################## Pacman Instructions ####################

Assignment:

Implement

1. Hill Climber Algorithm in the class **HillClimberAgent**,
2. Genetic Algorithm in the class **GeneticAgent**,
3. Monte Carlo Tree Search (MCTS) in the class **MCTSAgent**,

within the in **pacmanAgents.py** file, using **scoreEvaluation & normalizedScoreEvaluation** as a heuristic functions.

Notes:

- Python 2.7 is required to run the Framework.
- All your code must be inside the **pacmanAgents.py** file.
- Only submit **pacmanAgents.py** file. If you submit anything else you will fail the assignment.
- Only use the heuristic function in the **heuristics.py** file (do not implement your own heuristic).
- **RandomAgent**, **GreedyAgent, and RandomSequenceAgent** are implemented example agents.
- **Hill Climber** and **Genetic Algorithm** plans a sequence of actions to execute instead of building a tree. Check **RandomSequenceAgent** as an example of sequence planning.
- External libraries are not allowed (as you won't submit them)
- You will fail the assignment if you try to change any of the system params.
- You are only allowed to use these system functions (accessing/ changing any other functions or variables is considered cheating):
    - **state.getLegalPacmanActions():** return all the legal actions in this state

    - **state.getAllPossibleActions():** return all the possible actions (Directions.North, Directions.South, Directions.East, Directions.West)
    - **state.generatePacmanSuccessor(action):** return the next state if pacman take a certain action (return a new copy, doesn't modify the current state)
    - **scoreEvaluation(state):** evaluate the current state
    - **normalizedScoreEvaluation(rootState, currentState):** evaluate the current state with respect to rootState then scale it down.
    - **state.isWin():** check if this state is win state
    - **state.isLose():** check if this state is lose state
- The forward model (**generatePacmanSuccessor**) is limited to a certain amount of calls, don't waste them. If you exceed the limit, **None** will be returned.
- For **Hill Climber** and **Genetic Algorithm**:
    - Use **getAllPossibleActions** to assign your action sequence. Don't worry applying illegal action. The forward model ignore illegal actions and apply Directions.STOP instead.
    - Make sure to check if you reach the terminal state while implementing your action sequence as the remaining actions can't be executed.
    - Always return the first action from the sequence with the highest **scoreEvaluation**.
- For **Hill Climber**:
    - Action Sequence are of length 5
    - Each action in the sequence has 50% chance to be changed into random action.
- For **Genetic Algorithm**:
    - Each chromosome is an action sequence of length 5.

    - Use a population of size 8.
    - Where you have 70% to crossover followed by 10% mutation for each chromosome.
    - For each individual you roll the die, 70% will do the crossover, then these two chromosomes have 10% chances to be mutated.
    - Use rank selection to select chromosomes
        - Sort all chromosomes based on fitness.
        - Give each chromosome a rank (from 1 to length of chromosomes).
        - Select chromosomes proportionally to their rank
- For **MCTS**:
    - Exploitation should use **normalizedScoreEvaluation** function instead of **scoreEvaluation** to balance it with respect to the Exploration term.
    - Use **getLegalPacmanActions** to build the tree.
    - Use 1 as the constant between exploitation and exploration.
    - Fix the number of rollouts to 5 as its hard to reach a terminal state in small amount of time.
    - Don't save the state in the tree nodes as the game is stochastic. Always apply the actions from the root state. every time you go over the tree.
    - Final Action Selection: return the action associated with the most visited node (Break ties randomly).
- You can't change name of the agent classes.
- For array sorting, you can use python internal sorting function. example: **array.sort(key=lambda x: scoreEvaluation(x))**

How to run:

- To play pacman: `python pacman.py`
- To run a certain agent using graphics use the following command: `python pacman.py -p AgentName`

Note: You are implementing a vanilla version of these AIs, being slow or going back and forth might be part of the algorithm behavior in the amount of time given for each move