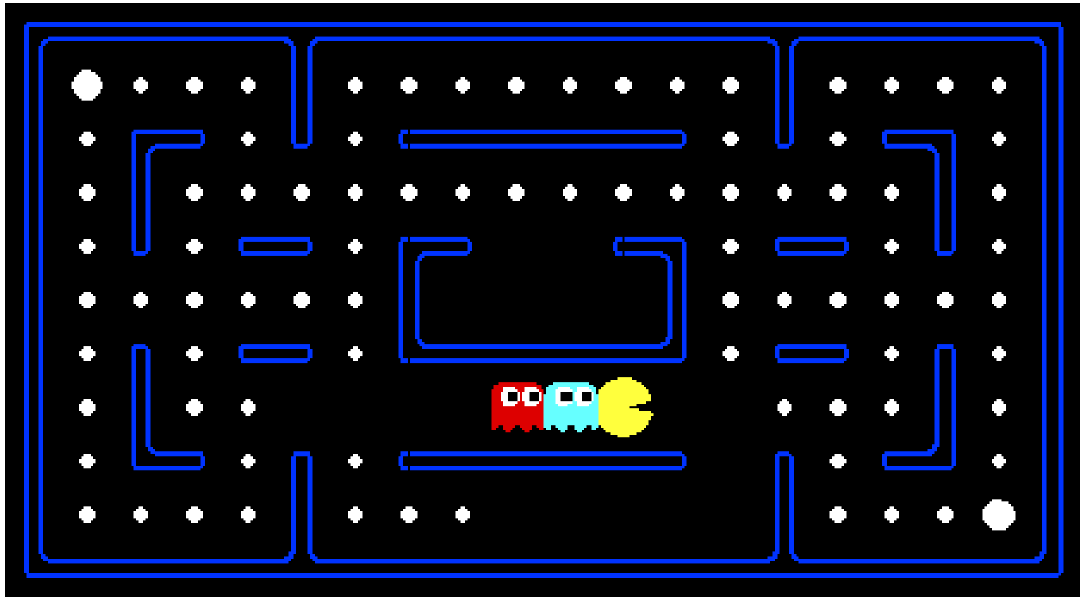
Introduction to Artificial Intelligence

Homework 3: Multi-Agent Search

TA 周千貿

PAC-MAN



- It’s a popular old game.
- You control pacman moves around in a maze.
- How to win: eats all the food in a maze.
- How to Lose: pacman touch the ghost.
- Special rule: the big white dots are capsules, which give pacman power to eat ghosts in a limited time.

Welcome to Multi-Agent Pac-Man

- The code base of Pac-Man was developed at UC Berkeley. (https://inst.eecs.berkeley.edu/~cs188/sp21/project2/)

- You can only execute it on a local machine.
- Google Colab cannot execute it because it has GUI.
- Please install python 3 on your own machine and be familiar with run code with CLI.

Welcome to Multi-Agent Pac-Man

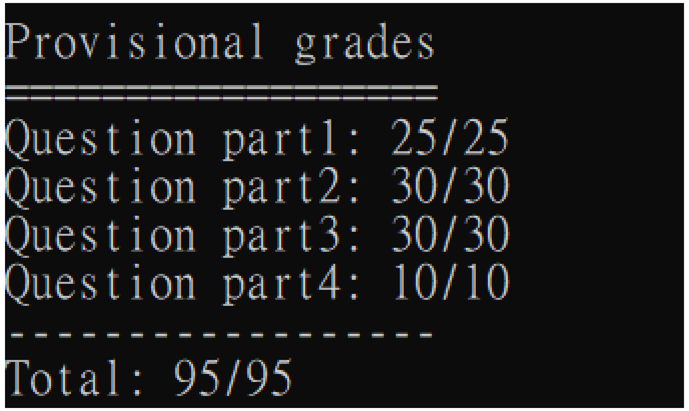
- Play a game and using the arrow keys to move:
  ```
  python pacman.py
  ```
- Play a game and using the provided ReflexAgent in multiAgents.py:
  ```
  python pacman.py -p ReflexAgent
  ```
- Play a game with different layout:
  ```
  python pacman.py -p ReflexAgent -l testClassic
  ```
- Other Options:
  - Default ghosts are random. you can also play for fun with slightly smarter directional ghosts using `-g DirectionalGhost`.
  - Play multiple games in one command with `-n`.
  - Turn off graphics with `-q` to run games quickly.
  - Use `-h` to know more options.

The file in the code base

Files you will edit:	
multiAgents.py	Where all of your multi-agent search agents will reside.
Files you might want to look at:	
pacman.py	The main file that runs Pac-Man games. This file also describes a pacman GameState type, which you will use extensively in this assignment.
game.py	The logic behind how the Pac-Man world works. This file describes several supporting types like AgentState, Agent, Direction, and Grid.
util.py	Useful data structures for implementing search algorithms. You don’t need to use these for this assignment, but may find other functions defined here to be useful.
Other files you might want to look at, if you are interested in the details of this game.	

Autogrding

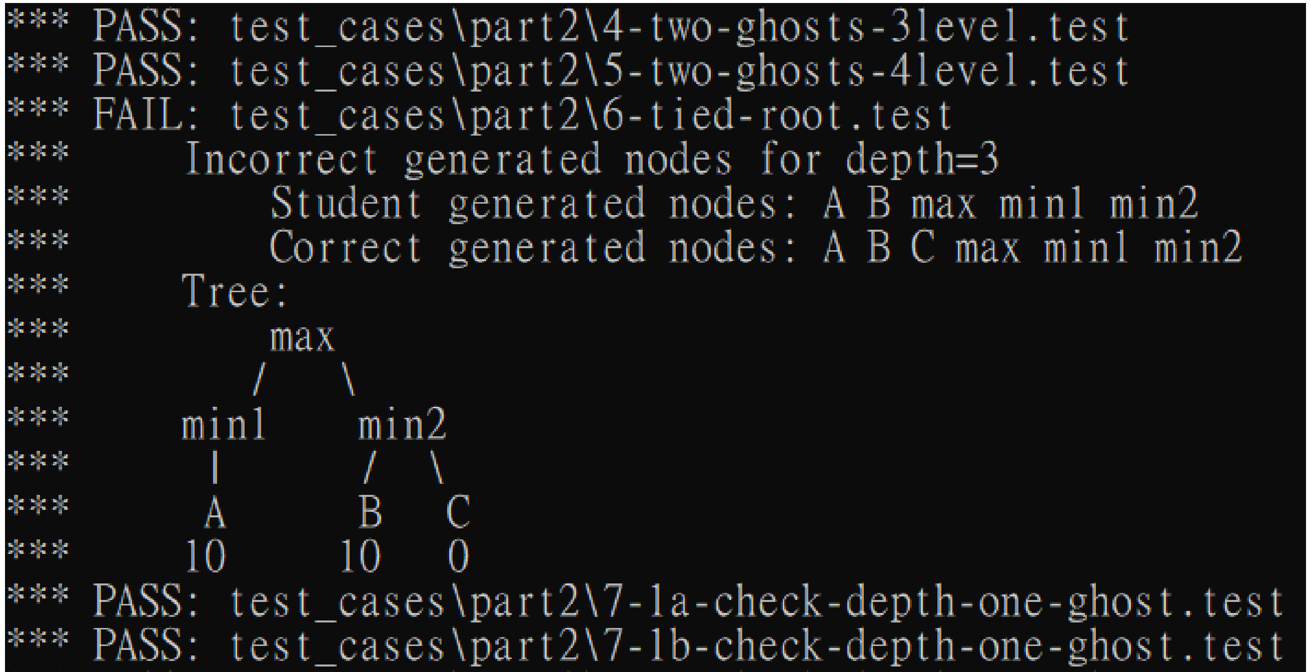
- TAs will use an autograder to grade your implementation.
- What autograder will run on your implementation?
  - Some simulated search trees.
  - Some Pac-Man games.
- The autograder will check your code to determine whether it explores the correct number of game states.
- After tests, It will show the score you will get.



```
Provisional grades
==================
Question part1: 25/25
Question part2: 30/30
Question part3: 30/30
Question part4: 10/10
------------------
Total: 95/95
```

Autogrding

- The autograder has been included in the code base. You can use the following command to test by yourself:
  `python autograder.py`
- Using the autograder to debug is recommended and will help you to find bugs quickly.



```
*** PASS: test_cases\part2\4-two-ghosts-3level.test
*** PASS: test_cases\part2\5-two-ghosts-4level.test
*** FAIL: test_cases\part2\6-tied-root.test
***     Incorrect generated nodes for depth=3
***         Student generated nodes: A B max min1 min2
***         Correct generated nodes: A B C max min1 min2
***     Tree:
***         max
***        /   \
***     min1    min2
***      |      /  \
***      A     B    C
***     10     10   0
*** PASS: test_cases\part2\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\part2\7-1b-check-depth-one-ghost.test
```

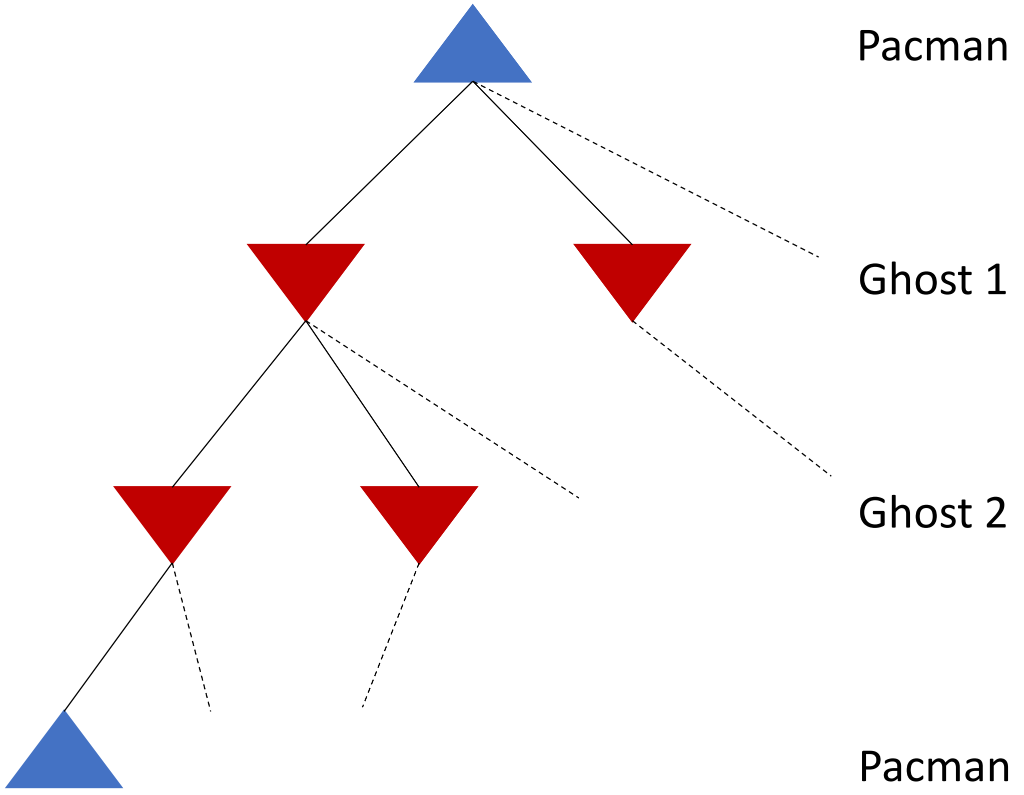
Autogrding

- To test and debug your code for one particular part, run the following command:
  ```
  python autograder.py -q part1
  ```
- To run it without graphics, use the following command:
  ```
  python autograder.py -q part1 --no-graphics
  ```

Requirements

- Please modify the codes in multiAgents.py between # Begin your code and # End your code.
- In addition, do not import other packages.
- All agents you will implement should work with any number of ghosts.
- In particular, your search tree will have multiple min/chance layers (one for each ghost) for every max layer.

EX: Two ghosts

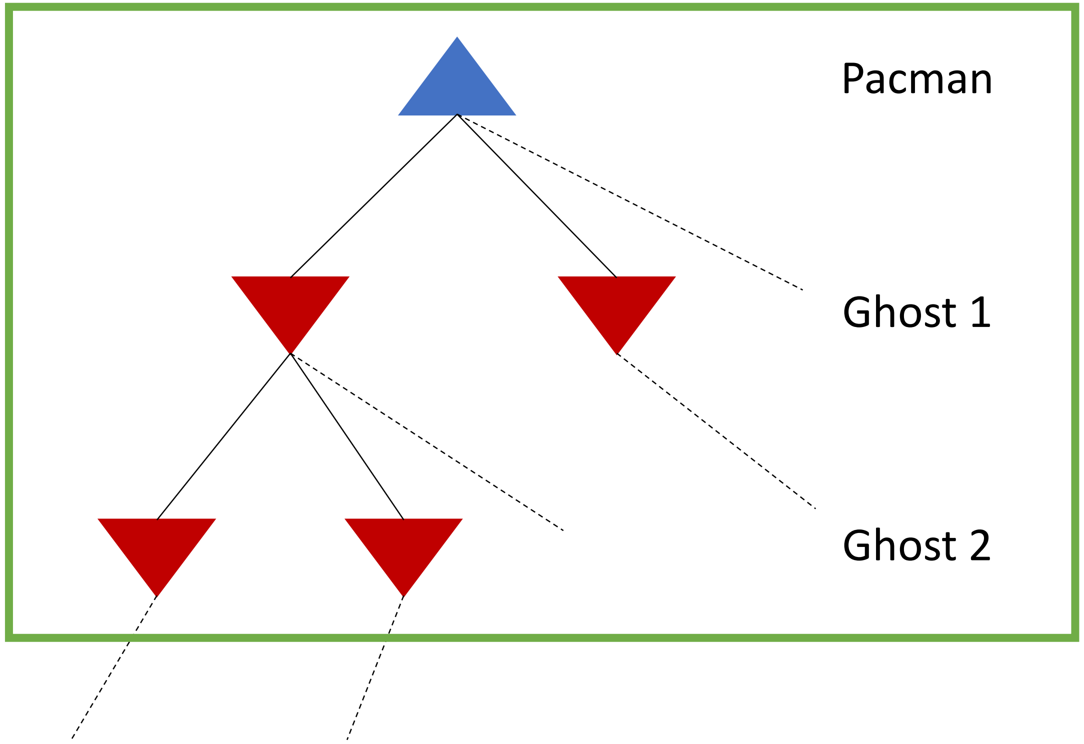


Requirements

- Your code should also expand the game tree to arbitrary depth with the supplied self.depth.

- A single level of the search is considered to be one pacman move and all the ghosts' responses.

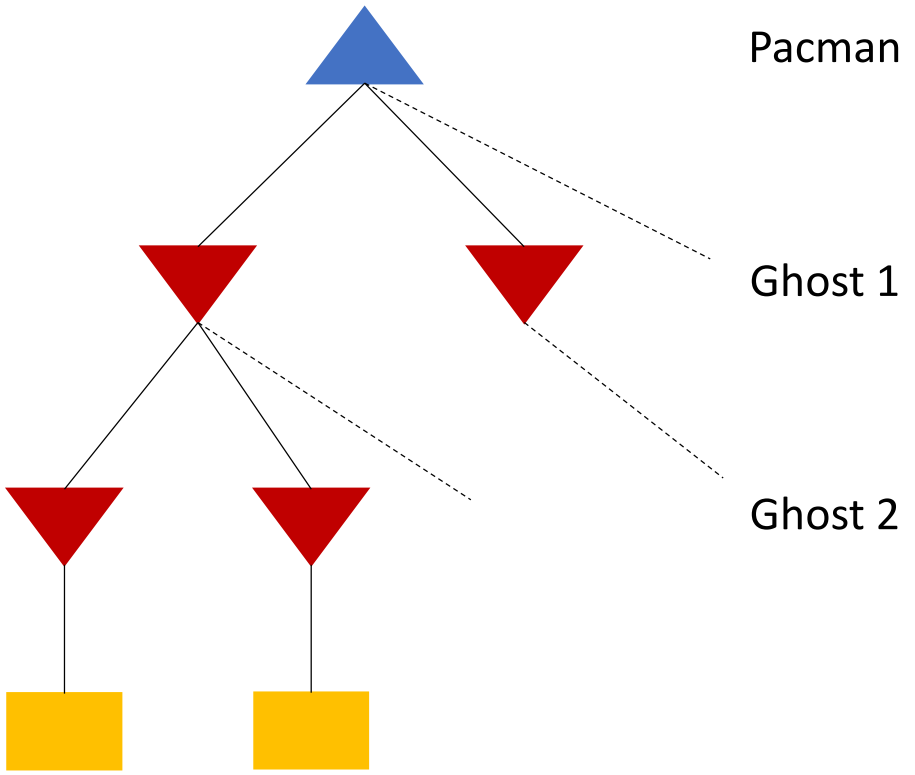
EX: One level for two ghosts



Requirements

- Your code should score the leaves of your search tree with the supplied self.evaluationFunction, which defaults to scoreEvaluationFunction.

EX: Two ghosts and depth = 1

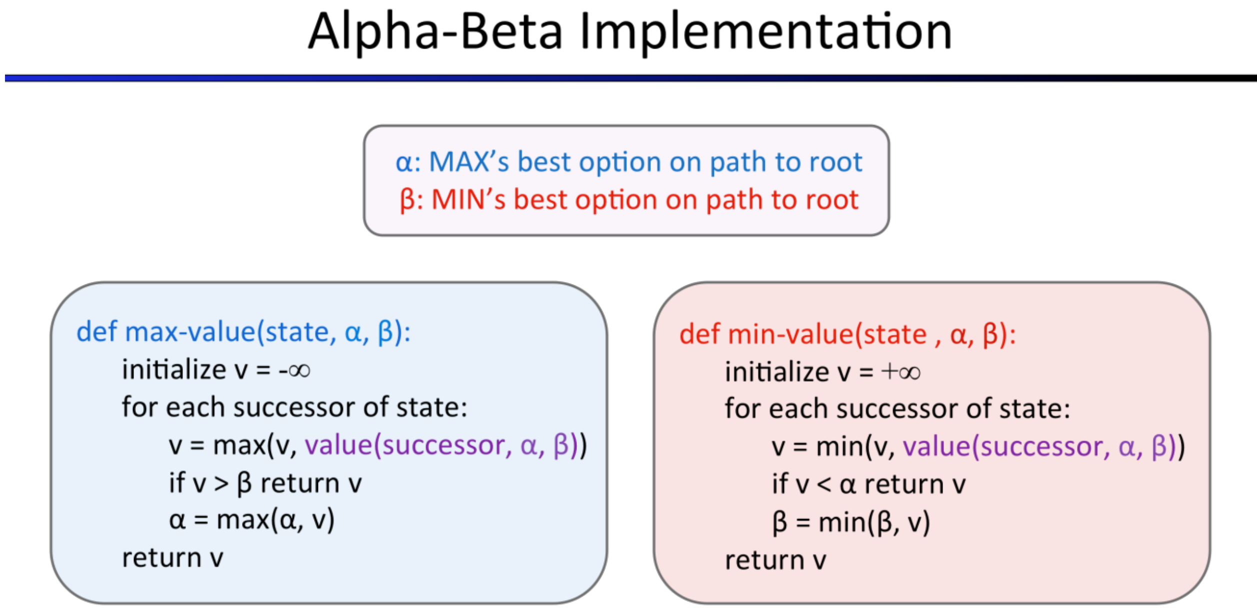


Part 1: Minimax Search (25%)

- Write an adversarial search agent in the provided MinimaxAgent class stub in multiAgents.py.
- The actual ghosts operating in the environment may act partially randomly.
- But the minimax algorithm assumes the worst.

Part 2: Alpha-Beta Pruning (30%)

- Make a new agent that uses alpha-beta pruning to more efficiently explore the minimax tree in AlphaBetaAgent class in multiAgents.py.
- You must not prune on equality in order to match the set of states explored by our autograder.
- The pseudo-code represents at right side you should implement for this part.



Alpha-Beta Implementation

α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v > β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v < α return v
        β = min(β, v)
    return v
```

Part 3: Expectimax Search (30%)

- Implement the ExpectimaxAgent class in multiAgents.py, which is useful for modeling probabilistic behavior of agents who may make suboptimal choices.
- Rather than taking the min over all ghost actions, expectimax agent will take the expectation according to your agent's model of how the ghosts act.
- To simplify your code, assume you will only be running against an adversary that chooses among its legal actions uniformly at random.

Part 4: Evaluation Function (Bonus) (10%)

- Write a better evaluation function for pacman in the provided function betterEvaluationFunction in multiAgents.py.
- The evaluation function should evaluate only states not including actions.
- What you can use for evaluation?
  - Where is the food?
  - Where is the ghosts?
  - …

Part 4: Evaluation Function (Bonus) (10%)

Grading:

- The autograder will run your agent on the smallClassic layout 10 times.
- We will assign points to your evaluation function in the following way:
  - If you win at least once without timing out the autograder, you get 1 point. Any agent not satisfying these criteria will receive 0 points.
  - +1 for winning at least 4 times, +2 for winning at least 7 times, +3 for winning all 10 times.
  - +2 for an average score of at least 500, +4 for an average score of at least 1000 (including scores on lost games)
  - +1 for no timeout at least 5 times, +2 for no timeout all 10 times.
- The autograder will be run on the same machine with --no-graphics.

Report (15%)

- A written report is required.
- The report should be written in English.
- Save the report as a .pdf file.
- For part 1 ~ 4, please take some screenshots of your code and explain how you implement codes in detail.
- Describe problems you meet and how you solve them.

Important Rules

- **Due Date: 2021/5/7 23:55**

- **Submission**
  - Please prepare your multiAgents.py and report (.pdf) into STUDENTID_hw3.zip.

- **Late Submission Policy**
  - 20% off per late day

Reminders

- More detail will be in the homework document.
- If there are any updates or problems of the homework, we will announce on E3.
- If you have any questions for homework please mail me.
    - TA 周千貿: ya11235813@gmail.com
    - TA 胡瑞麟: linhu.cs09g@nctu.edu.tw
    - TA 張桂華: amy09921@gmail.com