################## 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 **gameEvaluation** as a heuristic/fitness function.

Notes:

- Python 2.7 is required to run the Framework.
- All your code must be inside the **pacmanAgents.py** file.
- **RandomAgent** and **RandomSequenceAgent** are implemented as example agents.
- External libraries are not allowed (as you won't submit them).
- Ways to **fail** the assignment if:
    - You try to change any of the system params.
    - Your code didn’t run (has errors).
    - You didn’t write the code yourself.
    - You submit anything beside **pacmanAgents.py** file.
    - You changed the name of the agent classes.
    - You implemented your own heuristic/fitness function.
- **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.

- You are only allowed to use these system functions (accessing/ changing any other functions or variables is considered cheating):
    - **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)
    - **gameEvaluation(startState, currentState):** evaluate the current state with respect to the beginning state
    - **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 executing your action sequence as the remaining actions can't be executed.
    - Always return the first action from the sequence with the highest **gameEvaluation**.
- 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.
- Use rank selection to select chromosomes
    - Sort all chromosomes based on fitness.
    - Give each chromosome a rank (from 1(worst) to length of chromosomes (best)).
    - Select chromosomes proportionally to their ranking
    - For example:
        - Assume we have 4 chromosomes (they have ranks 4,3,2,1 where 4 is better than 1)
        - The total of the rankings is 10 = 4 + 3 + 2 + 1.
        - Based on that, the probability to pick each chromosome are 4/10, 3/10, 2/10, 1/10.
        - This is the probability that each chromosome is selected.
    - Extra reading:
        - You can have a description of the Rank Selection process here (http://www.obitko.com/tutorials/genetic-algorithms/selection.php).
        - For more details check the roulette wheel selection and use the rank instead of fitness (https://en.wikipedia.org/wiki/Fitness_proportionate_selection).
- For generating the next population:
    - Use Rank Selection for picking each pair
    - Apply a random test, If the test result is less (or equal) to 70%, the pair will generate two children by crossing-over.
    - The crossover process is:
        - Let's suppose we have a chromosome X and a

chromosome Y.

- Let's call by XY' the new chromosome.
- For each gene (action) in the XY' sequence, we do a random test.
- If the result of the test is below 50%, the gene will be donated by X. Otherwise by Y
- Otherwise (test result bigger than 70%), keep the pair in the next generation.

- After you get all the new chromosomes of the new population, apply a random test for each chromosome:
  - if the test result is less (or equal) to 10%, mutate the chromosome by random choice. Choose a random action in the chromosome sequence and replace it by another action, chose it randomly.
- The whole population is replaced using crossover and mutated chromosomes, and some of them have a chance to survive

- For **MCTS**:
  - Use **gameEvaluation** to evaluate the game states at the end of the rollout phase.
  - 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).

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.