CS 404 – Artificial Intelligence
HW 3 – Local Search

My Google Colab File Link: https://colab.research.google.com/drive/12HkB-LkUfyD215iRRR3S_F-MFprbbqnd?usp=sharing
I modified the given code to get the number of fails and restarts by adding some global variables. And you can find the topics of the simulations on the left side. I've also added my RandomNeighbor function at the end of this file in case of a problem in the Google Colab link.

Goal: To learn more about local search which has very low memory usage and can be quite successful for many problems; and to gain further experience with programing and reporting research results.

Task: Solve the N-queen problem with different local search algorithms. N-queen problem is finding the placement of N queens on an NxN board such that no one attacks one another. The given Python code implements basic hill climbing and random restarts. You will be asked to a) run simulation experiments with the given code and b) expand the code with stochastic hill climbing.

a**) 50 pts**- Given the code in link , run **100** simulations/experiments with different initial solutions and give the number of **successes, number of iterations, and the time it takes** to find the solution on **average** in each case, for **N=10 and N=20,** for **basic** hill climbing and **random restart** with increasing number of restarts (k=10, 100, 1000)

	N=10			**N=20**		
	Percentage of success in 100 runs	**Solutions found in how many restarts on average**	**Elapsed time to complete experiments**	**Percentage of success in 100 runs**	**Solutions found in how many restarts on average**	**Elapsed time to complete experiments**
Basic Hill Climbing	5%	-	0.009979753494262696	2%	-	0.2457879066467285
Random Restart with k=10	47%	3.851063829787234	0.07138312101364136	17%	5.294117647058823	2.439107937812805
Random Restart with k=100	100%	12.63	0.12097878694534302	84%	29.976190476190474	10.757348778247833
Random Restart with k=1000	100%	14.06	0.13472453117370606	100%	45.39	13.733358187675476
Stochastic hill climbing *(to fill for part b)*	8%	-	0.010958480834960937	2%	-	0.1493161988258362
Simulated Annea		-			-	

ling ***(to fill for part c) – if you will do the bonus)***						

b) **50 pts**- Add a new function randomNeighbor(....) to implement **stochastic hill climbing**. If no better neighbour, should return current one. Leave other code the same.

Fill the results of 100 experiments to the corresponding row of the table with stochastic hill climbing.

c) Bonus-**15 pts**: Implement **simulated annealing** and fill the results of 100 experiments to the corresponding row of the table. Specify the best parameters you found (what is the schedule and initial temperature) here or in the table.

```python
def randomNeighbor(position):
    """
    returns a random neighbor which is better than the current position
    """
    random_list = []
    while len(random_list) < len(position)**2:
      random_i = randrange(0, len(position))
      random_j = randrange(0, len(position))
      while [random_i, random_j] in random_list:  #if the random pair is generated before
        random_i = randrange(0, len(position))
        random_j = randrange(0, len(position))
      currentNumberOfConflicts = calculateNumberOfConflicts(position)
      if random_j != position[random_i]:
        temp = position.copy()
        temp[random_i] = random_j
        if calculateNumberOfConflicts(temp) < currentNumberOfConflicts:
          return temp
      random_list.append([random_i, random_j])
    return position #if there is no better neighbor
```