# Vigenere Visualization Tool
# Programmer's Documentation

Joshua Fawcett, Max Hopkins, Meghan Riehl, Yuyao Zhuge
May 30, 2021

This document should help facilitate programmers in making additional modifications and improvements to the Vigenere Visualization tool. We will give an overview of:

1. System Files
2. Procedures/Class Structures
3. User Interaction and Interface
4. Possible Improvements

The Vigenere Visualization tool is designed to allow teachers/professors to pedagogically visualize the steps and results of encrypting and decrypting using the Vigenere cipher. It also has menus which give the user information on how to use the tool, and what ciphers are/how they work. The visualization animation can be sped up, slowed down, or paused, allowing the user to step through each part of the cipher encryption/decryption process.

**System Files:**

***All files that will be discussed here can be found in the Main_Program directory, as files in the Development_Files may not be up to date, and are not required by the system.***

**main.py:** This is the main file for the system, and is where the main game loop is being run. In a traditional game loop, it will take input from the user, render items onto the screen, and update once each frame. Here, we are calling the functions Input(), Render(), and Update() each frame, depending on the scene that we are on, which do these actions for us. It also initializes the pygame window. This is the file that should be executed when developing to get the system to start.

**Cipher.py:** This file defines the Encrypt() and Decrypt() functions that are called when the user submits a plaintext message and a key, depending on whether they click the Encrypt or Decrypt buttons. These functions take the plaintext and key as input, and encrypt/decrypt using the Vigenere Cipher. It will also record all steps for the cipher into a tuple. Once finished, the functions will return the result, the original plaintext and key, and the steps.

**SceneManager.py:** This file houses all the scenes in the system inside of classes, and defines an abstract base class SceneManager(). These classes will be discussed in the class structures section.
**button_class.py:** This file defines the button() class, which is used by the system to render buttons into the scene, create the text on the button, and define the function that should be called when the button is pushed.

**TextHighlight.py:** This file draws the message, keyword, and result to the visualization scene and implements functionality for highlighting specific letters in the message, keyword, and result by defining a Text_and_Highlight() class.

**table.py:** This file draws a Vigenere table and implements functionality for highlighting specific rows, columns, and cells in the table by defining a Table() class.

**Procedures:**

This system is run by calling main.py, which only imports the classes in SceneManager.py in order to switch which scenes we are calling Input(), Render(), and Update() on while running the game loop. It will load the screen with our starting scene StartScene().

The menus that load when each button is pressed will be described more in the class structures section, but it is clear to see when using the system which buttons do what.

Once the user is on the input menu scene (MainMenu()), they will be able to put in a message and a key. When the user selects to either encrypt or decrypt, it will call the respective function from Cipher.py, which sends the return values to the visualization scene (ButtonScene()). From here, ButtonScene() will call from table.py to load the table along with the highlights for the column, row, and result letter. ButtonScene() will also call from TextHighlight.py to load the hypertext highlights for the message, key, and result.

**Class Structures:**

**SceneManager():** SceneManager is the base class used by all scenes in the system. It defines 5 functions:

1. Input(): This function is called to detect input from the user and update the scenes/switch to a new scene depending on the input.
2. Render(): This function is called to display all objects onto the scene (buttons, text, etc.). Objects are formatted onto the screen based on screen size, which is stored in the width and height variables.

3. Update(): This function is called to update the width and height variables when the screen size changes, which reformats all objects all the scene the next time Render() is called.
4. SwitchToScene(): This function is called once to switch the active scene being displayed on the screen by changing the variable “self.scene” in the class from the current scene to the next scene, which then gets updated in main.py.
5. Terminate(): Calls SwitchToScene and passes through None to terminate the scene. Is not used in the program.

All classes which derive from SceneManager will use Input(), Render(), Update(), and SwitchToScene. The classes using SceneManager will be best described through visuals. Code for the objects on the scene can be discovered through the code comments/documentation.

**StartMenu():**


**Notes:**

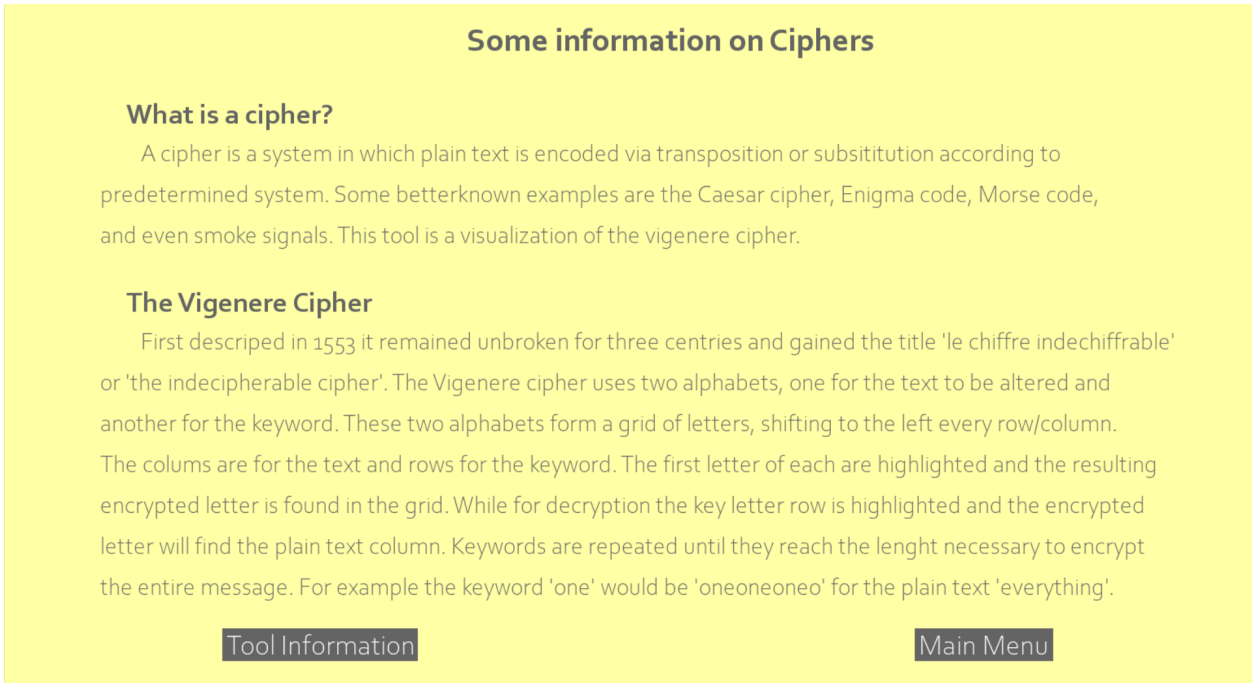
- First scene to load in the system.
- Input() will detect mouse clicks.

**InfoMenu():**


**Notes:**

- Displayed when “Info” button in StartMenu() is pressed.
- Input() will detect mouse clicks

**About():**



**Notes:**

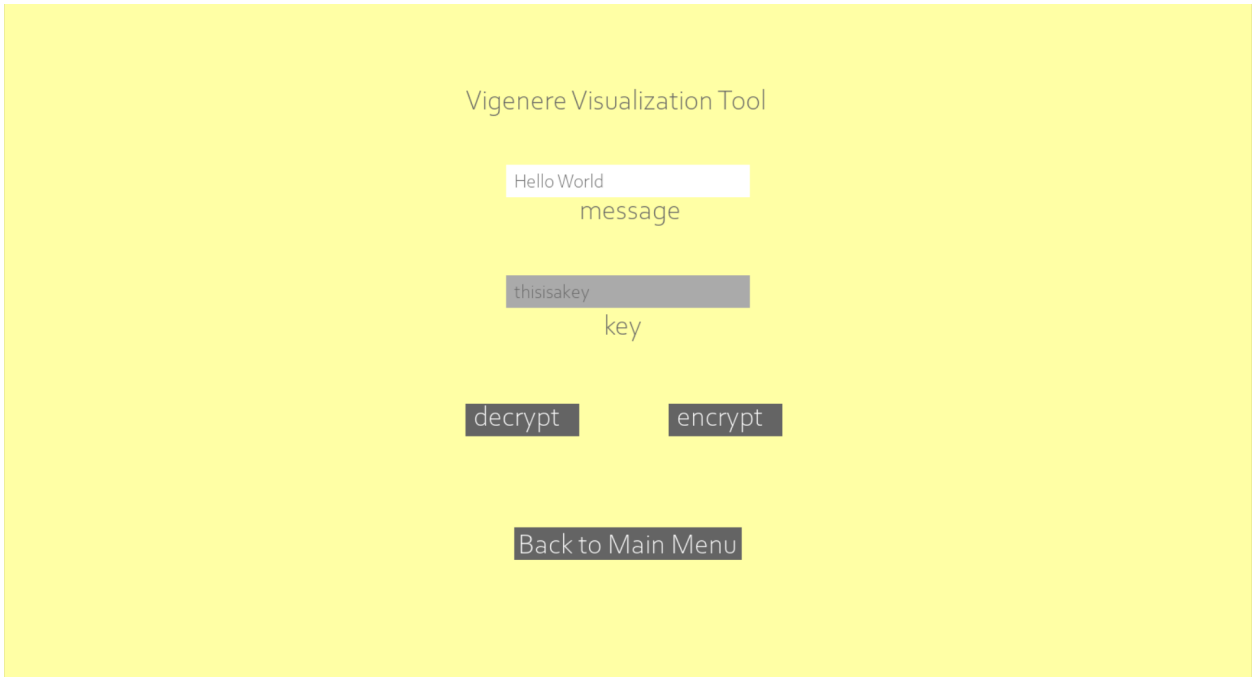
- Displayed when user clicks on “About Ciphers” in InfoMenu(), or when user clicks “Cipher information” in Use().
- Input() detects mouse clicks

**Use():**


**Notes:**

- Displayed when the “About this tool” button is clicked in InfoMenu(), or when the “Tool Information” button on About() is clicked().
- Input() detects mouse clicks

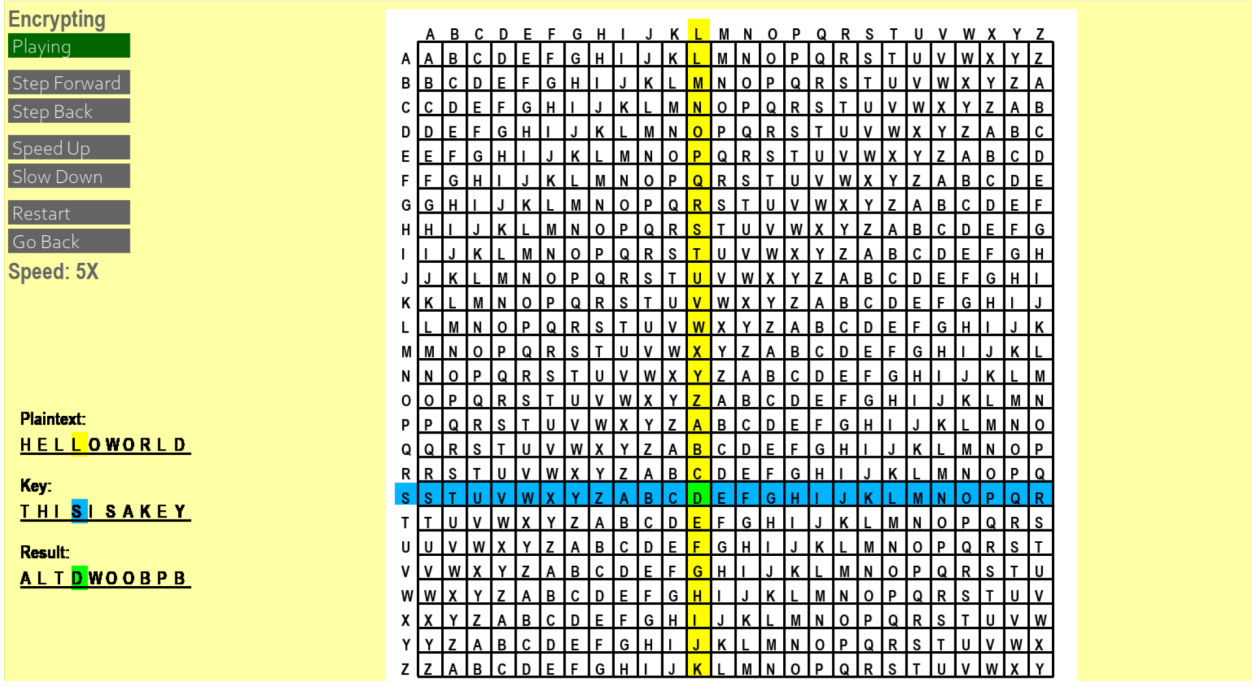
**MainMenu():**

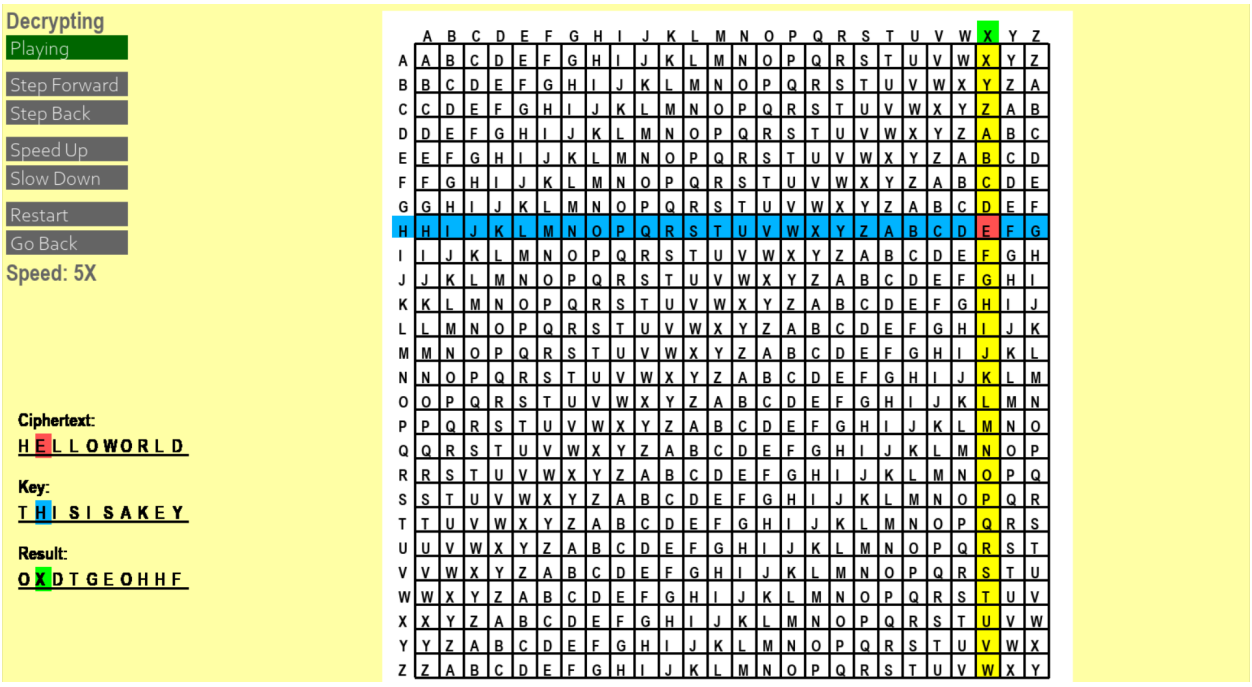



**Notes:**

- Displayed when the user clicks the "Visualization Tool" button in StartMenu()
- Input() detects mouse clicks. Also detects keyboard letters when the input box is selected and displays it to the respective box.

**ButtonScene():**






**Notes:**

- Displayed when the user clicks either the encrypt or decrypt buttons in MainMenu() after entering text into both input boxes
- The visualization that displays depends on whether we are encrypting or decrypting (the mode will be shown in the top left)
- Input() will detect mouse clicks

**Possible Improvements:**

- Fix display for smaller screens
- Add a testing screen that allows the user to test their knowledge of the cipher