Artificial Intelligence
AIC_4301C
Lab 3
2022-2023

The code and a report of this Lab have to be submitted in Balckboard before 22/04/2023 23:00.

1 Tests

1. Create a folder Lab3_AIC_4301C_names-of-the-group-members
2. Download the project **multiagent_AIC.zip** from Blackboard, unzip it. In the folder **multiagent_AIC** you will find the following python files:
   (a) multiAgents.py: Where all of your multi-agent search agents will reside.
   (b) pacman_AIC.py: The main file that runs Pacman games.
   (c) game.py: The logic behind how the Pacman world works.
   (d) util.py: Useful data structures for implementing search algorithms.
   (e) You can ignore all other .py files.
3. In the folder multiagent_AIC execute python3 pacman_AIC.py and use the arrow keys to move.
4. Execute python3 pacman_AIC.py -p ReflexAgent
5. Execute python3 pacman_AIC.py -p ReflexAgent -l testClassic
6. Inspect the code in multiAgents.py.
7. Somme useful options to run the pacman are:
   - use the option -n to play n games.
   - use the option -q to turn off graphics.
   - use -f to run ghosts with a fixed random seed.
   - use -g DirectionalGhost to play with non random ghosts.
   - use -k to play with k ghosts.
   - python pacman_AIC.py -h to see the list of possible options.

2 Reflex Agent

1. Improve the **ReflexAgent** in **multiAgents.py** by proposing an evaluation function in def evaluationFunction(self, currentGameState, action) with (read points from (a) to (d) before starting the code):
   (a) The evaluation function of the reflex agent have to consider both food locations and ghost locations to perform well.
   (b) Remember that newFood has the function asList().
   (c) Your evaluation function evaluates state-action pairs.
   (d) Pacman is always agent 0.

2. Test your code:

   - Your agent should easily clear the **testClassic** layout:
     python3 pacman_AIC.py -p ReflexAgent -l testClassic
   - Test your reflex agent on the default **mediumClassic** layout with one or two ghosts and animation off to speed up the display using the following commands:
     python3 pacman_AIC.py –frameTime 0 -p ReflexAgent -k 1
     python3 pacman_AIC.py –frameTime 0 -p ReflexAgent -k 2
   - You can do this part after the first session of the lab. Run 20 tests for the layout contestCalssic for all the combination of the following configurations (a, c), (a,d), (a, e), (b, c), (b,d), (b, e) with (a) one ghost, (b) two ghosts, (c) random ghosts, (d) random ghosts with fixed seed, (e) non random ghost.
     Comment your results in the report.

3 Minimax

1. Write an adversarial search agent in the provided **MinimaxAgent** class in **multiAgents.py** with (read points from (a) to (i) before starting the code):

   (a) Your minimax agent should work with any number of ghosts, i.e. your minimax tree will have multiple min layers (one for each ghost) for every max layer.

   (b) Your code should expand the game tree to an arbitrary depth.

   (c) Score the leaves of your minimax tree with the supplied self.evaluationFunction, which defaults to scoreEvaluationFunction.

   (d) Make sure your minimax code makes reference to self.depth and self.evaluationFunction. You can use them because MinimaxAgent extends MultiAgentSearchAgent.

   (e) A single search ply is considered to be one Pacman move and all the ghosts' responses, so depth 2 search will involve Pacman and each ghost moving two times.

   (f) The correct implementation of minimax will lead to Pacman losing the game in some tests. This is not a problem.

   (g) The evaluation function for the Pacman test in this part is already written (self.evaluationFunction).

   (h) All states in minimax should be GameStates, either passed in to getAction or generated via GameState.generateSuccessor.

   (i) **Idea: Write two recursive methods in class Minimax(MultiAgentSearchAgent), def MAX_VALUE(self, gamestate, d): and def MIN_VALUE(self, gamestate, d, indexagent): where d is the depth of the search.**
   The two methods should return the value (evaluation function of the state gamestate) and the action from this state that leads to this value. You should also write a code in getAction(self, gameState).

2. Test your code:
   python3 pacman_AIC.py -p MinimaxAgent -a depth=5 -l smallClassic

3. You can do this part after the first session of the lab. Test MinimaxAgent using the same tests used for the ReflexAgent but with a variation of the depth (2, 3, 4) and make a comparison between MinimaxAgent and ReflexAgent.

4 Alpha Beta Algorithm

Make a new agent that uses alpha-beta pruning in **AlphaBetaAgent**.

1. Test your code:

   python3 pacman_AIC.py -p AlphaBetaAgent -a depth=5 -l smallClassic

2. You can do this part after the first session of the lab. Test AlphaBetaAgent using the same tests used for the MinimaxAgent and make a comparison between MinimaxAgent and AlphaBetaAgent.

5 Expectimax

Minimax and alpha-beta both assume that you are playing against an adversary who makes optimal decisions. Implement the **ExpectimaxAgent**, which is useful for modeling probabilistic behavior of agents who may make suboptimal choices by considering ghosts as chance nodes.

Test ExpectimaxAgent using the same tests used for the MinimaxAgent and make a comparison between MinimaxAgent and ExpectimaxAgent.