**Tp3 Update:**

I made the program more resistant to user input. Now, the user will continue to be prompted for input until what they have entered contains at least one letter. Also, I added checks to the crack program so that it is better at recognizing invalid inputs before it runs. Specifically, in the Playfair cipher, a letter will never encrypt to itself, and there is a bijection between plaintext digraphs and ciphertext digraphs. So if the input has a letter encrypting to itself, or there are multiple digraphs that either encrypt or decrypt to the same thing, an error message will be returned (instead of trying to make the key grid). Likewise, if the ciphertext and plaintext are different lengths once all non-letters are removed, then they can’t be the same message, so an error will be returned. The error shows up on the app by having the key grid turn red and display a message. The user can then click a button to learn more about what the error means.

**TP2 Update:**

My plan remains virtually the same, there are two small changes to what was on my design doc before. First of all, my graphics are slightly less colored than shown in the storyboard. Specifically, in the story board, I colored the changes made to the user’s input in red, and when encrypting / decrypting digraphs, the digraphs were colored based on whether they were in a row, column, or rectangle. This ended up being too time consuming to do, at least while trying to reach MVP, so for the time being I don’t have this (it may or may not be added later). The second change is that in my timeline, I said I would reach MVP and make the functions more user-proof by Apr 22. While I think I have reached MVP, doing the graphics took longer than anticipated, so I have not yet had a chance to make the program respond appropriately if the user inputs strange stuff. I still plan on doing this, I just have moved the date I plan to have it done by to Apr 24.

* **Project Description** [2.5 pts]: The name of the term project and a short description of what it will be.

Name: Playfair Cipher

Description: This project allows uses to encrypt, decrypts, and crack messages using the Playfair Cipher. For encryption and decryption, the user inputs a message (either a normal message or one that has already been encoded) and a key word. Then there are graphics that walk the user through how the cipher works. These graphics show how the key is used to build the encryption/decryption grid, and then how the grid is used to encrypt/decrypt the message.

For cracking, the user enters the message and its encoded version. The program then returns an encryption grid that would yield the proper encryption.

* **Competitive Analysis** [2.5 pts]: A 1-2 paragraph analysis of similar projects you've seen online, and how your project will be similar or different to those.

These are three projects I have seen online that I will be comparing my project to.

Project 1: <https://www.boxentriq.com/code-breaking/playfair-cipher>

Project 2: <https://www.dcode.fr/playfair-cipher>

Project 3: <https://crypto.interactive-maths.com/playfair-cipher.html>

While there are some projects concerning the Playfair cipher online, my project will build upon these projects in two main ways. First of all, my project will be more interactive in explaining how the cipher encrypts and decrypts text. Projects 1 and 2 will allow users to enter a message and a keyword to either encrypt or decrypt, but they don’t illustrate how the encryption/decryption is obtained. This makes it harder for the user to learn how to use the cipher because they can only see what happens, not why it is happening. On the other hand, Project 3 does explain how the cipher works, but only on a fixed example text and key word. My project would improve on this by allowing the user to enter their own message and then walking through the steps. Only one step would be shown on the screen at a time, which would make the process less overwhelming for someone learning about the cipher.

The second way my project will improve upon existing projects is by being able to recreate the encryption grid when given plaintext (message before encryption) and ciphertext (message after encryption). Online, Project 1 doesn’t do any cracking, you always need to know the keyword. Project 3 only vaguely discusses weaknesses of the Playfair but doesn’t explain how one would crack it. Project 2 is the closest to cracking the cipher and does offer a “Known Plaintext Attack” where the user can enter the plain and ciphertext and it will attempt to decode. However, the results of this attack are very hard to understand, as there are just long strings of potential text displayed along the left side of the screen. My project will be much clearer because when the user enters the plaintext and ciphertext, the program will return exactly one grid, or it will say that there is no encryption grid that works. This will make the outcome much easier for the user to understand.

* **Structural Plan** [2.5 pts]: A structural plan for how the finalized project will be organized in different functions, files and/or objects.

I plan on having 5 files:

1. encryptDecrypt.py : This is where the functions for encrypting and decrypting the functions are. The main functions are (I’m not detailing the small helper functions of helper functions):
   1. encDecPlayfair: This is the outside function that calls all helpers and does the encrypting decrypting. It has two modes, ‘encrypt’ and ‘decrypt’.
   2. makeKeyTable: This makes the key table (the encryption grid) when given a key word.
   3. makeDigraphL: This processes the text of the message, removing disallowed characters and making a list of digraphs (pairs of two letters) in preparation for encryption/decryption.
   4. findNewDigraph: This takes a digraph and the encryption grid, and encrypts/decrypts it
2. classes.py : This keeps all classes of objects using for OOP. I plan on having two classes.
   1. A Digraph class (a digraph is a pair of two letters that are encrypted together). This way when I’m in a digraph I can do digraph.let1 and digraph.let2 so that it is very clear what part of the digraph I’m using. And I can have a function to reverse digraphs.
   2. A Letter class which keeps track of information about each letter (for example, which other letters it is in a row or column with).
3. crackTable.py: This is where the cracking of the cipher will happen (user will input plaintext and ciphertext and then an encryption grid / key table will be returned). Its main functions are:
   1. crackKeyTable: This is the main function of this file. A plaintext and ciphertext are input, and a key table that leads to this encryption is returned. All functions below are called by this function (or by helpers of this function)
   2. makeDigraphMap: This takes the plaintext and ciphertext and creates a dictionary where plaintext digraphs are keys and the ciphertext digraphs they encrypt to are the values
   3. createAndPopulateLetterDict: This creates a dictionary that maps each uppercase letter to its instance. It then calls helper functions to add information to the instances. So at the end of the function, the dictionary it returns knows which letters every letter encrypts to, which letters every letter is in a row with, and which letters every letter is in a column with. The function also returns three sets: rows (strings such as ‘ABCD’ that must appear consecutively in a row), cols (strings that must appear consecutively in a column) and rowsOrCols (strings that must either appear consecutively in a row or column)
   4. outerBacktrack: This places all of the strings in rows, cols, and rowsOrCols using backtracking. Its basecase is when all strings have been placed in the board, at which point it calls innerBacktrack
   5. innerBacktrack: This function takes the partial board made in outerBacktrack, and uses backtracking to fill in the gaps by placing all remaining letters one at a time.
   6. Then there are some helpers for the backtracking function which check legality.
4. playfairGraphics.py: This is the main file that must be run. It launches the graphics environment which is what the user interacts with. Here the user can click on options (encrypt, decrypt, crack) and enter their input. This file will call the functions in the other files. This file has your standard graphics functions, such as onAppStart, onMousePress, onMouseHover, and redrawAll, all of which call helper functions to help with each screen of the app (see story board to see screens).
5. Potentially: a file for crackTable.py helper functions. If I end up having a lot of functions that are used once but aren’t really central to the cracking, I might move them to a different file to make each file easier to understand.

* **Algorithmic Plan** [2.5 pts]: A detailed algorithmic plan for how you will approach the trickiest part of the project. Be sure to clearly highlight which part(s) of your project are algorithmically most complex, and include details of the algorithm(s) you are using in those cases.

The most complex part of my project is cracking the key table. What makes this complex is that usually when finding the key table by hand, there is some playing around with placement of letters that uses my knowledge and recognition of the English language, but the computer can’t do this. And while I could do a very brute force backtracking approach, this takes too long for the program to finish because there are 25! possibilities (not all of these are explored, but a lot of them still are explored). Thus, the complexity comes from figuring out how to best gather information from the plaintext and ciphertext in order to efficiently find the key table.

The algorithm that I ended up using has two main phases: first a data collecting phase, and then a double backtracking phase. In the data collecting phase, information about how letters and digraphs (pairs of letters) encrypt is gathered to help with cracking (we get this information by looking at how digraphs encrypt and exploiting facts about how Playfair encrypts things). The difficult part of this phase was probably figuring out data structures to best store the information, along with figuring out how to gather the information about the letters.

However, the main complex part is the double backtracking phase. During the data collection phase, I gathered information on sequences of letters that appear consecutively in rows or columns, and then information on letters individually and which other letters they share rows and columns with. Doing a standard backtracking, where one fills one grid square at a time,

therefore doesn’t work well because it has trouble with the sequences of letters. Therefore, there are two layers of backtracking: the outer layer places all sequences of letters, one block at a time (for example if I knew ‘ABCD’ appeared consecutively in a row, placing all four of those letters at once). Then once all blocks have been placed in the outer backtracking, the base case of the outer backtracking function calls the inner backtracking function which places all letters that haven’t been placed, one at a time. Having two layers of backtracking, which both have different legality checks and different objects that they are placing, is what makes this complex.

* **Timeline Plan** [2.5 pts]: A timeline for when you intend to complete the major features of the project.
* Apr 8: Finish encryptDecrypt (the algorithm itself, not the user interaction part)
* Apr 14: Finish crackTable, including verifying that it is correct, at least in usual cases (the algorithm itself)
* Apr 19: Finish basic graphics for the user interaction component of encryptDecrypt. This doesn’t have to pretty at this point, but it should be functional.
* Apr 22: Make sure everything is working well, make functions more user proof (so that they don’t crash if for example user inputs weird stuff). Should be at MVP by this point. I’m not sure what my MVP looks like, but if there are any MVP features that haven’t been mentioned yet, finish them.
* Apr 24: Continue polishing, verifying correctness, improving user experience.
* Apr 26: Finish TP video demo.
* **Version Control Plan** [1.5 pts]: A short description **and image** demonstrating how you are using version control to back up your code. Notes:
  + **You must back up your code somehow!!!**
  + **Your backups must not be on your computer** (ideally, store them in the cloud)

I am storing all my code in a repository in Github. At the end of every work session, I commit the changes in VS code and then push them to Github. This is an image of the online repository:

Table

Description automatically generated

* **Module List** [1 pts]: A list of all external modules/hardware/technologies you are planning to use in your project. Note that any such modules must be approved by a tech demo. If you are not planning to use any additional modules, that's okay, just say so!

I’m not currently planning on using any additional modules.