time complexity and the space complexity, were not arranged throughout the development. Based on the size of the image, there is an apparent time delay to execute the Save Image function. For instance, an image size of 24MB requires about 13 seconds of execution time. Therefore, the current program will be more suitable to work with the smaller size images.

Next, we are unable to implement automated testing for some parts of the application due to some design choices. Notably, exception testing cannot be automated for methods which handle their own exceptions instead of passing the exception to the caller. A major refactor would be required to fix this issue which may affect our ability to deliver on time. We decided to press forward with manual testing the exceptions due to time constraints.

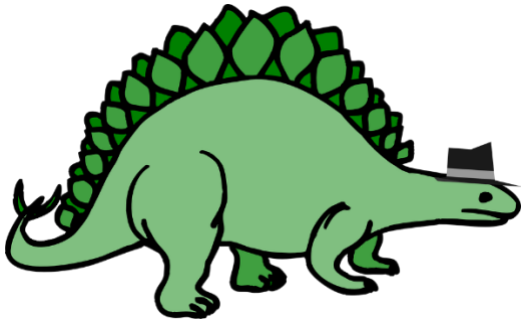
Finally, another small issue with the testing framework came up while testing on Windows systems. The framework presented depreciation warnings for a few methods used in the tests themselves. While not something that needs to be corrected right now, this is something that should be corrected before the methods are completely removed in Python 3.12.

Future Improvements

The primary focus of future improvements will be reevaluating implementation choices which impacted application performance and the ability to automate more of the tests. As previously stated, the save function is slow and might be improved upon. Other improvements to reduce code complexity and enhance human readability would also be considered.

APPENDIX A:

Steganosaurus User's Guide



About This Application

Steganography is the practice of concealing a secret message in something that is not secret. It is a simple yet powerful technique for safeguarding and transmitting secret information. ***Steganosaurus*** allows you to explore one of the fundamental concepts of steganography by hiding text inside of an image.

This software allows you to choose any image from your computer, encode that image with your own secret message, then save the encoded image which you may send to your friends or post to the social media platform of your choice. The resulting image will look identical to the original image. However, unbeknownst to the naked eye, deep within the pixels of the image will be your embedded secret message.

Steganosaurus will also allow you to retrieve and display secret messages which have been encoded into an image using the same technique.

Use ***Steganosaurus*** to:

- Send secret messages to your friends
- Protect your 5th century battle plans
- Hide your grandmother's secret cookie recipe from prying eyes (a secret worth keeping!)

System Requirements

Steganosaurus is built with Python, so it is necessary to have Python version 3.9 or higher installed on your computer prior to starting up the application. To get the latest version of Python installed onto your computer, follow the installation procedures for your operating system:

- Windows: https://www.python.org/downloads/windows/
- macOS: https://www.python.org/downloads/macos/
- Linux: https://www.python.org/downloads/source/

You can verify that the installation has been successful by opening the terminal (that's the scary screen where you control your computer with words instead of pictures) and checking the Python version.

Windows

1. Open a terminal using the windows shortcut, *Windows Key + x* (or alternatively navigating to Start -> search -> and type "Command Prompt") and then selecting "Windows Terminal" or "Command Prompt."
2. Once the terminal is opened, type `python --version` and press enter.
3. Verify that the Python version is displayed (e.g., Python 3.9.4). If the Python version is not displayed, please refer to the above Python installation link for troubleshooting/reinstallation of Python.

macOS

1. Open the Terminal application (open the Applications folder, double-click the Utilities folder, then double-click Terminal.)
2. Once the terminal is opened, type `python --version` and press enter.
3. Verify that the Python version is displayed (e.g., Python 3.9.4). If the Python version is not displayed, please refer to the above Python installation link for troubleshooting/reinstallation of Python.

Linux

Python usually comes prepackaged with your Linux distribution so you might not need to install it. To check if it installed, open a terminal window with Ctrl+Alt+T and type `python --version` and press enter.

Getting Started

Open a terminal window in the Steganosaurus directory where you installed the application and type `python3 steganosaurus` to get started.

How To

We think ***Steganosaurus*** is easy to use, but we created it so we may be a bit biased. Below you will find instructions on how to use the various features of ***Steganosaurus***.

Choose an Image

1. Click the "Open Image" button to open the file browser.
2. Navigate to the image file that you want to use for your steganography. Make sure that the file you select is of a proper image file type.

NOTE: Only image files with .jpg, .jpeg, or .png are allowed. An error will display if you attempt to load any other file type.

Your image will display in the window when successfully loaded.

Decode an Image

The application will attempt to decode a stored message as soon as an image is loaded. The decoded message will be displayed in the text window below the image. Nothing will be displayed if the image selected does not contain a secret message.

NOTE: You may see a garbled message which does not make sense due to no message being encoded. This is completely okay.

Encode an Image

1. Open an image file (see Choose an Image above).
2. Enter your secret message in the text field. The number of characters remaining along with the number of characters allowed will be displayed.

NOTE: The length of the allowed message will be limited by the size of the selected image.

3. Click the "Encode Image" button when you are ready to encode the message into the image.

Congratulations! You have successfully encoded an image using steganography!

Reset an Image

If you change your mind, you can start over by removing an encoded message from an image before it is saved.

NOTE: Resetting an image can only be performed on newly encoded images. Saved images cannot be reset.

1. Click the "Reset Image" button.

When selected, any secret messages that you have encoded onto the image will be erased and the image will be restored to its unaltered original form from the time it was opened.

Save an Image

You may save your image after successfully encoding a message.
1. Click the "Save Image" button. The default File Explorer will open.
2. Navigate to the directory where your encoded image is to be saved.

NOTE: You may rename and save the image immediately after opening it before you begin the encoding process to make a copy and preserve the original image file.

Once saved, you can send your secret message as an attachment to an email, text message, or social medial post like any other image.

APPENDIX B:

Test Plan and Results

Introduction

This appendix contains test cases used to measure functionality of the application. The tests were run on four different systems listed in Table 4. Automated unit test cases for the backend (application logic) and frontend (user interface) are contained in Table 5 and Table 6 respectively. Manual integration tests are listed in Table 7.

Descriptions and screen shots of the results of each test begin on page 21.

Test Systems

The application was tested on four separate systems listed in Table 4. These systems represent the expected typical application user systems. Each system has Python version 3.9 installed. A Linux system was unavailable at the time of testing.

Table 4
Test Systems

System ID	OS (Version)	Processor	RAM
S1	macOS (12.3.1)	3.8 GHz 8-Core Intel i7	8 GB 2667 MHz DDR4
S2	macOS (12.1)	1.6 GHz Dual-Core Intel i5	8 GB 2133 MHz LPDDR3
S3	Windows 11 Home	2.9 GHz Intel Core i7	16GB 2933 MHz DDR4
S4	Windows 11	3.0 GHz Intel Core i7	16GB 3200 MHz DDR4

Test Cases

Table 5
Automated Unit Tests (Back End)

#	Test Input	Expected	Pass/Fail				Fig. #
			S1	S2	S3	S4	
1	check max chars of 300x300 pix img	30,000	Pass	Pass	Pass	Pass	1
2	check max chars of 5x5 pix img	8	Pass	Pass	Pass	Pass	1
3	encode 5x5 pix img to max chars without truncation	‘01234567’	Pass	Pass	Pass	Pass	1
4	encode 5x5 pix image to max chars with overflow input ‘01234567ABCDEF’	‘01234567’	Pass	Pass	Pass	Pass	1
5	image and backup image attributes are exact copies	Attributes match	Pass	Pass	Pass	Pass	1

6	image encode changes pixels to not match backup image	Attributes do not match	Pass	Pass	Pass	Pass	1
7	check decoded message matches encoded message	Messages match	Pass	Pass	Pass	Pass	1
8	check the image can save to disk as a new file	new file saved	Pass	Pass	Pass	Pass	1
9	check the saved image decoded message matches the encoded message	Messages match	Pass	Pass	Pass	Pass	1
10	check that encoded image pixel values match original values after reset	Images reset	Pass	Pass	Pass	Pass	1
11	check convert message utility return value matches binary representation of test message	Values match	Pass	Pass	Pass	Pass	1

Table 6
Automated Unit Tests (Front End)

#	Test Input	Expected	Pass/Fail				Fig. #
			S1	S2	S3	S4	
1	on_open_button_click	ImageObject matches.	Pass	Pass	Pass	Pass	1
2	on_encode_button_click . class variable: enable_bool = True.	• Object display_image's method encode_image is called once with default text from TextField. • update_widgets_status is called with the arguments: (False, True, False).	Pass	Pass	Pass	Pass	1
3	on_encode_button_click . class variable: enable_bool = False.	• Method popip_user_notification is called with the arguments: ('Failed to execute encode function!	Pass	Pass	Pass	Pass	1

		\nPlease modify the text field input.', MainWidget.MESSAGE_TYPE.ERROR) •					
4	on_reset_button_click	• Variable warning_type maches MainWidget.WARNING_TYPE.RESET • Method popup_user_notification is called with the arguments: (‘Are you sure you want to reset the image?', MainWidget.MESSAGE_TYPE.WARNING)	Pass	Pass	Pass	Pass	1
5	execute_reset	• Object display_image’s method reset_image is called once.	Pass	Pass	Pass	Pass	1
6	on_save_button_clic	• Variable new_filename maches the String “expected”. • ImageChooserPopup Class method show_filechooser is called once.	Pass	Pass	Pass	Pass	1
7	on_save setup: new_filename is valid. overwrite with new_filename.	• Method popup_user_notification is called with the arguments: (‘Image name already exists. \nAre you sure you want to	Pass	Pass	Pass	Pass	1

		overwrite the image?', MainWidget.MESSAGE_TYPE.WARNING)					
8	execute_save setup: decode_image method return string "decoded_msg" new_filename = 'test_image_3.jpeg' new_filepath = 'path'	• Object display_image's method save_image is called once with arguments: ('path', 'test_image_3.jpeg') • Variable title maches the String "Steganosaurus – test_image_3.jpeg". • main_image.source maches the String "path/ test_image_3.jpeg". • Variable textfield_str matches "decoded_msg".	Pass	Pass	Pass	Pass	1
9	validate_image_name setup: image_name = 'test_image_3.jpeg'	• Method validate_image_name returns True. •	Pass	Pass	Pass	Pass	1
10	validate_image_name setup: image_name = '_test_image_3.jpeg'	• Method validate_image_name returns False. • Method popup_user_notification is called with the arguments: ('Invalid file name! \n Only alphabet characters, numbers, dot,	Pass	Pass	Pass	Pass	1

		underscore and hyphens are allowed. (e.g. image_1)', MainWidget.MESSAGE_TYPE.ERROR)					
11	validate_image_name setup: image_name = '_test_image_3.jpeg'	• Method validate_image_name returns False. • Method popup_user_notification is called with the arguments: ('Invalid file name! \n Only alphabet characters, numbers, dot, underscore and hyphens are allowed. (e.g. image_1)', MainWidget.MESSAGE_TYPE.ERROR)	Pass	Pass	Pass	Pass	1
12	update_warning_btn_yes setup: warning_btn_yes = True self.warning_type == self.WARNING_TYPE.WARNINGSAVE	• Method update_textfield_input is called once. • Method execute_reset is called once. • Method update_textfield_input • is called once with arguments: (True, False, False)	Pass	Pass	Pass	Pass	1
13	update_warning_btn_yes setup:	• Method execute_reset is called once.	Pass	Pass	Pass	Pass	1

	warning_btn_yes = True self.warning_type = self.WARNING_TYPE.RESET	• Method update_textfield_input • is called once with arguments: (True, False, True)					
14	update_warning_btn_yes setup: warning_btn_yes = False self.warning_type = self.WARNING_TYPE.RESET	• Method update_textfield_input is called once with arguments: (False, True, False) •	Pass	Pass	Pass	Pass	1
15	update_widgets_status setup: reset_btn_disabled = False textfield_disabled = True image_saver_dismiss = True	• Variable value reset_btn_disabled matches • Variable value textfield_disabled matches • Variable value image_saver_dismissmatches •	Pass	Pass	Pass	Pass	1
16	update_main_widgets setup: main_text_field.text = " display_image.max_available_chars = 100	• Variable value main_image.source matches • Variable value textfield_str matches • Variable value maximum_char_count matches • Variable value encodable_bool matches	Pass	Pass	Pass	Pass	1
17	update_main_widgets setup: display_image.decode_image returns “decoded_msg” display_image.max_available_chars = 11	• Variable value main_image.source matches • Variable value textfield_str matches	Pass	Pass	Pass	Pass	1

		• Variable value maximum_char_count matches • Variable value user_notification_msg matches • Variable value encodable_bool matches					
18	update_main_widgets setup: display_image.decode_image returns “decoded_msg” display_image.max_available_chars = 10	• Variable value main_image.source matches • Variable value textfield_str matches • Variable value maximum_char_count matches • Variable value user_notification_msg matches • Variable value encodable_bool matches	Pass	Pass	Pass	Pass	1
19	update_textfield_input setup: textfield_str ="somestring"	• Variable value main_text_field.text matches •	Pass	Pass	Pass	Pass	1

Table 7
Manual Tests

#	Test Input	Expected	Pass/Fail				Fig. #
			S1	S2	S3	S4	
1	Randomly selected image loads on launch	Image displays	Pass	Pass	Pass	Pass	2
2	Image is decoded when loaded	Decoded message displays	Pass	Pass	Pass	Pass	2
3	Open image button opens file chooser	File chooser displays	Pass	Pass	Pass	Pass	3
4	File chooser has read access to user file system	Able to navigate file system	Pass	Pass	Pass	Pass	3
5	Attempt to open non-image file	File chooser rejects non-image files	Pass	Pass	Pass	Pass	4
6	Cancel button in the File chooser view is pressed	Return to the main GUI	Pass	Pass	Pass	Pass	NA
7	Attempt to open image file	Image opens and is displayed	Pass	Pass	Pass	Pass	5
8	Attempt to enter test phrase into text input area	Text is displayed. Allowed character count decreases	Pass	Pass	Pass	Pass	6
9	Attempt to enter text greater than maximum allowed for the image	Warning message displays in the Main GUI.	Pass	Pass	Pass	Pass	7
10	Attempt to enter text greater than maximum allowed for the image, then encode image button is pressed.	Error dialog popups. Not encodable.	Pass	Pass	Pass	Pass	8
11	Enter valid characters in the textfield, then encode image button pressed	Text input is disabled Reset button is enabled	Pass	Pass	Pass	Pass	9
12	Save image button pressed	Save file dialog opens	Pass	Pass	Pass	Pass	10
13	Attempt to save file with invalid file name “123 ” which end with a space.	Error dialog popups.	Pass	Pass	Pass	Pass	11
14	Attempt to save file with incorrect extension “123.JNG”	New image is saved with the file name “123.JNG.png”. Successfully saved image dialog popups.	Pass	Pass	Pass	Pass	12

15	Attempt to save file as JPEG “123.jpeg”	New image is saved with filename “123.jpeg”	Pass	Pass	Pass	Pass	13
16	Save image button pressed. Select Cancel button in save dialog	Returned to main screen	Pass	Pass	Pass	Pass	NA
17	Overwrite existing filename	Warning pop up. Allows overwrite	Pass	Pass	Pass	Pass	14 15
18	Reset image after encoding. Then Yes button in the warning popup is pressed.	Warning pop ups. Image resets to original value	Pass	Pass	Pass	Pass	16 17
19	Reset image after encoding. No button in the warning popup is pressed.	Returned to main GUI.	Pass	Pass	Pass	Pass	NA
20	Open image containing message to decode	Decoded message displays in text area. File name is displayed in the title bar. Image is displayed in the main window.	Pass	Pass	Pass	Pass	18

Results

The automated tests were run on systems S1 – S4. All tests passed on every system. Figure 1 shows the results of the automated tests run on S2.

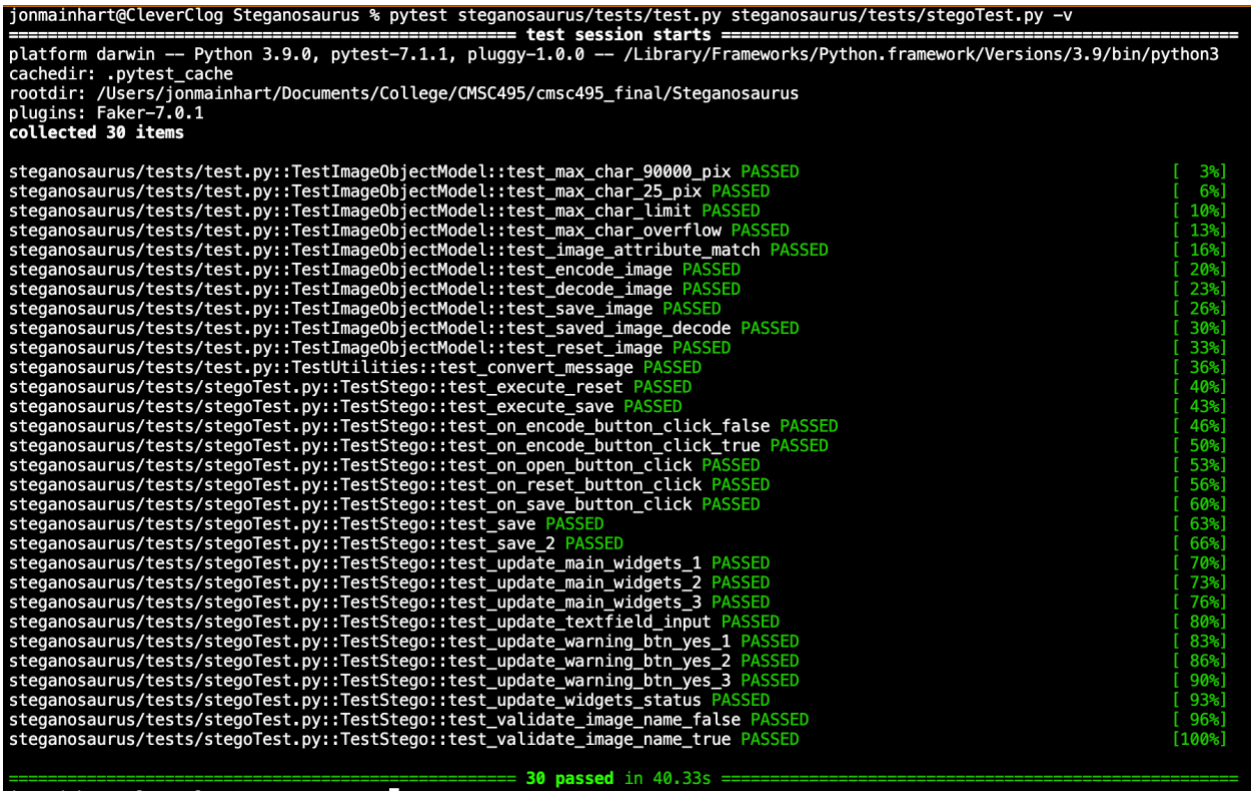
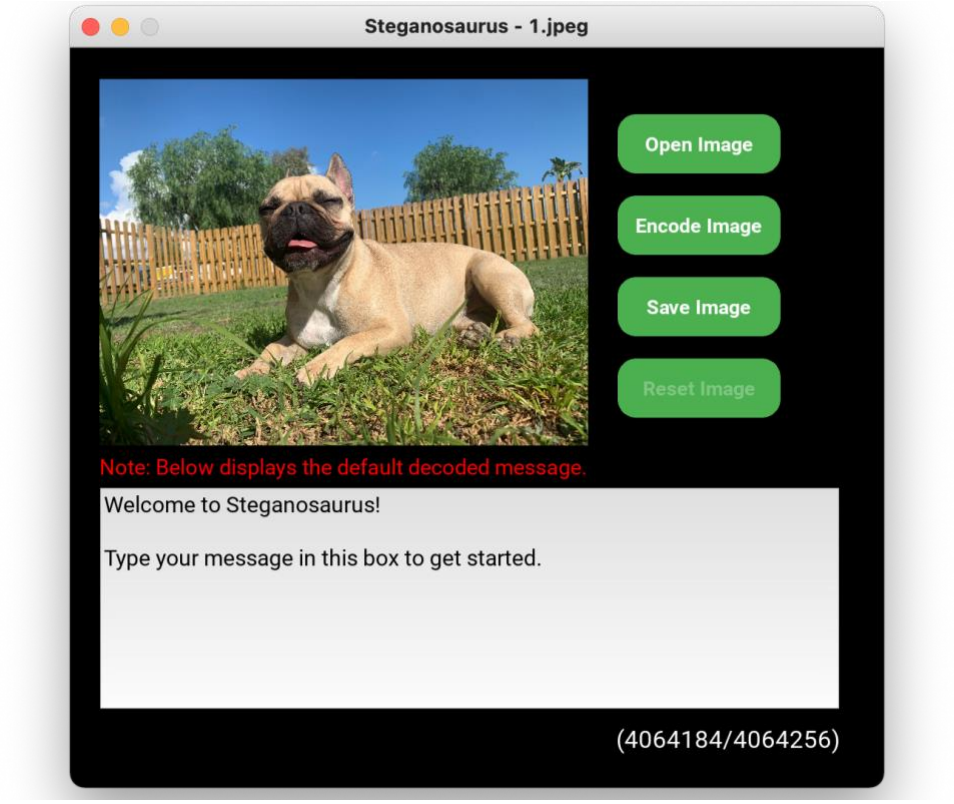


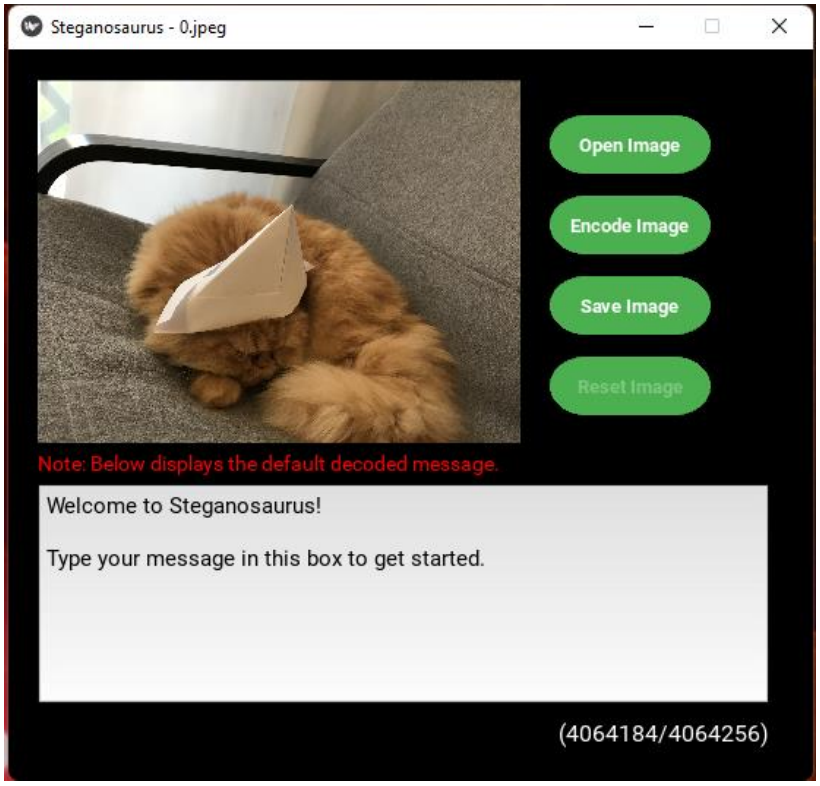
Figure 1. System 2 automated unit test results.

The 30 automated tests cover ImageObject methods used to encode, decode, calculate maximum message size, save, and reset images to their original state; binary encoding of messages; and front-end logic.

The manual testing of the application was successfully performed on all test systems. These tests included verifying that all buttons and fields worked as expected, that any expected user interaction worked as required, and that any unexpected user input was handled properly to prevent the application from crashing or operating outside of design parameters. The results of the tests contained in Table 7 begin with Figure 2.

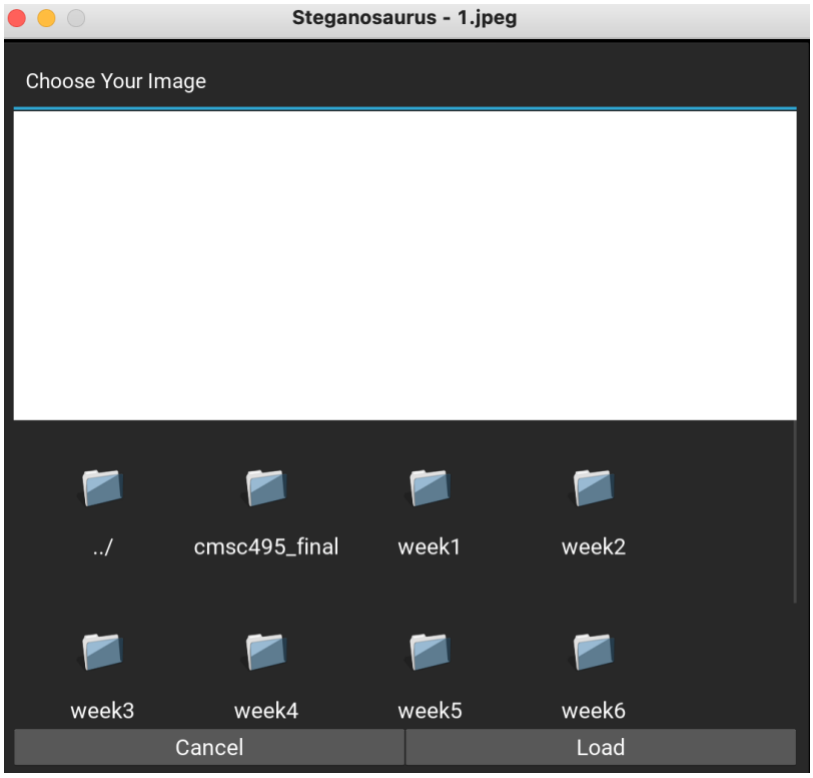


macOS

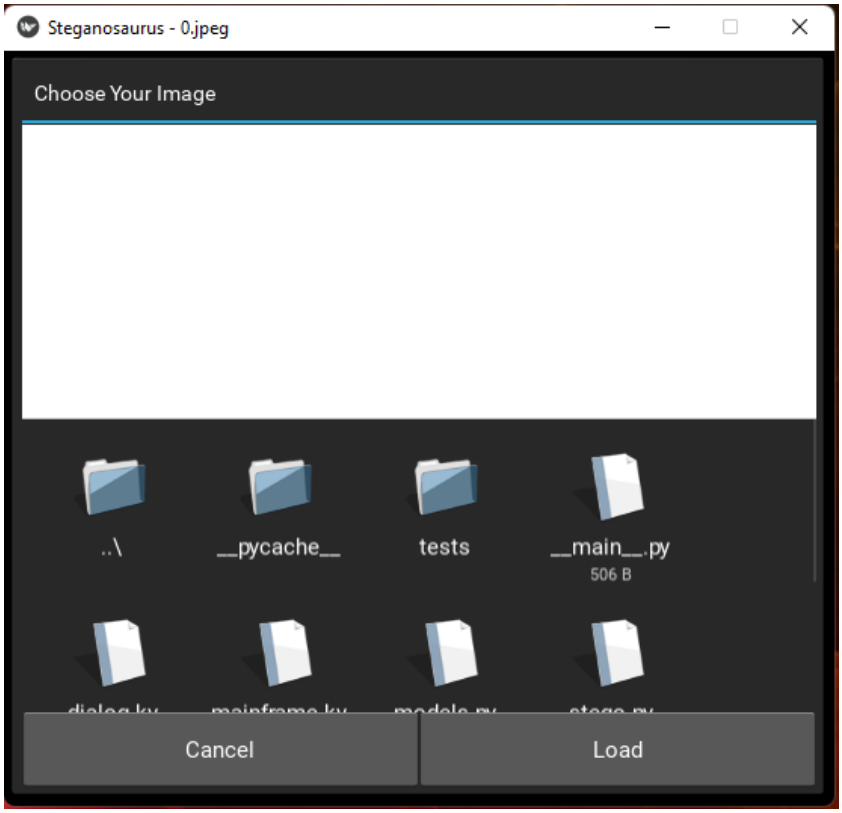


Windows

Figure 2. The application loads a randomized image when launched and decodes the secret message (manual test cases 4-1 and 4-2).

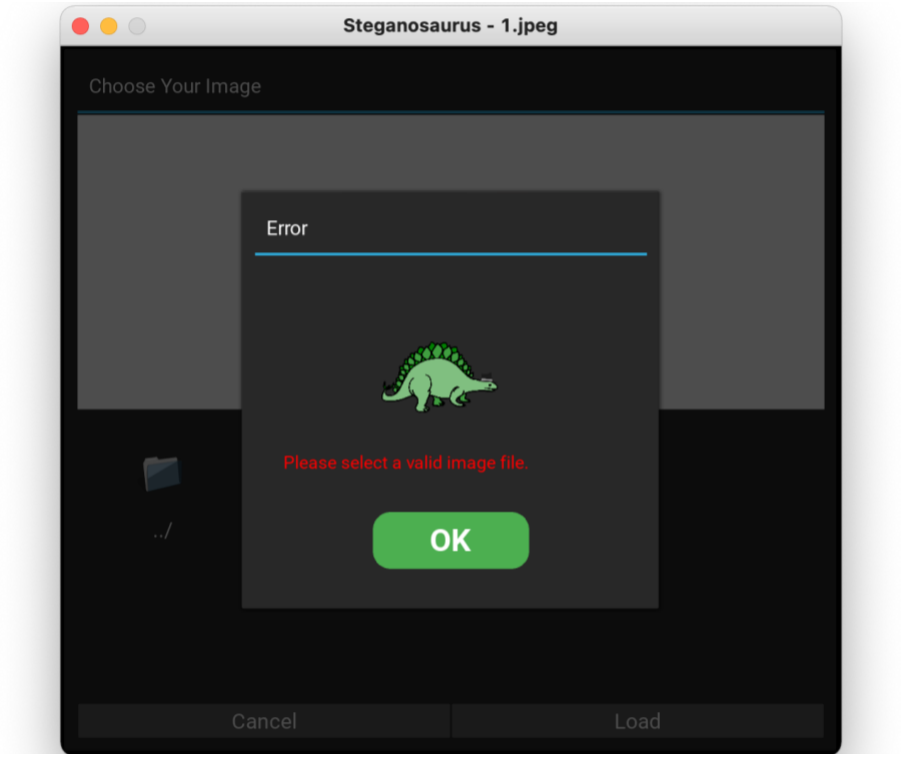


macOS

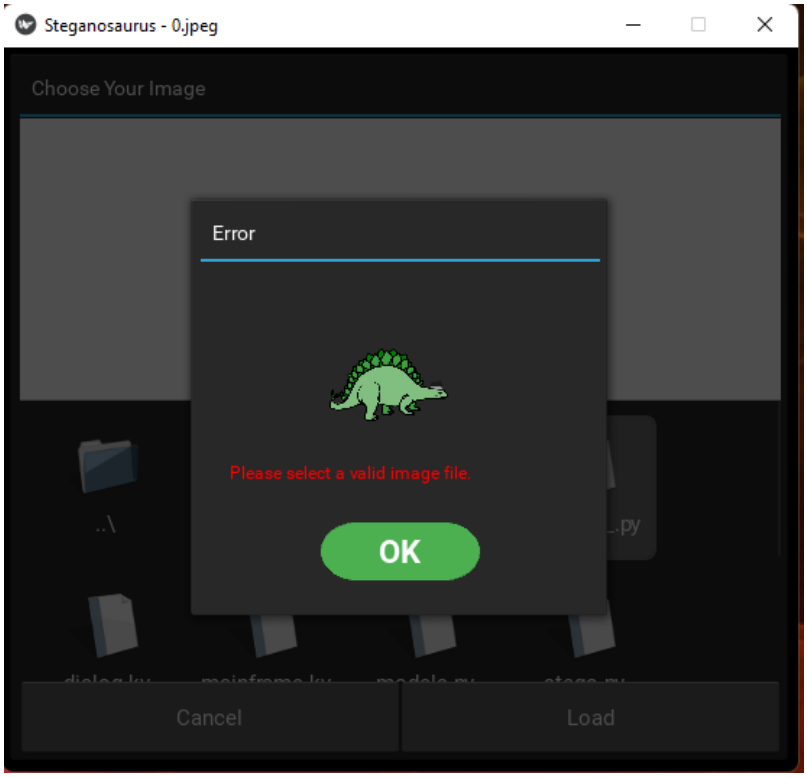


Windows

Figure 3. Results of clicking the “Open Image” button. The application has access to the file system (manual test cases 4-3 and 4-4).

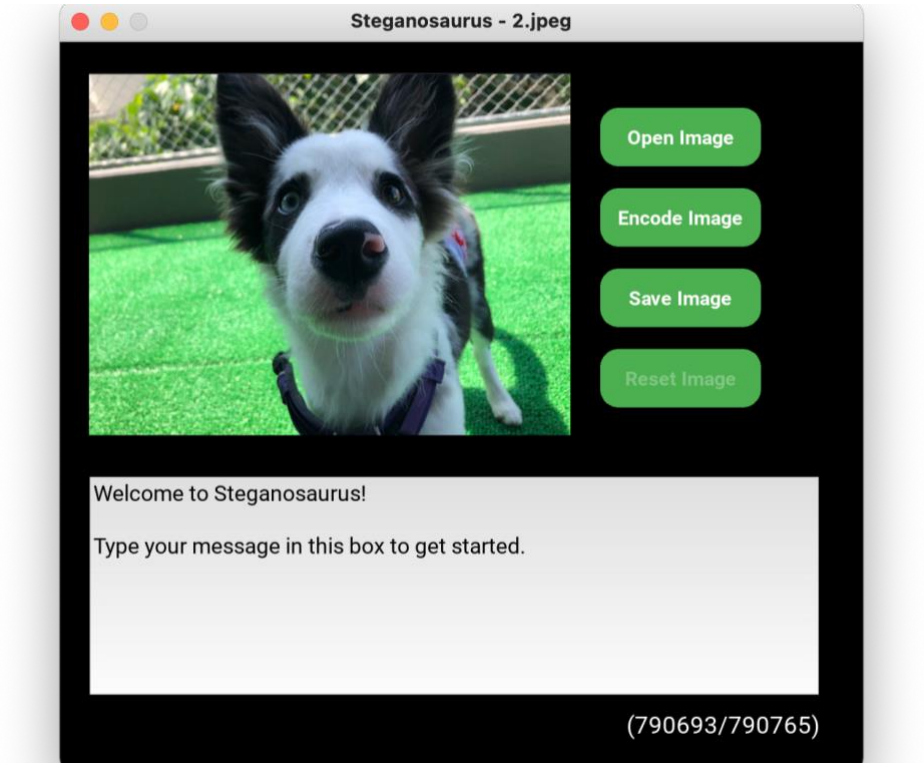


macOS

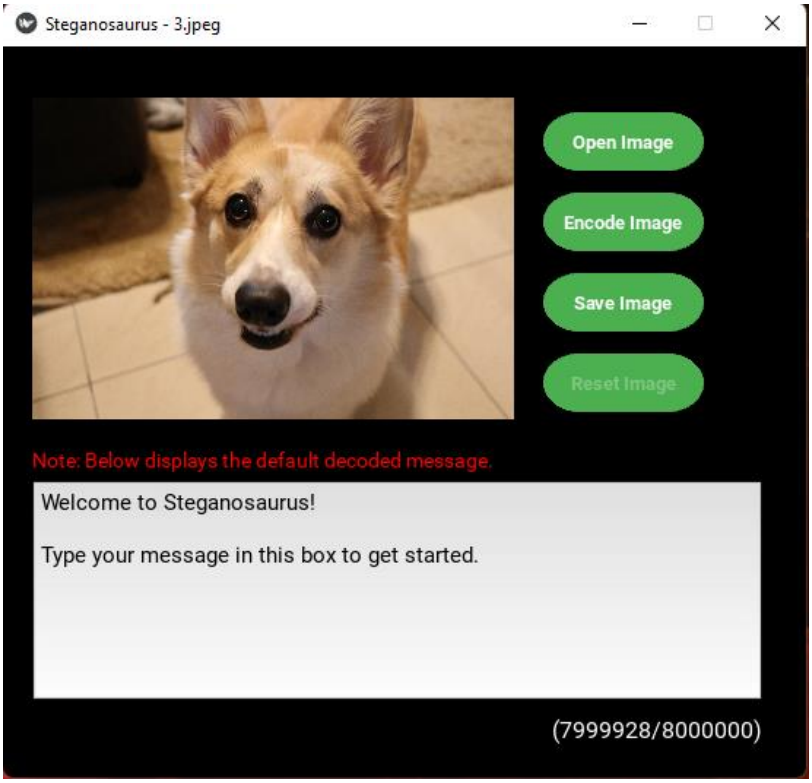


Windows

Figure 4. Results of attempting to open a ineligible file (Test case 4-5).

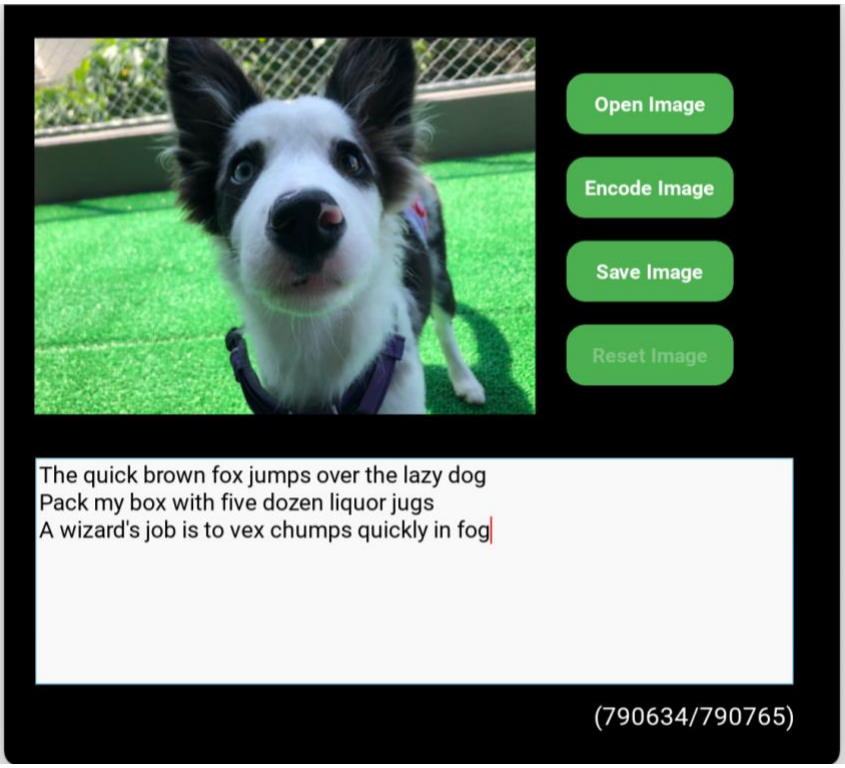


macOS

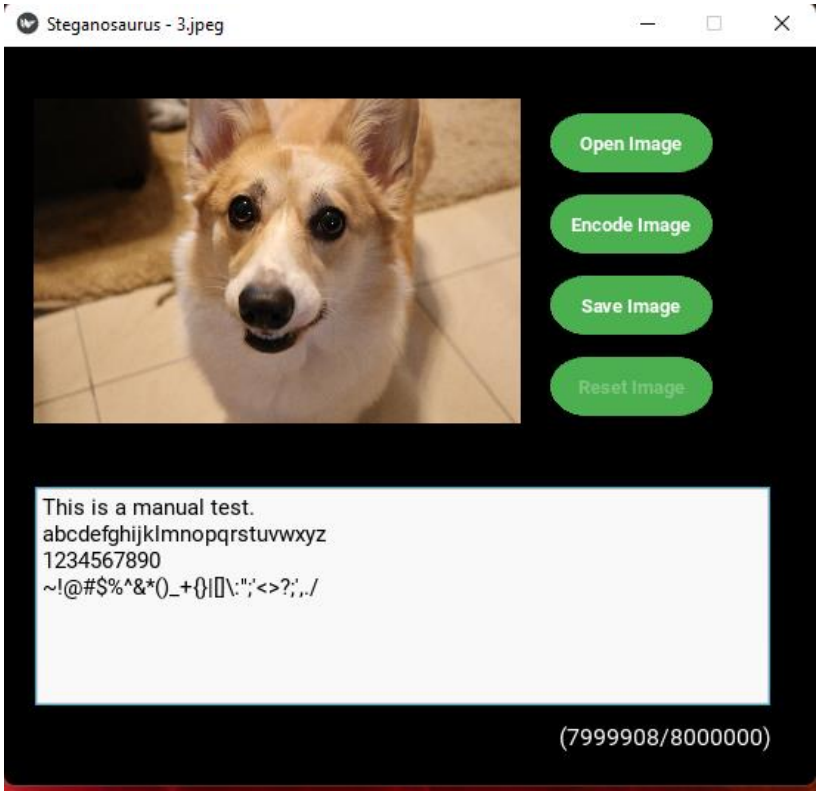


Windows

Figure 5. The application opens image files as expected (Test case 4-6).

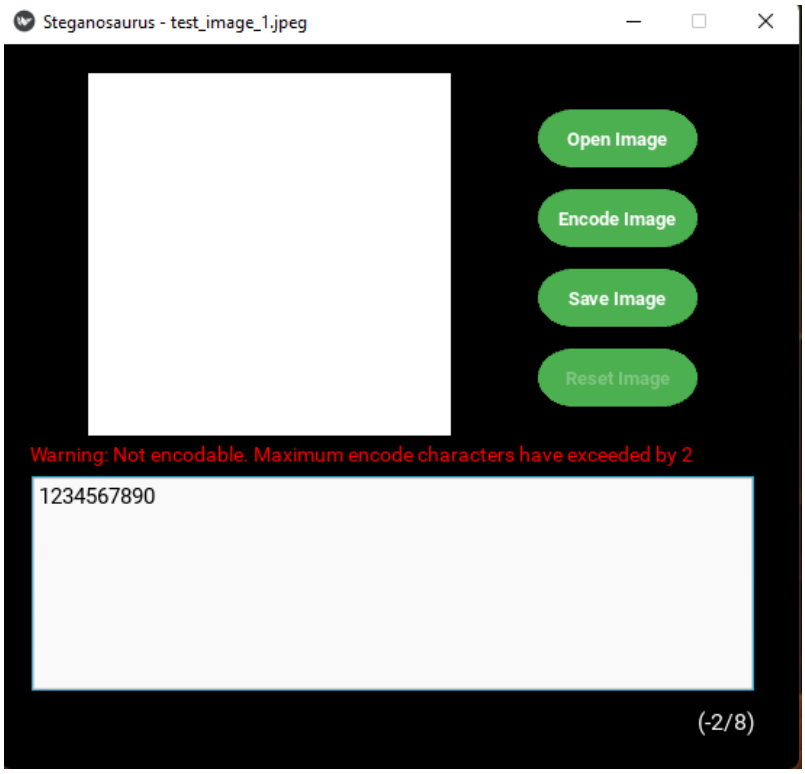


macOS



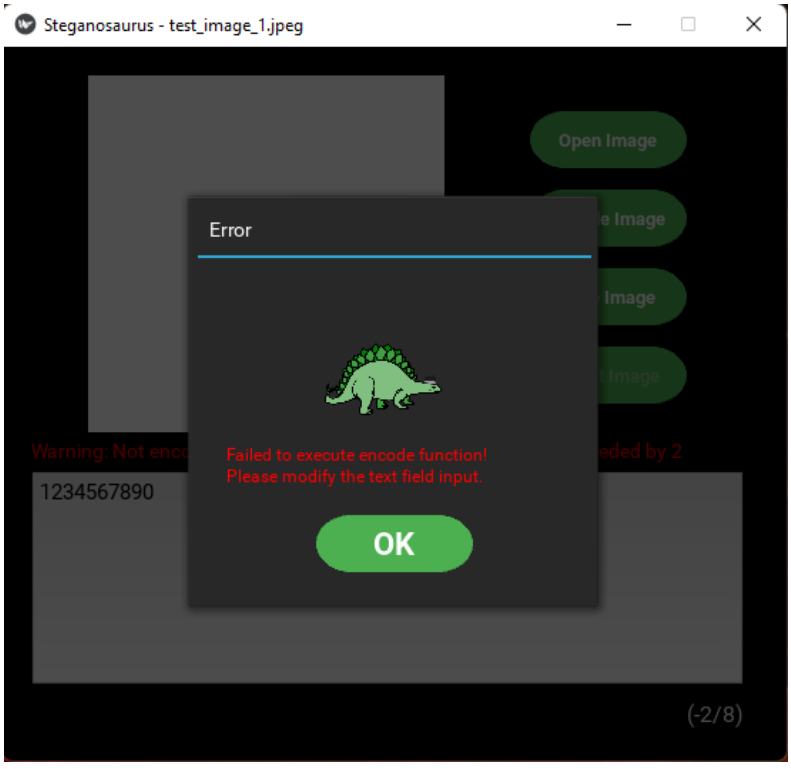
Windows

Figure 6. The remaining characters allowed decrease as text is input (Test case 4-8).



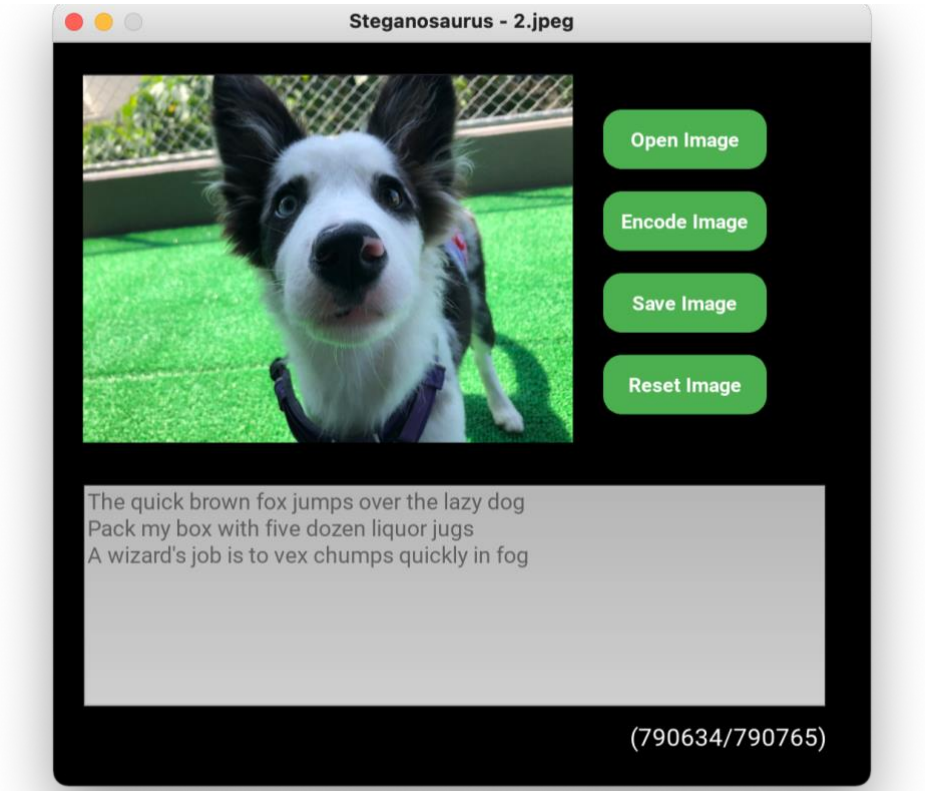
Windows

Figure 7. Entering too many characters displays a warning (Test case 4-9).

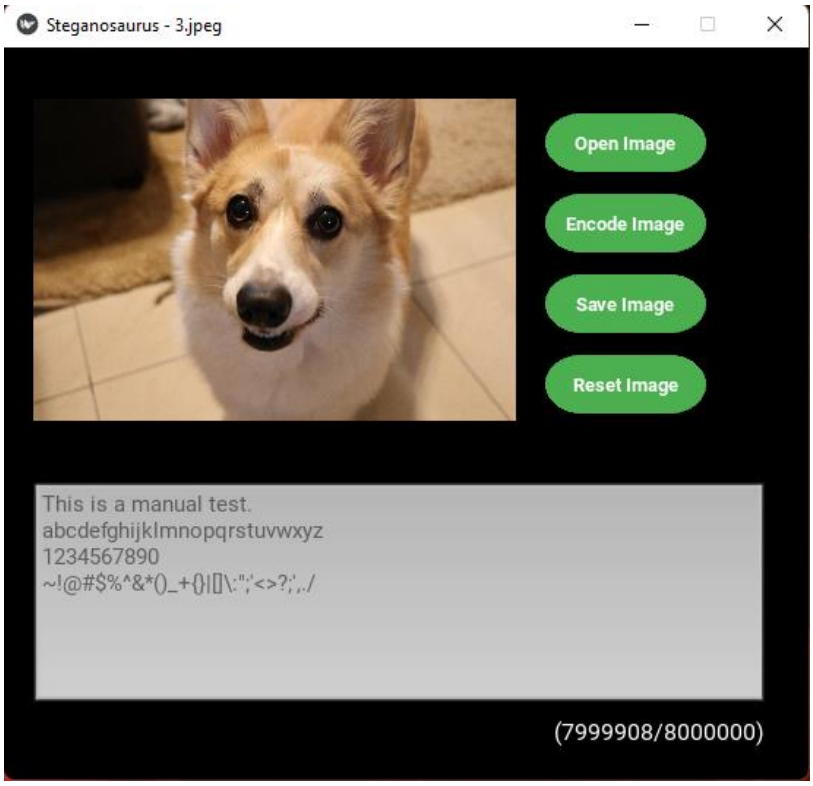


Windows

Figure 8. Attempting to encode an image with too many characters (Test case 4-10).

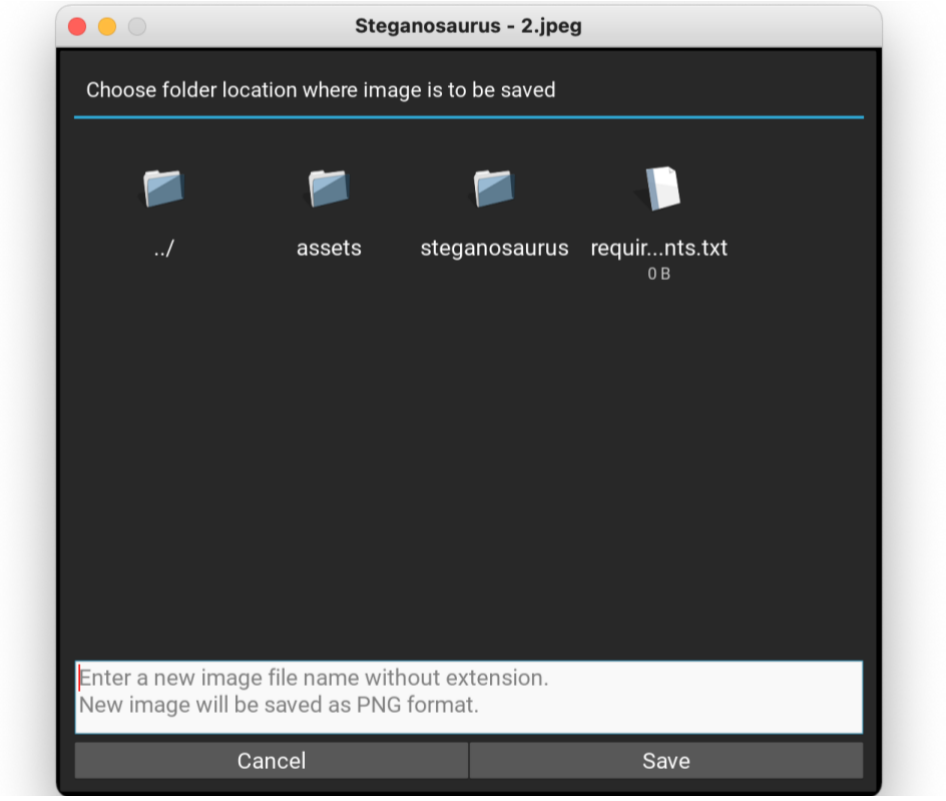


macOS

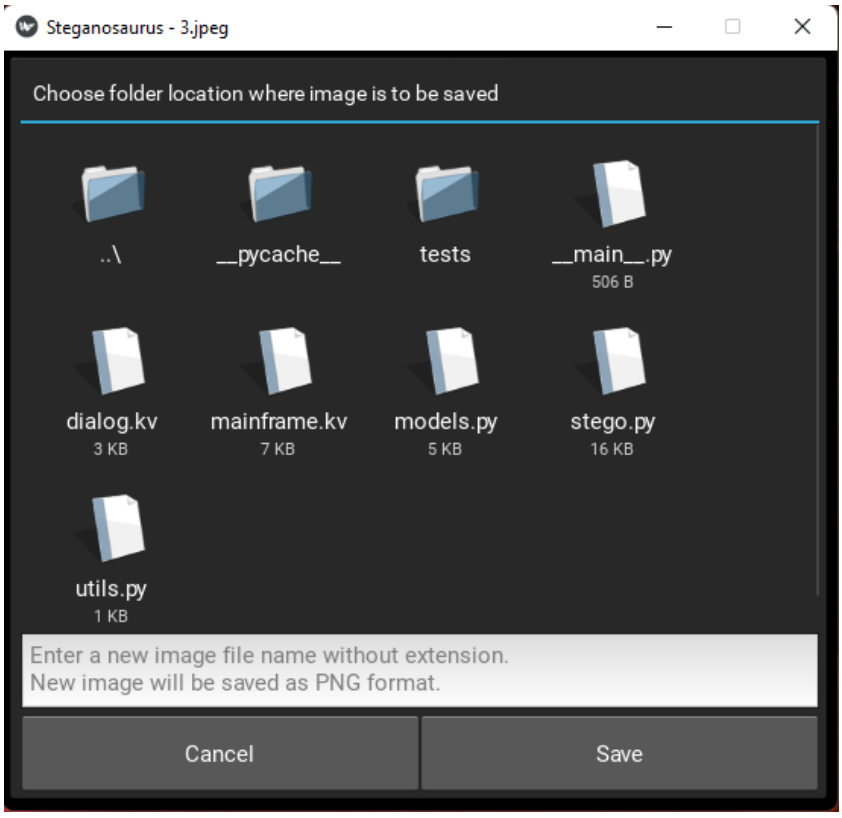


Windows

Figure 9. A message of the correct length will encode (Test case 4-11).

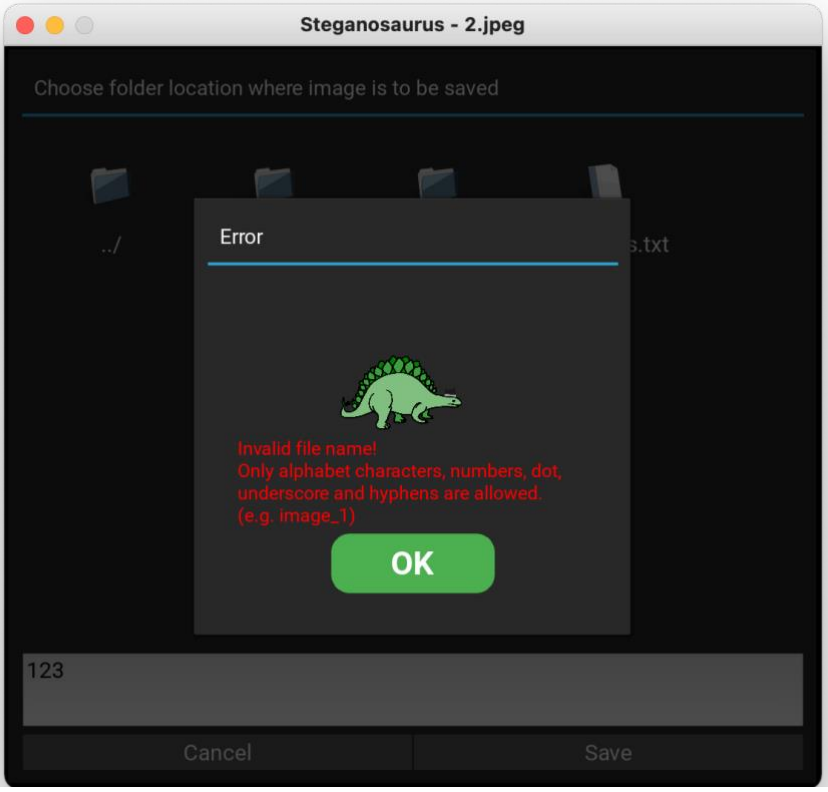


macOS

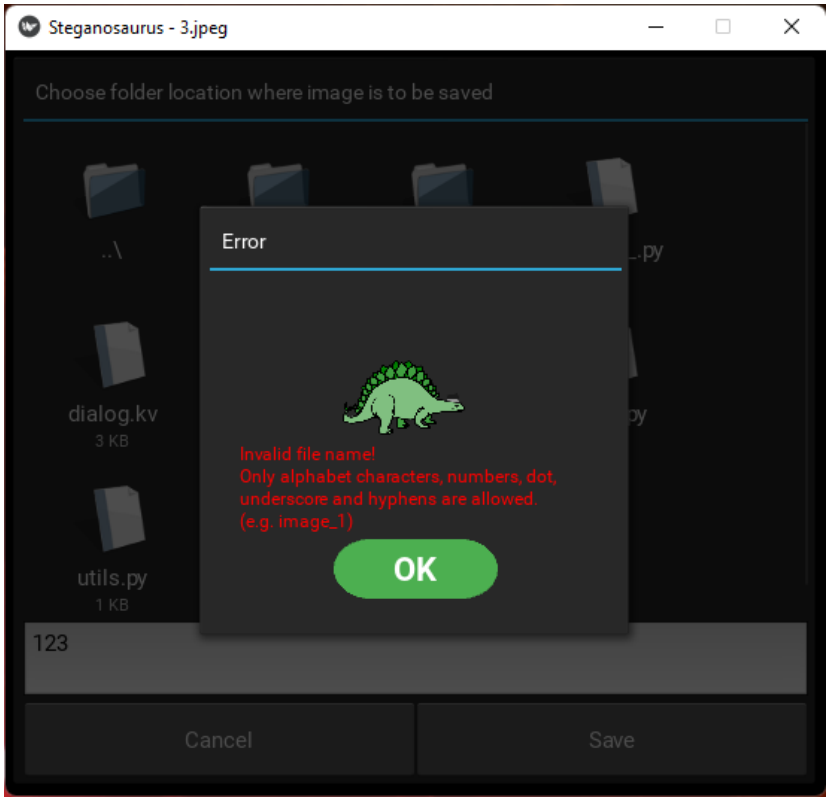


Windows

Figure 10. Save dialog (Test case 4-12).

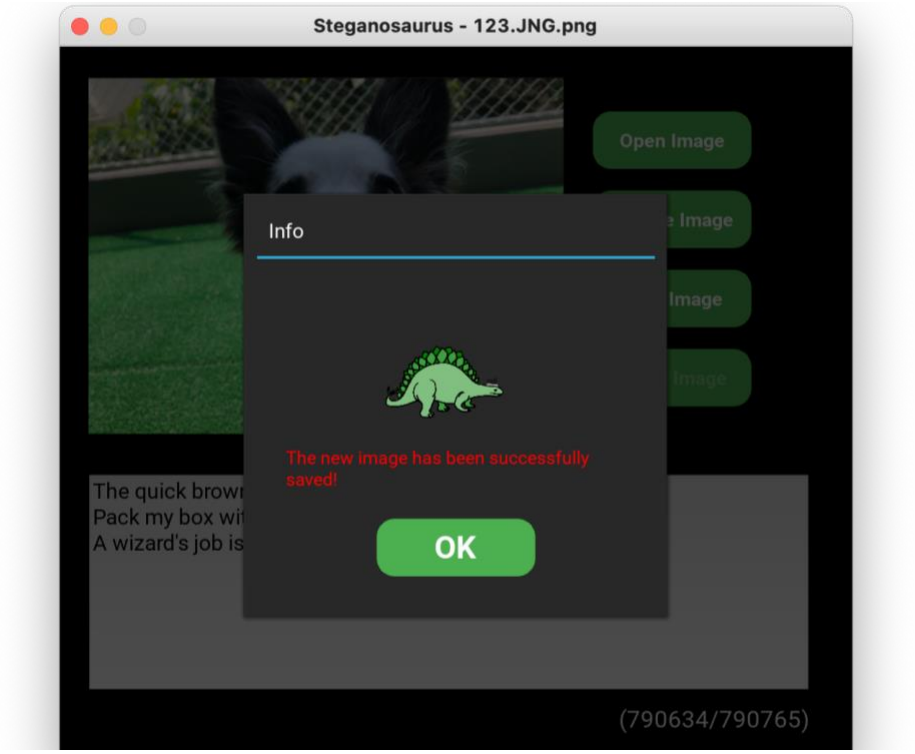


macOS

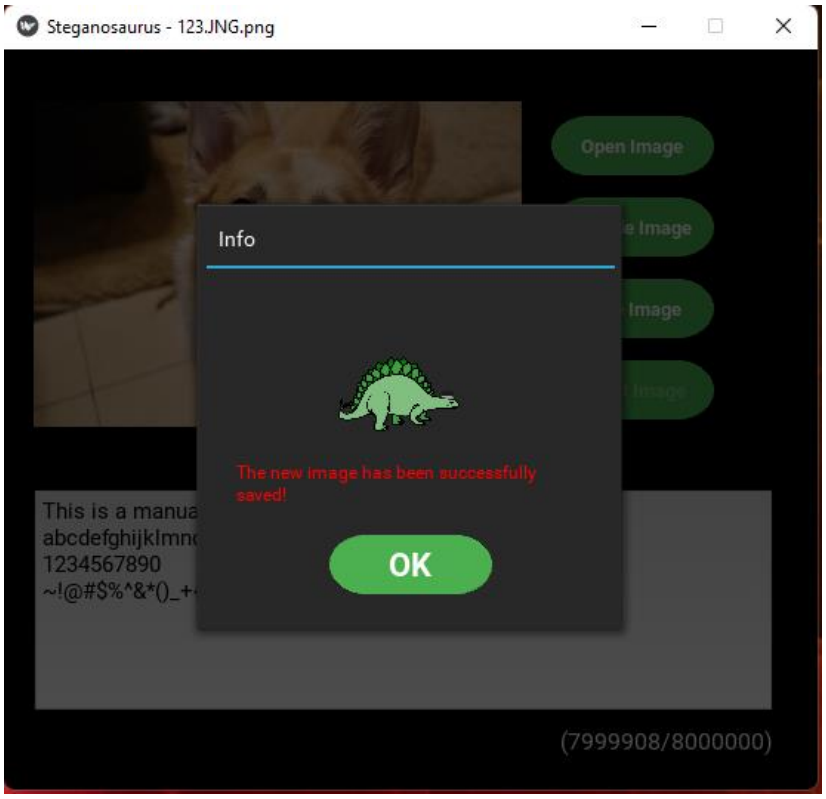


Windows

Figure 11. Attempting to save a file with an invalid name (Test case 4-13).

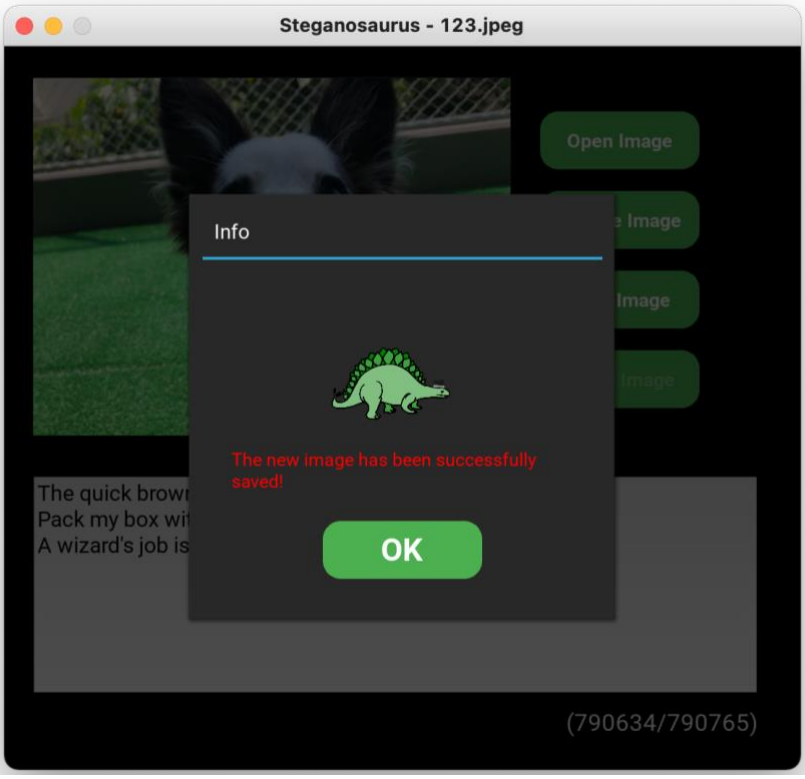


macOS

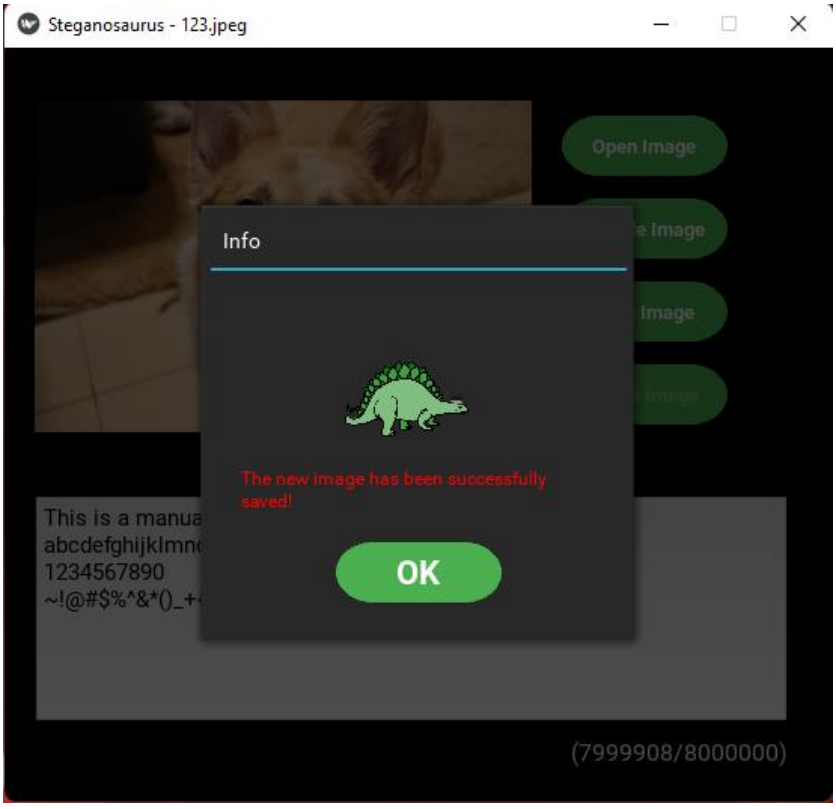


Windows

Figure 12. Successfully saving an image with default encoding (Test case 4-14).

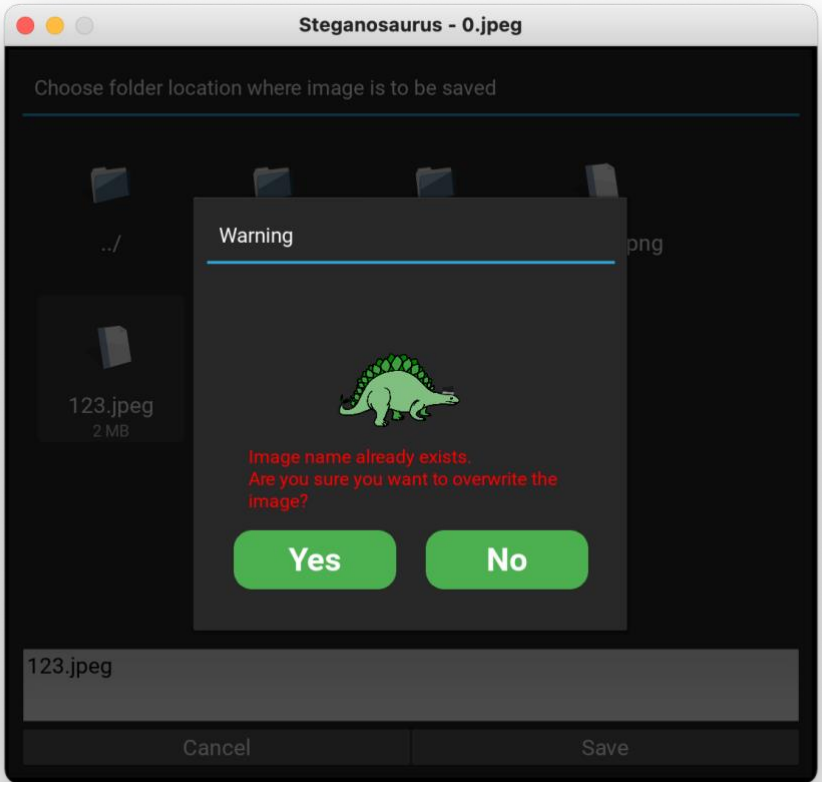


macOS

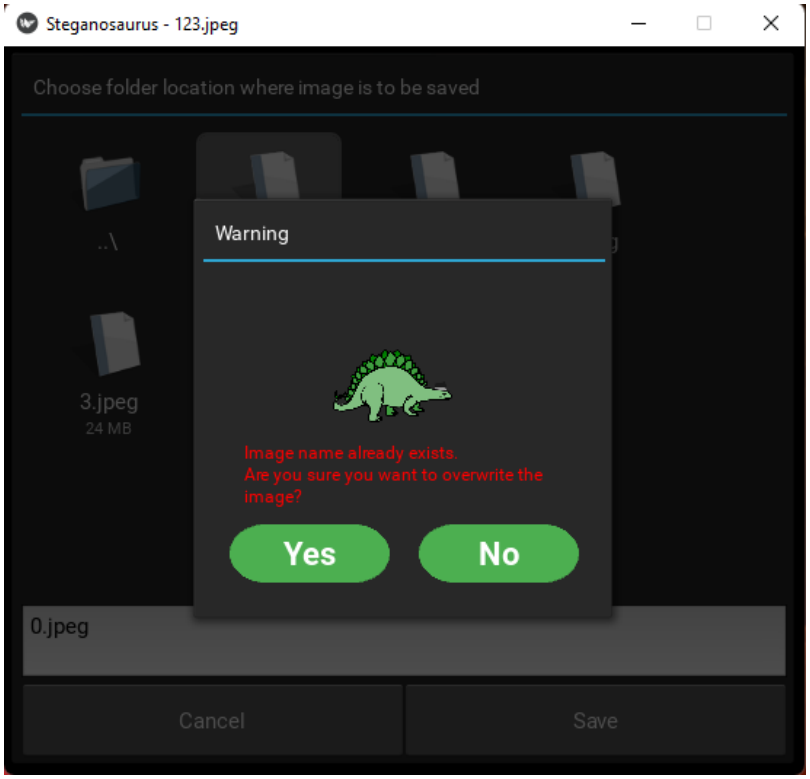


Windows

Figure 13. Successfully saving an image as JPEG (Test case 4-15).

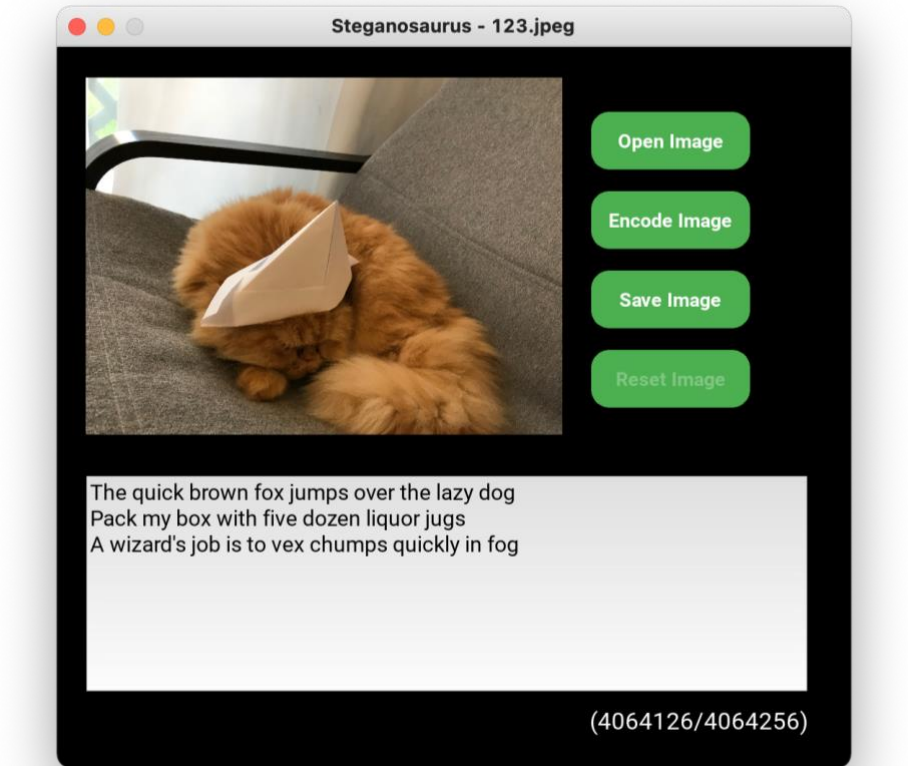


macOS

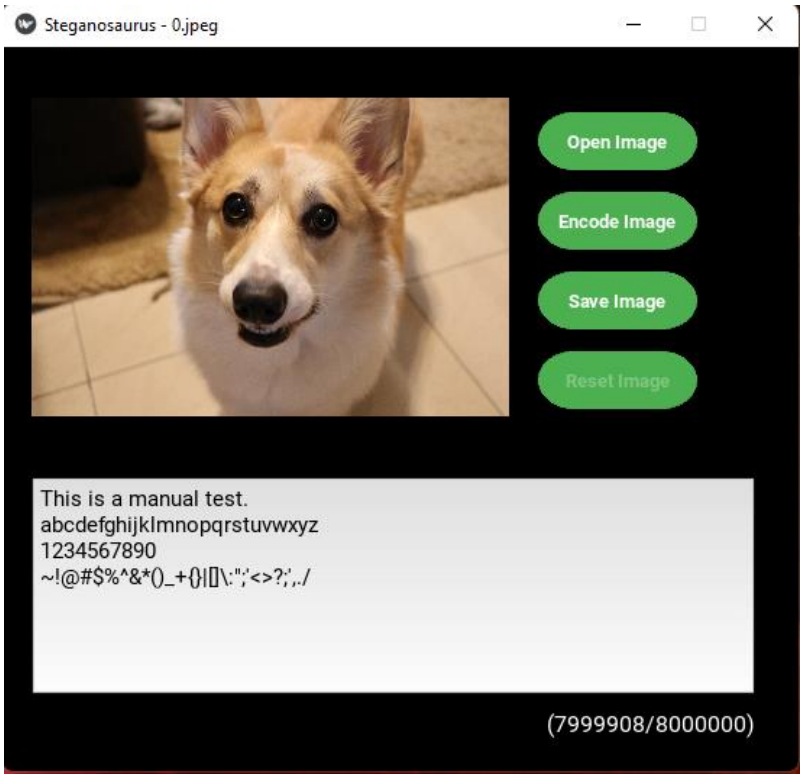


Windows

Figure 14. Attempting overwrite triggers warning (Test case 4-17).

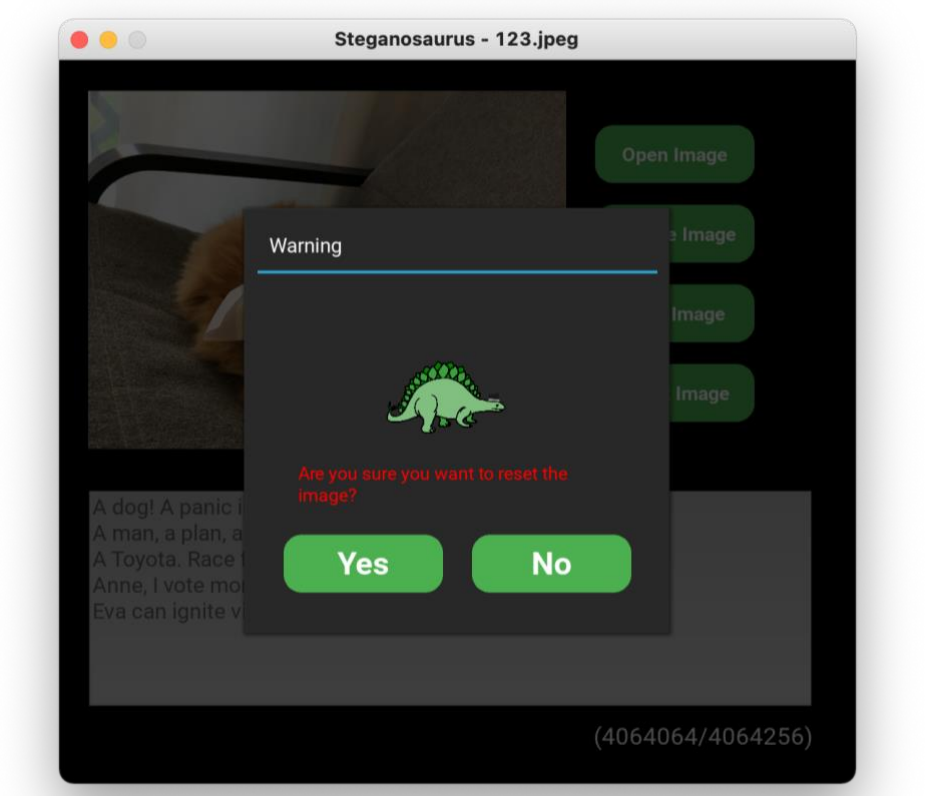


macOS

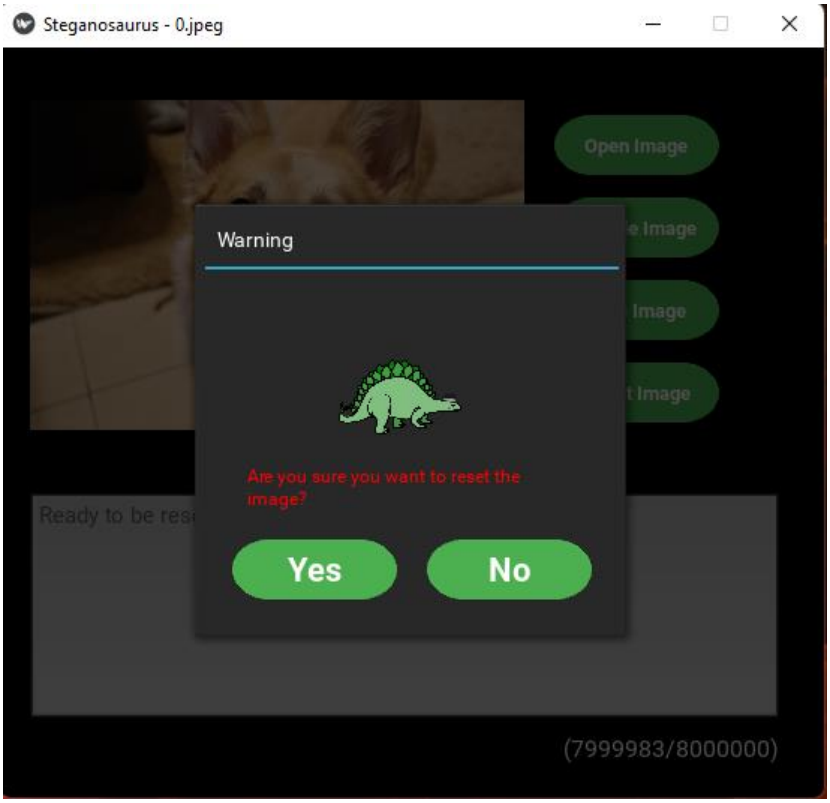


Windows

Figure 15. The files have been overwritten (Test case 4-17).

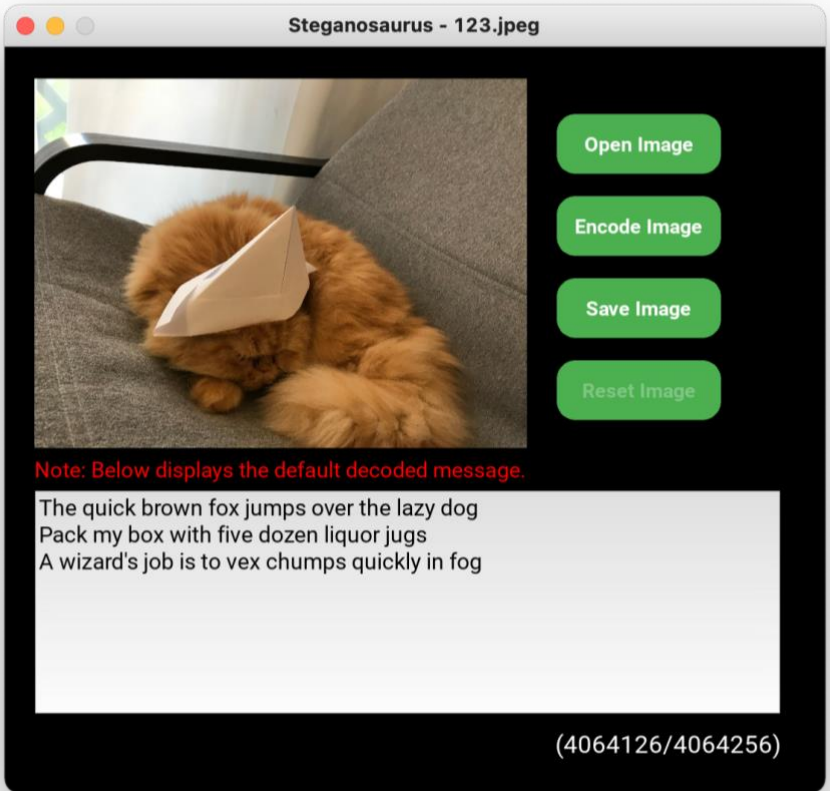


macOS

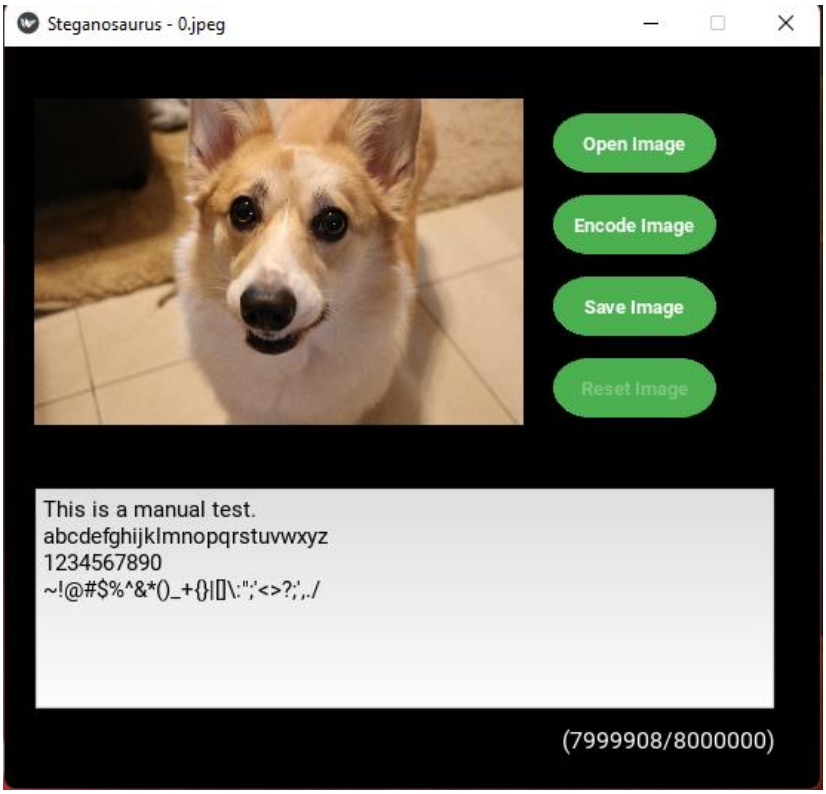


Windows

Figure 16. Reset button triggers warning (Test case 4-18).

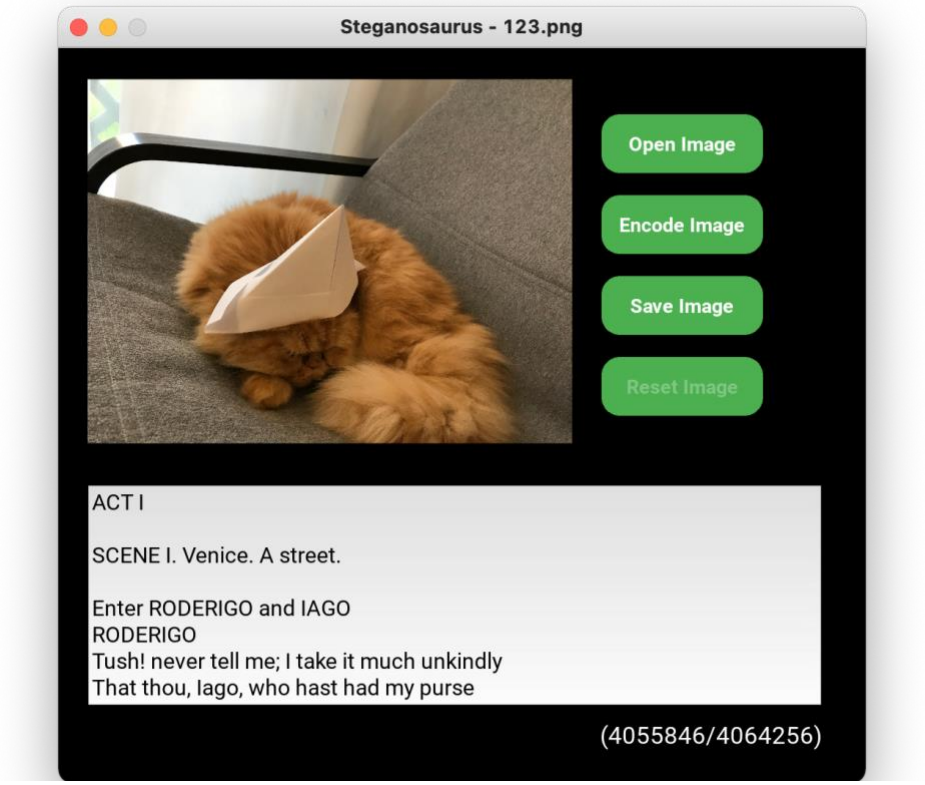


macOS

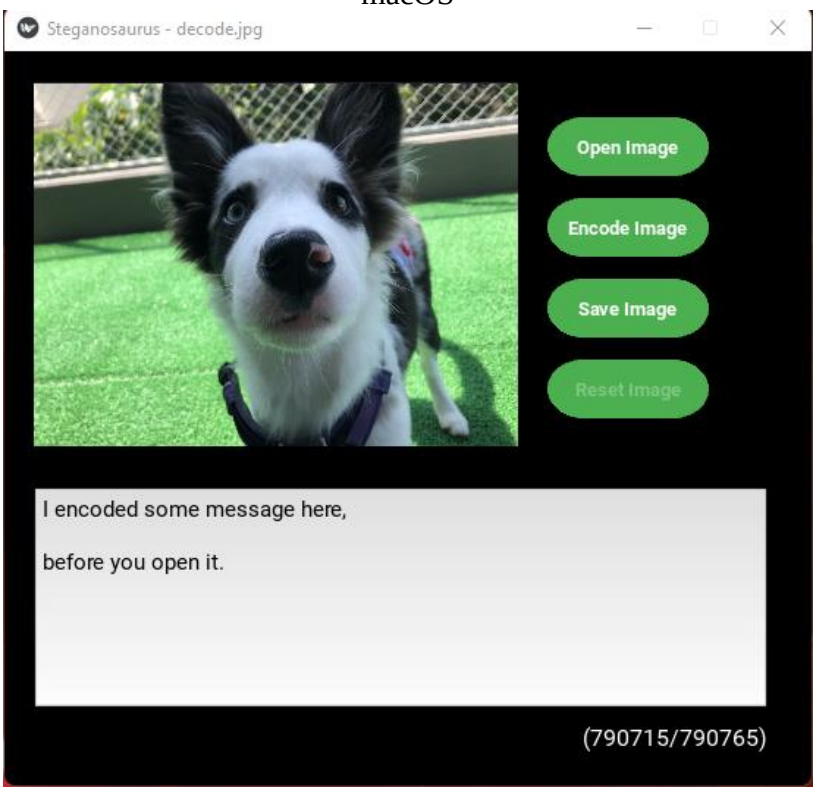


Windows

Figure 17. Image resets (Test case 4-18).



macOS



Windows

Figure 18. Image is decoded when opened (Test case 4-20).

APPENDIX C:

Project Design

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 19.
Figure 19. File structure

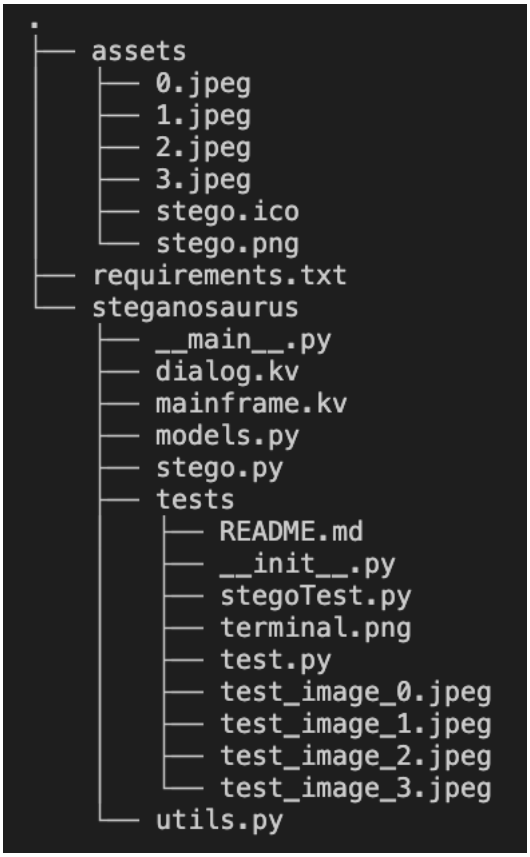


Figure 19. File structure

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, Image, 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 20 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.

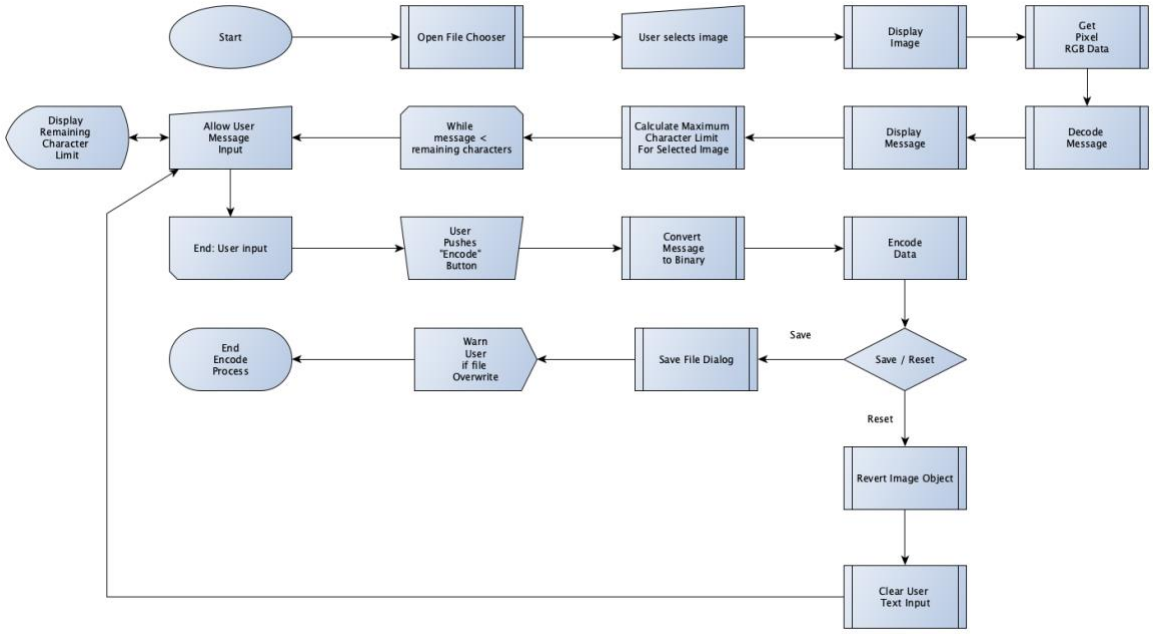


Figure 20. Logic Control Flow of a typical user interaction

User Interface

GUI Overview

The Graphical User Interface (GUI) of the application shown in Figure 21 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.

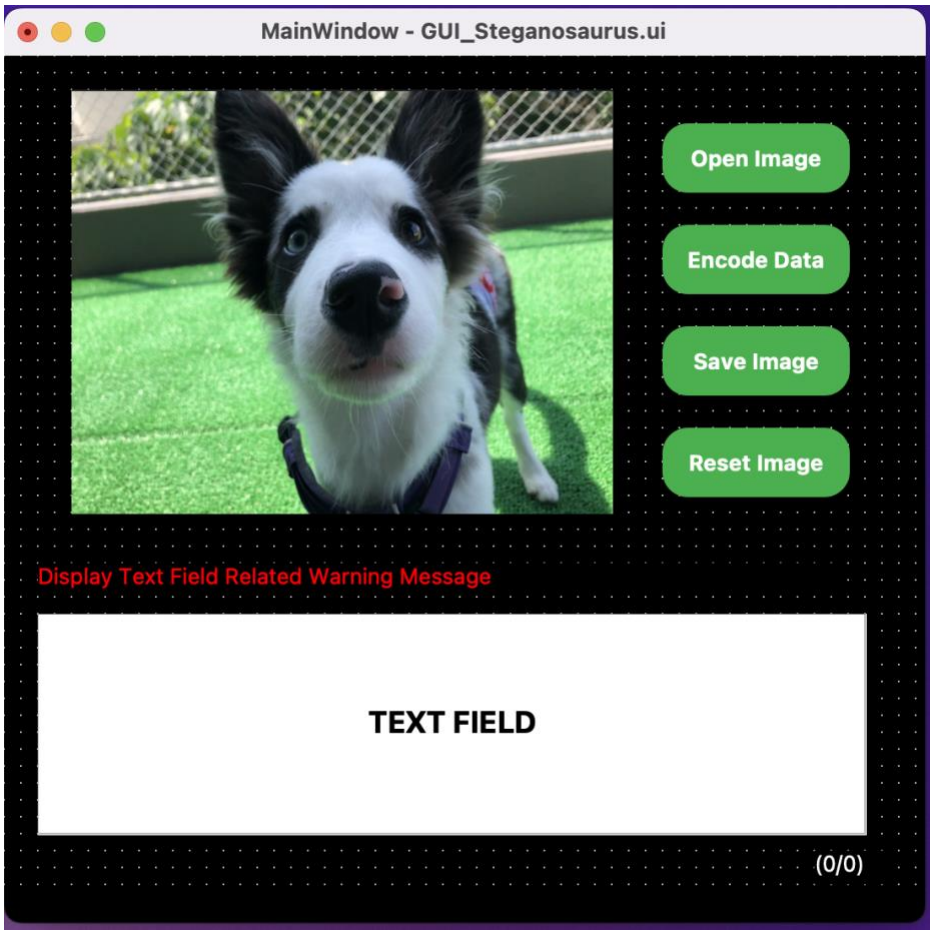


Figure 21. Graphical User Interface

User Notification Dialog

User messages resulting from errors and runtime exceptions will be displayed in a popup dialog as shown in Figure 22 and Figure 23. 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.

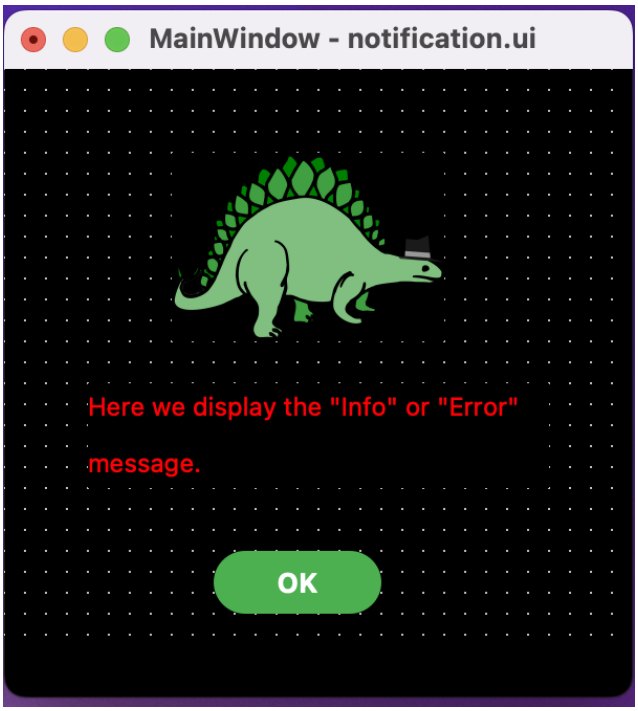


Figure 22. Info and error message popup dialog

Figure 23. 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 8**:** 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 9: 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 10: 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 11: 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 12: 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 13: 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 14: 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