CS1132 Fall 2013 Assignment 2 due 10/3 11:59pm

> *Adhere to the Code of Academic Integrity.* You may discuss background issues and general strategies with others and seek help from course staff, but the implementations that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is never OK for you to see or hear another student's code and it is never OK to copy code from published/Internet sources. If you feel that you cannot complete the assignment on you own, seek help from the course staff.

When submitting your assignment, follow the instructions summarized in Section 4 of this document.

Do not use the `break` or `continue` statement in any homework or test in CS1132.

Crypto Vigenere

Just several decades ago, cryptography existed solely in the domain of national governments and large corporations. However, the advent of the Internet and the wide availability of computers have now made cryptography an indispensable part of everyday digital communications. In this part of the assignment, you will implement one well-known classical ciphers: the Vigenre cipher. This ciphers predates modern cryptography but nonetheless provide a relevant and insightful look at some commonly used cryptography techniques. You will implement it as MATLAB functions that can be used to encrypt and subsequently decrypt messages.

The Vigenre cipher is one of the most well-known ciphers of all time. It is a method of encrypting plaintext using a series of different Caesar ciphers based on the letters of the keyword. The Vigenre cipher can be implemented using what is called a tabula recta, or Vigenre square. It consists of the alphabet written out 26 times in different rows with each row consisting of the alphabet shifted cyclically to the left. The Vigenre square is shown below [1]:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

[1] http://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher

For example, the plaintext MATLABISFUN when encrypted using the key CIPHER, gives the ciphertext OIISESKAUBR. The encryption works by first taking the key and repeating it till it matches the length of the plaintext. Subsequently, each character in the plaintext is paired with the character in the key in the corresponding position. The characters of the plaintext correspond to the columns of the Vigenre square and the characters of the key correspond to the rows. For instance, using the example below, the first letter of the plaintext M is paired with the first letter of the key C. The row C and column M of the Vigenre square gives the letter O. This is repeated for the rest of the letters in the plaintext.

Plaintext: MATLABISFUN
Key: CIPHERCIPHE
Ciphertext: OIISESKAUBR

For the purpose of the assignment, we simplify the implementation of the cipher by only allowing the plaintext (and ciphertext) to contain only uppercase letters with no other punctuation, whitespaces or symbols. If the user were to enter invalid characters, the function should remove them. If the user were to enter lowercase letters, they should be converted to uppercase. For example, if the input is MATLAB is fun!, it should be converted to MATLABISFUN.

Additionally, the function that you implement, when given the correct key, should be capable of decrypting any ciphertext that was encrypted using the Vigenre cipher. A parameter in the function should be used to indicate which mode the function is in: encryption or decryption.

Write a function that implements the Vigenre cipher as such:

```
function output = vigenere(input, key, encdec)
% An implementation of the Vigenere cipher
%
% input: Plaintext to be encrypted or ciphertext to be decrypted
% key: key to be used in encryption
% encdec: 0 means encrypt and 1 means decrypt; assume that encdec is always
% 0 or 1
% output: outputs ciphertext when encrypting; outputs plaintext when
% decrypting
```

Breaking Bad

Breaking Bad[2] is an American television crime drama series created and produced by **V**ince Gilligan. Set and produced in Albuquerque, New Mexico, Breaking Bad is the story of Walter White (Bryan Cranston), a struggling high school chemistry teacher who is diagnosed with inoperable lung cancer at the beginning of the series. He turns to a life of crime, producing and selling methamphetamine in order to secure his family's financial future before he dies, teaming with his former student, Jesse Pinkman (Aaron Paul)

One interesting fact is that the title of the series is spelled using the chemical symbols for bromine ("Br"), and barium ("Ba") `https://www.youtube.com/watch?v=F1HNuAE9WdU`. Chemical symbols from the periodic table of the elements also appear in every name in the opening credits: a single capital letter, or letter-pair with only the first letter capitalized, shown in a different color.

[2] http://en.wikipedia.org/wiki/Breaking_Bad

You are provided with the names of the cast in `BreakingBadCastName.txt` and the chemical symbol list `ElementList.txt`. For each cast name, find the first substring that matches one of the elements in the chemical symbol list. Then convert the name by adding quotation before and after the found substring along with an arrow pointing to the corresponding element name. For example, the main character "Walter White" is converted to `"W"alter White -> Tungsten`. The string matching process is not case sensitive and you only need to find the first match for each name. However, you should keep the converted name's capitalization the same as the original input.

Your task includes:

- Read the two txt files and store the string information in two cell arrays.
- Write the converted cast names (in the same order as the given file) to a file called `ConvertedCastName.txt`. If no match is found, the converted name should be the same name as the original cast name.

Below is a list of useful built-in functions for handling characters, strings, and files. Use only these built-in functions for handling characters, strings, and files. You may not need all of them. You can of course still use general built-in functions not related specifically to strings and files, such as *length*, *zeros*, *rem*,$\cdots$ etc.
You can use:

- `fopen, feof, fgetl, fclose`
- `strcmp, upper, isspace, isletter`

You must `NOT` use matlab build in function `find, strfind, findstr`.

Write a function that implements the following:

```
function BreakingBadStyle(castNameFile,elementNameFile ,convertedNameFile)
% This function converts a list of name from castnNmefile to convertedNameFile
% according to the match in elementNameFile
%
% input:
% castNameFile: the name of the file that contains the cast names
% elementNameFile: the name of the file that contains the chemical symbols.
% convertedNameFile: the name of the file that contains the converted names for output.
%
% e.g:
% BreakingBadStyle('BreakingBadCastName.txt','ElementList.txt','ConvertedCastName.txt')
```

The file `ConvertedCastName.txt` looks like:

```
"W"alter White -> Tungsten
"S"kyler White -> Sulfur
J"es"se Pinkman -> Einsteinium
"H"ank Schrader -> Hydrogen
M"ar"ie Schrader -> Argon
```

Reversi

In this part of the assignment, you will implement a two-player game Reversi.

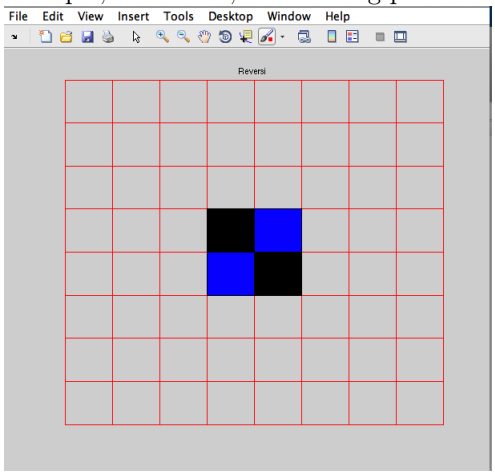
Basic rules: Each reversi has a blue side (player1) and a black side (player2). On your turn, you place one piece on the board with your color facing up. You must place the piece so that an opponent's piece, or a row of opponent's pieces, is flipped (captured) by your pieces (After placing the piece, your piece turns over all opponent pieces lying on a straight line between the new piece and any of your anchor pieces. The opponent pieces can be on the same row, same column or same diagonal as your pieces). In another word, a valid move must be a move that can capture your opponent piece. Then all of the opponent's pieces between your pieces are then turned over to become your color. The rules of the game, along with some good ideas for a sound strategy of play, are described in the Wikipedia entry on Reversi – do read it `http://en.wikipedia.org/wiki/Reversi`. You can try playing this game by clicking on this link `http://www.gamesforthebrain.com/game/reversi/`. Usually, Reversi is played on a square board divided into an 8×8 grid.

Be sure you know how to play the game before attempting to complete this problem; it will save you much time and effort.

Starting Point In the `files.zip`, we provided a **ReversiAI.p** which is an intelligent *AI* player that can play the Reversi game.

```
function [xNew,yNew] = ReversiAI(gameboard,myTurn)
%The is a ReversiAI function that can play the Reversi game
%
%Input
%gameboard: is a NxN board that contains only 0 ,1 or 2
%           0 means the space is open, 1 means player1 has taken it
%           2 means player2 has taken it
%myTurn: an integer either 1 or 2. 1 indicates it is player1's turn,
%            2 indicates it is player2's turn
%
%Output:
%[xNew,yNew]: the output for the location where the AI wants to place its
%             piece on the gameboard given.
%             if xNew is -1 or yNew is -1, it means no valid move is
%             available for the current board configuration.
```

Project details The game is set up as an $N \times N$ grid (N is an even number.) The grid is initialized by having the central four pieces taken with two player1's pieces and two player2's pieces. For example, for $N = 8$, the starting position would look like the following:



Then you can start playing with the *AI*. Take advantage of matlab build in function *ginput* by asking the user to click on a location of the board. If it is a valid move, repaint the board according to the new board configuration.
As a player clicks on the board, three situations might happen:

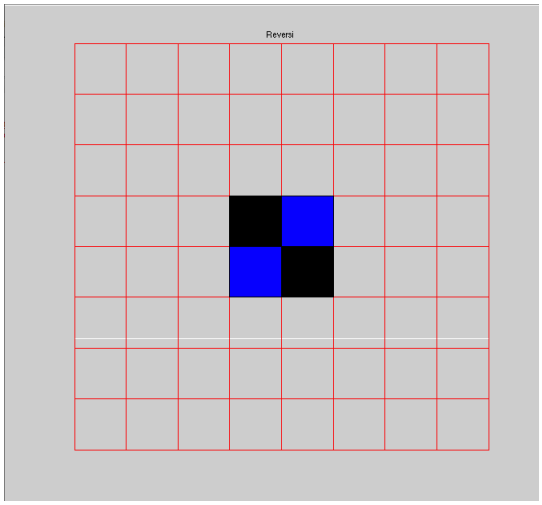
- If it is an invalid move, nothing should happen and the player needs to click on the board again. You can display messages to indicates that a player needs to reclick on the figure.
- If it is a valid move, then recompute the gameboard, flip and repaint the pieces accordingly. Call the AI player to play on the new gameboard, then recompute and repaint the board again.
- if after the player's move, no more valid move is available for either players, then the game should stop and display who won as the title of the figure.

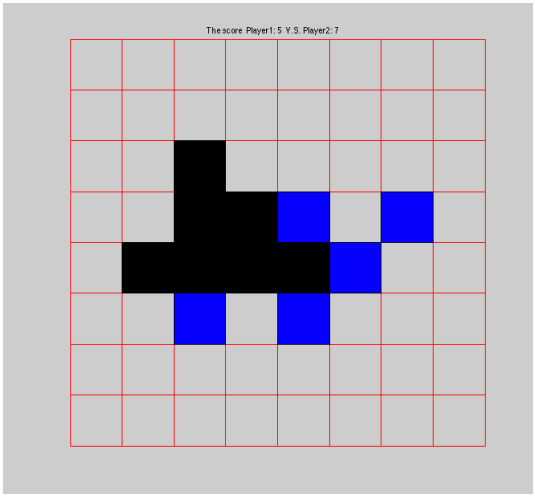
Pay extra attention to when there is no move left for either players. When that happens, you should let that player skip and ask the other player play.

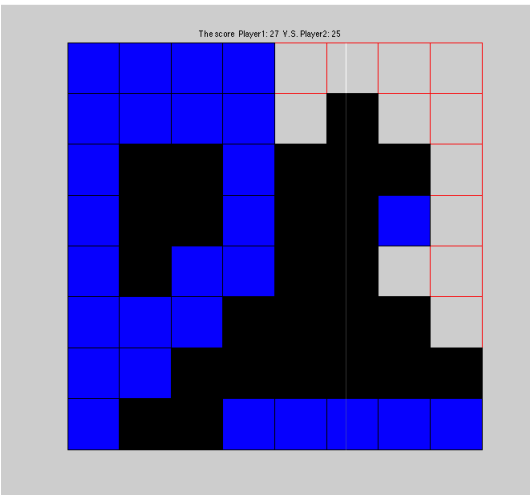
Write a function `Reversi.m` to implement the game as described above. Write subfunctions to build this game in a modular fashion. Taking the time to decompose the problem and implement subfunctions will end up saving you time overall!

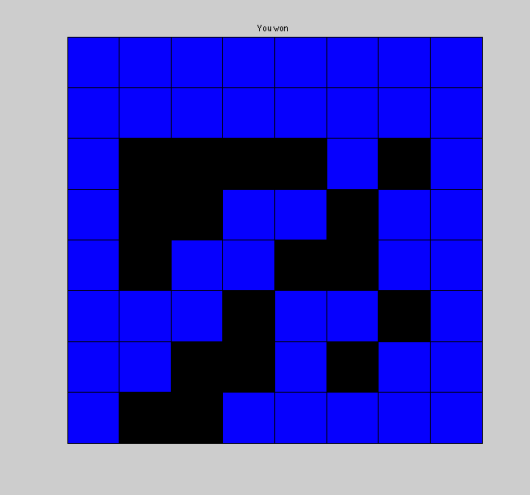
```
function gameboard = Reversi(N,myTurn)
 % A two-player game of Reversi on an N-by-N grid
 % Input:
 % N:  the size of the board. N should be an even number.
 % myTurn:  an integer 1 or 2. 1 means the human is the first player to make a move,
 %                2 means the human is the second player to make a move.
 %Output:
 %gameboard: a NxN matrix which stores the finished board configuration.
 %                   0 means the piece is not taken, 1 means player1's piece
 %                   2 means player2's piece
```

Below are four snapshots of a game.










```
gameboard = Reversi(8,1)
gameboard =
     1     1     1     1     1     1     1     1
     2     1     1     2     2     2     1     1
     2     2     1     1     2     2     1     1
     1     2     2     1     1     2     1     1
     1     1     1     2     1     2     1     1
     1     2     1     2     2     1     1     1
     1     1     2     1     1     2     1     1
     1     1     1     1     1     1     1     1
```

Self-check list

The following is a list of the minimum *necessary* criteria that your assignment must meet in order to be considered *satisfactory*. Failure to satisfy any of these conditions will result in an immediate request to resubmit your assignment. Save yourself and the graders time and effort by going over it before submitting your assignment for the first time. Although all these criteria are necessary, meeting all of them might still not be *sufficient* to consider your submission satisfactory. We cannot list everything that could be possibly wrong with any particular assignment!

- Δ Comment your code! Make sure your functions are properly commented, regarding function purpose and input/output parameters.
- Δ Suppress all unnecessary output by placing semicolons (;) appropriately. At the same time, make sure that all output that your program intentionally produces is formatted in a user-friendly way.
- Δ Make sure your functions names are *exactly* the ones we have specified, *including* case.
- Δ Check that the number and order of input and output parameters for each of the functions match exactly the specifications we have given.
- Δ Test each one of your functions independently, whenever possible, or write short scripts to test them.

Δ Check that your scripts do not crash (i.e., end unexpectedly with an error message) or run into infinite loops. Check your script several times in a row. Before each test run, type the commands `clear all; close all;` to delete all variables in the workspace and close all figure windows.

Submission instructions

1. Upload files `vigenere.m`, `BreakingBadStyle.m` and `Reversi.m` to CMS in the submission area corresponding to Assignment 1a in CMS before the deadline. Late submission is accepted up to 24 hours after the deadline with a 10% penalty.

2. *After grading:* If you resubmit the assignment, upload your corrected files and *be sure to select the* ***Regrade Request*** *option.*