DFS: I have implemented DFS in a very similar way as the given pseudo code. I have used the terminal condition as **pacman's death** and the comparison of states is done based on the **score**. Also, I have kept a depth limit (Maximum integer value because it still works well). It doesn't work outstandingly well because it considers the first move out of the available ones and keeps on expanding the tree using that move. If the pacman doesn't die after reaching the depth limit, it selects that move. But it ignores that fact that there could have been better moves if it had searched for them in the tree. (Max score: 10,200)

IDFS: IDFS is like DFS but the depth keeps on increasing at each iteration. IDFS gives somewhat similar results to DFS (Max score: 8,300)

Stochastic Hill Climbing: When the getMove function is called, it finds out the possible moves and then sends them to the hillClimb function. Now, in that function, again the next possible moves are found out and a random move is selected. If it is good enough, then it goes forwards to check the next moves instead of just returning that move to the getMove function. It goes on till the pacman is dead or the depth limit is reached. The getMove function gets the best values of all the possible moves given to the function one by one and then selects the best move. So, if there are 3 possible moves at an instance, I generate 3 trees and use the stochastic hill climbing on them and compare the root values of the trees and make the move which is better. (Max score: 6,160)

Simulated Annealing: It is like hill climbing but pacman can make a bad move (to avoid local maximum) after satisfying some conditions. If the score after making a random move is still the same, even then the pacman takes that move if the temperature and the acceptance condition is satisfied. The acceptance condition is based on the badness of the move which is calculated using the distance from the ghost after taking that move, i.e., pacman makes a bad move (score remains the same) only if the pacman doesn't end up getting too close to the ghost. (Max score: 7,230)

BFS: Unlike DFS, it wasn't appropriate to use recursion for BFS. Hence, I used node class and stored the state information for that node so that the information wouldn't be lost. I have used a linked blocking queue to implement BFS. (Max score: 8,210)

A star: I have used a Heuristic class which implements the comparator interface to compare and evaluate the heuristic values and a priority queue (as suggested in the lectures). I have used number of remaining pills as the heuristic. This algorithm works better than most because of the nature of the algorithm to consider the previous moves along with the next ones. (Max score: 12,440).

Genetic algorithm (with and without mutation): The getMove function finds the possible moves at that instance and calls the genetic function. The genetic function generates a random population based on the next possible moves after advancing the game state. It evaluates the score of the game for every individual, selects the best score from the population and returns it to the getMove function. The getMove function compares the best score it obtained from those function calls and selects the best one. In case of mutation, the worst half of the population is mutated using uniform mutation, the scores are calculated again and best score is evaluated accordingly. (Max score: - with: 7,910, without: 5,850)

I have modified all the algorithms by using the following things:

1. The pacman **runs away** from the ghost if it is closer than a specified distance
2. The pacman **tries to eat** the ghost if it is edible and near a specified distance
3. The pacman **searches the power pills** and runs towards them if they are close enough

(Note: The scores of all the algorithms vary if we use different values for the variables used for the modifications mentioned above)