Artificial Intelligence
AIC_4301C
TP2
2021-2022

1. This project uses python 3.
2. You should execute your code in a linux environment by executing the command on a terminal.

1 Tests

1. Create a folder TP2_AIC_4301C_names-of-the-group-members
2. Download the project **multiagent_AIC.zip** from Blackboard, unzip it.
3. In the folder multiagent_AIC execute python3 pacman_AIC.py and use the arrow keys to move.

2 Python Files

In the folder **multiagent_AIC** you will find the following python files:

1. multiAgents.py: Where all of your multi-agent search agents will reside.
2. pacman_AIC.py: The main file that runs Pacman games. This file describes a Pacman GameState type, which you use in this project.
3. game.py: The logic behind how the Pacman world works. This file describes several supporting types like AgentState, Agent, Direction, and Grid.
4. util.py: Useful data structures for implementing search algorithms. We encourage you to look through util.py for some data structures that may be useful in your implementations.
5. You can ignore all other .py files.

3 Starting

1. Execute python3 pacman_AIC.py -p ReflexAgent
2. Execute python3 pacman_AIC.py -p ReflexAgent -l testClassic
3. Inspect the code of ReflexAgent (in multiAgents.py) and make sure you understand what it is doing.

4 Reflex Agent

1. Improve the **ReflexAgent** in **multiAgents.py** by proposing an evaluation function in def evaluationFunction(self, currentGameState, action) with:
   - The evaluation function of the reflex agent will have to consider both food locations and ghost locations to perform well.
   - Remember that newFood has the function asList().
   - Your evaluation function evaluates state-action pairs, in later parts of the project, you will evaluate states.

- Pacman is always agent 0, and the agents move in order of increasing agent index.

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 ghost or two (and animation off to speed up the display):
     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 TP if you want. Run 100 tests for every layout in the folder Layouts 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.
     - to play multiple games use the option -n.
     - to turn off graphics use the option -q to run lots of games quickly.
     - use -f to run gosths with a fixed random seed.
     - use -g DirectionalGhost to play with non random ghosts.
     - use -k to play with k ghosts.

     Comment your results.

5 Minimax

1. Write an adversarial search agent in the provided **MinimaxAgent** class in **multiAgents.py** with:

   (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) Implement the algorithm recursively.

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

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

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

2. Test your code:

   python pacman.py -p MinimaxAgent -a depth=5 -l smallClassic

3. You can do this part after the first session of the TP if you want. Test MinimaxAgent using the same tests used for the ReflexAgent but with a variation of the depth (1, 2, 3, 4) to see the behavior of the algorithm using different depths. Comment your results.

6 Alpha Beta Algorithm

Make a new agent that uses alpha-beta pruning in **AlphaBetaAgent**. Part of the challenge is to extend the alpha-beta pruning logic appropriately to multiple minimizer agents.

1. Test your code:

   python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic

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

7 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.

8 Evaluation Function

Write a better evaluation function for pacman in the provided function betterEvaluationFunction. The evaluation function should evaluate states, rather than actions like your reflex agent evaluation function did.

Make a comparison between MinimaxAgent using the betterEvaluationFunction and MinimaxAgent using the default evaluation function scoreEvaluationFunction.