Assignment 6

Please be reminded that you may be penalized for inefficient code or code that is too long.

Part 1: Tic-tac-toe (40 marks)

We've provided some partial code for the game Tic-tac-toe. In this part, you are going to complete the implementation to make it a complete game.

In order to complete the game, you will have to complete three tasks. Three utility functions are provided for you: `create_game_matrix`, `print_TTT` and `ttt_gameplay`.

Task 1

Write a function `check_valid_move(game,inp)` that takes the board game and `inp` as user input and check if the input is valid (You have to think about what is a valid input). **If the input is invalid, you need to return `False`, otherwise return `True`.** You are ensured that the input will always be an integer.

Task 2

Write a function `check_win(game)` to check if a board game has a winner. If there is a winner, return the winner symbol 'X' or 'O'. Otherwise, if the game has no winner yet, return an empty string.

Sample gameplay can be found below. (The color is just to show you the new move. You are not required to implement it.)

```
1|2|3
-----
4|5|6
-----
7|8|9

Player X move:5
1|2|3
-----
4|X|6
-----
7|8|9

Player O move:1
O|2|3
-----
4|X|6
-----
7|8|9
```

```
Player X move:1
Invalid move!
Player X move:4
O|2|3
-----
X|X|6
-----
7|8|9

Player O move:6
O|2|3
-----
X|X|O
-----
7|8|9
```

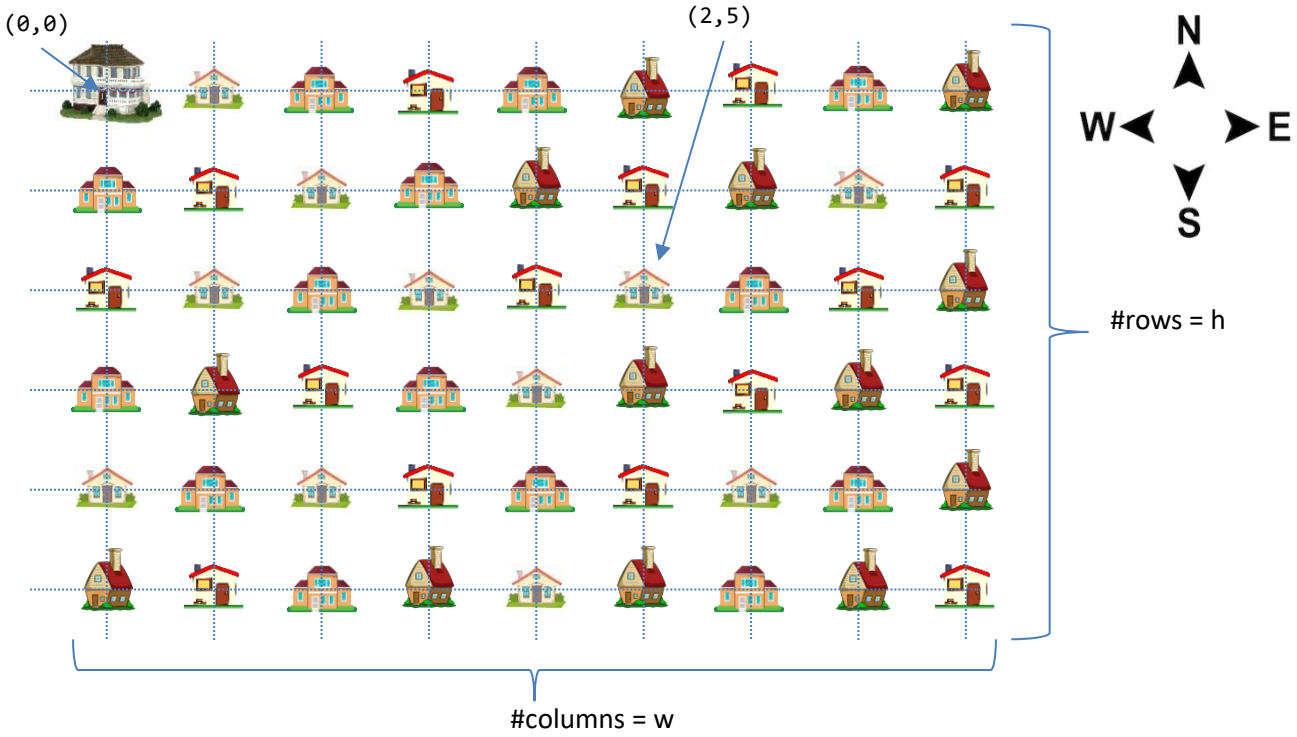
```
Player X move:7
O|2|3
-----
X|X|O
-----
X|8|9

Player O move:2
O|O|3
-----
X|X|O
-----
X|8|9
```

```
Player X move:3
O|O|X
-----
X|X|O
-----
X|8|9
Player X won!
```

Part 2: Lightning Pizza Delivery (60 marks)

In a town called ***Regulaville***, and all the people living there have very strange habits. All the buildings have centers on a well-structured rectangular grid. The mayor of the town is at the most north-west corner of the town, and his house coordinates are `(0,0)`. And each building in town has coordinates `(i,j)` that indicate that his house is `i` km *south* and `j` km *east* from the mayor's house for i and j are integers such that $0 \leq i < r$ and $0 \leq j < c$ for some integers `r` and `c`.



A new brand of pizza came to town! They set up **n** stores in some of the buildings in town for some **n** ≤ 10. We store the coordinates of the pizza stores in a list. For example, a list of

```
[[10,20],[30,20],[40,50]]
```

represents three stores of pizza with Store 0 at `(10,20)`, Store 1 at `(30,20)` and Store 2 at `(40,50)`[1]. (Somehow the big boss of the pizza stores knows programming and he starts counting by 0 also.) There are no two pizza stores in the same location. All the people in Regulaville only eat pizza in their own homes by calling for delivery, and all the stores will deliver pizza by flying drones that will fly directly from the stores to the destination.

In order to minimize the time and power used by the drones, the mayor ordered that every home must only order pizzas from the nearest store, unless there is more than one store with equal minimal distance to that home. For example, for the three pizza stores mentioned in the example above, the house at coordinates `(40,20)` will be

[1] The coordinates are a bit confusing because for a location `[10,20]` in this assignment, it means going south (vertical direction) for 10km and 20km east (horizontal direction). This is the opposite way of what we usually think in real life where for `(x,y)`, `x` is the horizontal direction and `y` is the vertical direction.

closest to Store 1 with the nearest distance 10 km. Stores 0 and 2 will have more than 1 store with the same distance 30 > 10 km. Two utility functions `create_zero_matrix` and `m_tight_print` are provided for you.

Task

Write a function `pd_map(r,c,sites)` to compute a map for ALL houses in Regulaville to show the closest pizza store number to each house, where `r` and `c` are the height and width of the map respectively, and `sites` is a list of pizza store coordinates.

The location of each house should be represented by the coordinates `(i,j)` such that `0 ≤ i < r` and `0 ≤ j < c`. You can assume the number of pizza stores will be less than or equal to 10 and their corresponding store numbers (i.e. labels) will be from 0 to 9. Below shows a sample usage of the function pd_map().

```
>>> pizzaMap = pd_map(7,8,[[1,3],[4,7],[7,2]])
>>> pprint(pizzaMap)
[[0, 0, 0, 0, 0, 0, 0, 1],
 [0, 0, 0, 0, 0, 0, 0, 1],
 [0, 0, 0, 0, 0, 0, 1, 1],
 [0, 0, 0, 0, 0, 1, 1, 1],
 [2, 2, 2, 0, 1, 1, 1, 1],
 [2, 2, 2, 2, 2, 1, 1, 1],
 [2, 2, 2, 2, 2, 1, 1, 1]]
```

```
>>> m_tight_print(pizzaMap)
00000001
00000001
00000011
00000111
22201111
22222111
22222111
```

For example, it shows that the home at `(3,6)` is closest to the pizza Store 1.

For houses with more than one pizza store with equal minimal distance to the house, mark the house with an **'X'** instead of a store number. An example is shown below.

```
>>> m_tight_print(pd_map(10,10,[[2,3],[4,9],[7,2]]))
0000000X11
0000000111
0000000111
000000X111
X000001111
22222X1111
2222221111
2222222111
2222222111
2222222X11
```

Note that since we are using only integer arithmetic, the distance computation must be exact. (Wait, isn't that a "`sqrt()`" in the distance computation?)

You should submit the function pd_map, together with any helper functions created to support pd_map (if any).

An example with a larger map (for large h and w) is shown on the next page.

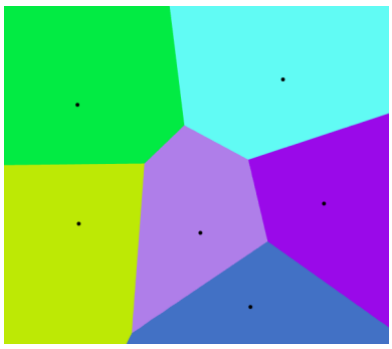
```
>>> m_tight_print(pd_map(50,80,[[20,10], [30,30],[40,20],[45,55],[10,55],[35,70],[35,60]]))

00000000000000000000000000000044444444444444444444444444444444444444444444444444
00000000000000000000000000000044444444444444444444444444444444444444444444444444
00000000000000000000000000000044444444444444444444444444444444444444444444444444
00000000000000000000000000000044444444444444444444444444444444444444444444444444
00000000000000000000000000000004444444444444444444444444444444444444444444444444
000000000000000000000000000000X4444444444444444444444444444444444444444444444444
00000000000000000000000000000011444444444444444444444444444444444444444444444444
00000000000000000000000000000X11144444444444444444444444444444444444444444444444
00000000000000000000000000000111144444444444444444444444444444444444444444444444
0000000000000000000000000000X111114444444444444444444444444444444444444444444444
00000000000000000000000000001111111444444444444444444444444444444444444444444444
000000000000000000000000000X1111111144444444444444444444444444444444444444444444
00000000000000000000000000011111111114444444444444444444444444444444444444444444
00000000000000000000000000X11111111114444444444444444444444444444444444444444445
00000000000000000000000000111111111111444444444444444444444444444444444444444555
0000000000000000000000000X111111111111144444444444444444444444444444444444X5555
00000000000000000000000001111111111111114444444444444444444444444444444444555555
000000000000000000000000X1111111111111111444444444444444444444444444444455555555
00000000000000000000000011111111111111111444444444444444444444444444444X555555555
00000000000000000000000X1111111111111111114444444444444444444444444445555555555555
00000000000000000000000111111111111111111114444444444444444444444445555555555555
0000000000000000000000X111111111111111111111444444444444444444444X55555555555555
000000000000000000000011111111111111111111111444444444444444X6666X55555555555555
000000000000000000000X111111111111111111111114444444444X666666666X55555555555555
00000000000000000000011111111111111111111111114444X66666666666666X55555555555555
00000000000000000000X11111111111111111111111111666666666666666666X55555555555555
00000000000000000000111111111111111111111111111666666666666666666X55555555555555
0000000000000000000X111111111111111111111111116666666666666666666X55555555555555
00000000000000000001111111111111111111111111116666666666666666666X55555555555555
00000000000000000X2X111111111111111111111111116666666666666666666X55555555555555
000000000000000X2222X11111111111111111111111116666666666666666666X55555555555555
0000000000000X2222222X1111111111111111111111116666666666666666666X55555555555555
00000000000X2222222222X111111111111111111111116666666666666666666X55555555555555
000000000X2222222222222X11111111111111111111166666666666666666666X55555555555555
0000000X2222222222222222X11111111111111111111366666666666666666666X55555555555555
00000X2222222222222222222X111111111111111111X33366666666666666666X55555555555555
000X2222222222222222222222X11111111111111111333333666666666666666X55555555555555
0X2222222222222222222222222X111111111111111333333333666666666666666X55555555555555
2222222222222222222222222222X11111111111111333333333336666666666X55555555555555
22222222222222222222222222222X11111111111133333333333333666666666X55555555555555
222222222222222222222222222222X1111111111X33333333333333336666666X55555555555555
2222222222222222222222222222222X111111111333333333333333333366666X55555555555555
22222222222222222222222222222222X11111113333333333333333333333666X55555555555555
222222222222222222222222222222222X11111133333333333333333333333336X55555555555555
2222222222222222222222222222222222X111133333333333333333333333333355555555555555
22222222222222222222222222222222222X11X333333333333333333333333333355555555555555
222222222222222222222222222222222222X1333333333333333333333333333335555555555555
22222222222222222222222222222222222223333333333333333333333333333333555555555555
22222222222222222222222222222222222223333333333333333333333333333333555555555555
22222222222222222222222222222222222223333333333333333333333333333333355555555555
```

*In fact, the *map* we produced in this problem is called the *Voronoi Diagram*. You can google for the definitions of Voronoi Diagram and Delaunay Triangulation, or some apps to play around with. (E.g. http://alexbeutel.com/webgl/voronoi.html)

Extra (No mark)

If you want to challenge yourself even more, try to convert your map into an image and display it with different colours.



Part 3(Optional) [0 mark]: Image Composition

Welcome to the Marval Studio! (Oops, you noticed that? We are 'Marval', not 'Marvel'…) You are the one who is responsible to make the movie ***AveNUgerS*** !
Here is a scene captured in the green screen studio on the left, and the COM1 building on the right with some modifications.

"avergers green.jpg"

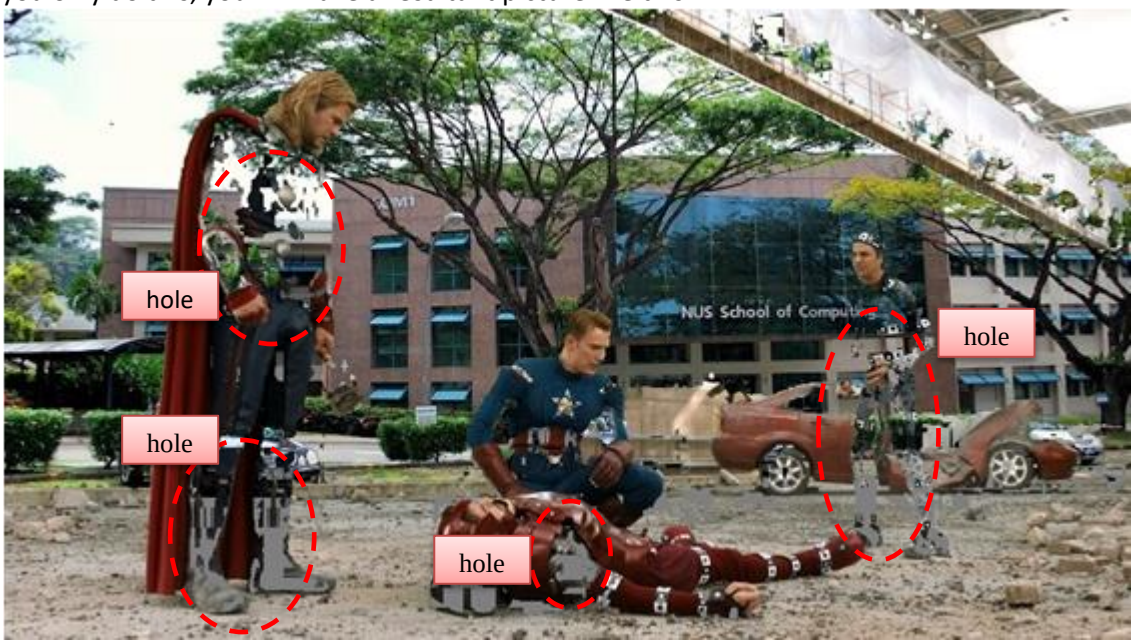
"background.jpg"

Your job is, to remove the green pixels in the left picture and replace them by the background at the right.

The two pictures are aligned at the top left corner. And the background picture is "larger" than the foreground picture. And you do not need to care about the extra part in the background that is larger than the foreground picture. Therefore, your resultant picture should have the same size as the original one, NOT the background.

First, you should try to figure out all the pixels that are "green". The criteria for a "green" pixel is that the green component G in its color [R, G, B], is larger than the other two, G > R and G > B.

However, if you only do this, you will have a resultant picture like this:

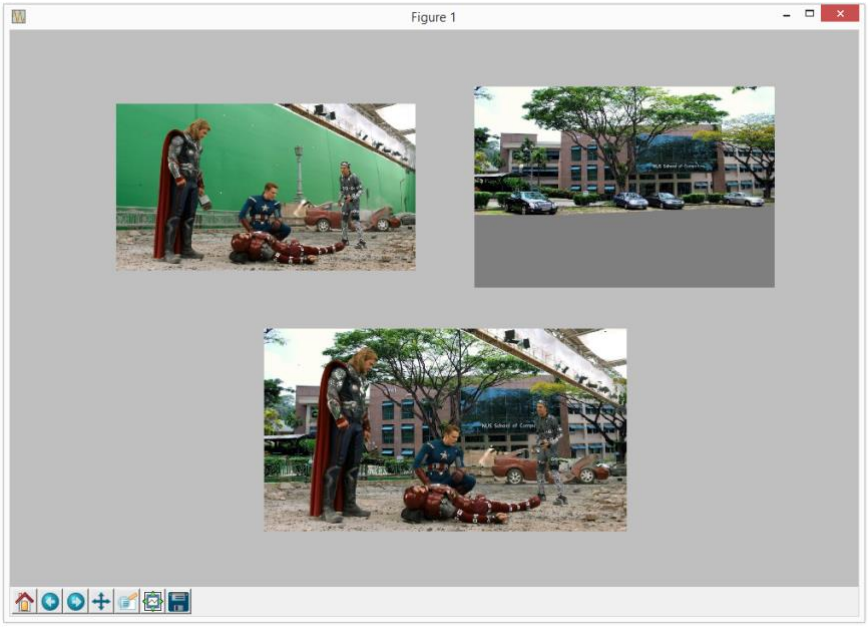


There are a lot of holes because those regions have very dark colors and are considered as “green”. So in order to mend the holes, we need to include one more criteria that the pixel is replaced when it is “green” AND the green value G is > 110. Combining these two criteria, we will have a resultant image like this:

Although there are still some holes on Bruce and other heroes… but let’s forget it and leave it to the next stage of compositions with the additions of armors and Hulk.

So your **tasks** are

a. Write a function “`image_composition(filename1, filename2)`” that takes in the foreground (filename1) and remove the green and replaced by the background image (filename2)
b. Save your resultant image as `'aveNUgerS.jpg'` in the function “`image_composition`”.
c. Also, in the function “`image_composition`”, display the original foreground and background images. You may display any way you want or here is a sample below:



Assignment 6