COSE361(03) - Final project 2/2

Due Date: 2023/6/14(Wednesday) 23:59

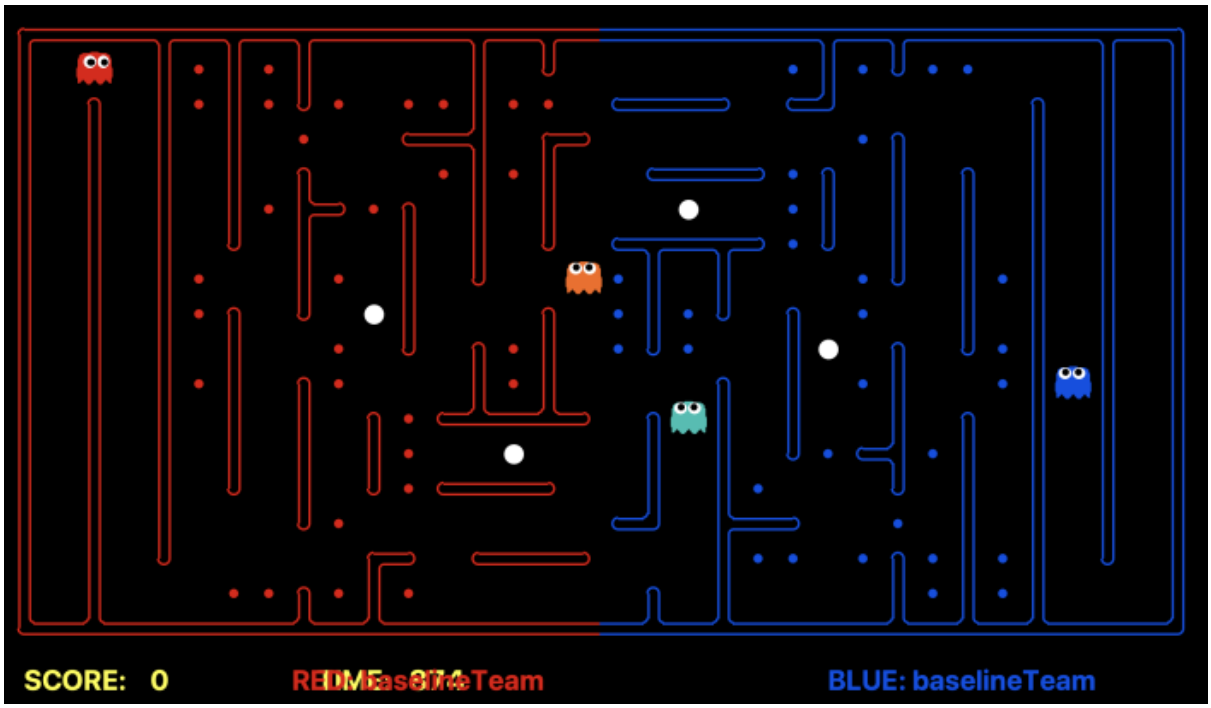
Python Version: 3.6.

All of the source code is from CS188 Berkeley.
(https://inst.eecs.berkeley.edu/~cs188/sp20/minicontest1/)

Submission policy is at the end of this PDF.

Overview

This mini-contest involves a multi-player capture-the-flag variant of Pacman, where agents control both Pacman and ghosts in coordinated team-based strategies. Your team will try to eat the food on the far side of the map while defending the food on your home side.



Grading

Base credit(10)

Your agent will be tested against the baseline on several selected maps in layouts

- 5 points for under 50% winning rate against the provided baseline agent in `baseline.py`.
- 10 points for 51% ~ 100% winning rate against the provided baseline agent in `baseline.py`.

Report (15)
Check "Submission" for details

Extra Credit(5)

Students that perform well in the **final** submission will receive the following extra credit:

- 1st to 10th place: 5 points
- 11st to 20th place: 4 points
- 21st to 30th place: 3 points
- 31st to 40th place: 2 points
- 41st to 50th place: 1 points

> 💡 ***The mini-contest is a round robin tournament, where each student meets all the other participants and competes five times. The points for each match are as follows:***
> ***Win : 3 pt***
> ***Tie : 1 pt***
> ***Lose : 0 pt***
> ***More details in Ranking system section.***

Introduction

The primary change between the first and second mini-contests is that mini-contest 2 is an adversarial game, involving two teams competing against each other. Your

team will try to eat the food on the far side of the map while defending the food on your home side

Your agents are in the form of ghosts on your home side and Pacman on your opponent's side. Also, you are now able to **eat your opponent when you are a ghost**. If a Pacman is eaten by a ghost before reaching his own side of the board, he will explode into a cloud of food dots that will be deposited back onto the board.

Files to Edit and Submit: You will fill and submit `your_best.py`, `your_baseline1.py`, `your_baseline2.py`, `your_baseline3.py` .
You should change the file name `your_best.py` to `{Student_ID}.py`.

Please do not change the other files in this distribution or submit any of our original files other than this file.

External libraries: In this contest, you are allowed to use numpy as a dependency

Academic Dishonesty: We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; please don't let us down. If you do, we will pursue the strongest consequences available to us.

Discussion: Please be careful not to post spoilers.

Rules

Layout

The Pacman map is now divided into two halves: blue (right) and red (left). Red agents (which all have even indices) must defend the red food while trying to eat the blue food. When on the red side, a red agent is a ghost. When crossing into enemy territory, the agent becomes a Pacman.
There are a variety of layouts in the `layouts` directory.

Scoring

As a Pacman eats food dots, those food dots are stored up inside of that Pacman and removed from the board. When a Pacman returns to his side of the board, he “deposits” the food dots he is carrying, earning one point per food pellet delivered. Red team scores are positive, while Blue team scores are negative. If Pacman gets eaten by a ghost before reaching his own side of the board, he will explode into a cloud of food dots that will be deposited back onto the board.

Power Capsules

If Pacman eats a power capsule, agents on the opposing team become “scared” for the next 40 moves, or until they are eaten and respawn, whichever comes sooner. Agents that are “scared” are susceptible while in the form of ghosts (i.e. while on their own team’s side) to be eaten by Pacman. Specifically, if Pacman collides with a “scared” ghost, Pacman is unaffected and the ghost respawns at its starting position (no longer in the “scared” state).

Observations

Each agent can see the **entire state** of the game, such as food pellet locations, all Pacman locations, all ghost locations, etc. See the GameState section for more details.

Winning

In this adversarial game, a team wins when they return all but two of the opponents’ dots. Games are also limited to 1200 agent moves (moves can be unequally shared depending on different speeds - faster agents get more moves). If this move limit is reached, whichever team has returned the most food wins.

If the score is zero (i.e., tied), the game is recorded as a tie.

Computation Time

Each agent has 1 second to return each action. Each move that does not return within one second will incur a warning. After three warnings, or any single move taking more than 3 seconds, the game is forfeited. There will be an initial start-up allowance of 15 seconds (use the `registerInitialState` function).

Designing Agents

An agent now has the more complex job of trading off offense versus defense and effectively functioning as both a ghost and a Pacman in a team setting. The added time limit of computation introduces new challenges.

Baseline Team

To kickstart your agent design, we have provided you with a team of two baseline agents, defined in `baseline.py`. They are quite bad.

The `OffensiveReflexAgent` simply moves toward the closest food on the opposing side. The `DefensiveReflexAgent` wanders around on its own side and tries to chase down invaders it happens to see.

Interface

The `GameState` in `capture.py` should look familiar, but contains new methods like `getRedFood`, which gets a grid of food on the red side (note that the grid is the size of the board, but is only true for cells on the red side with food). Also, note that you can list a team's indices with `getRedTeamIndices`, or test membership with `isOnRedTeam`.

Distance calculation

To facilitate agent development, we provide code in `distanceCalculator.py` to supply shortest path maze distances.

CaptureAgent Methods

To get started designing your own agent, we recommend subclassing the `CaptureAgent` class. This provides access to several convenience methods. Some useful methods are:

`getFood(self, gameState)`

Returns the food you're meant to eat. This is in the form of a matrix where `m[y]=True` if there is a food you can eat (based on your team) in that square.

`getFoodYouAreDefending(self, gameState)`

Returns the food you're meant to protect (i.e., that your opponent is supposed to eat). This is in the form of a matrix where `m[y]=True` if there is food at `(x,y)` that your opponent can eat.

`getOpponents(self, gameState)`
Returns agent indices of your opponents. This is the list of the numbers of the agents (e.g., red might be `[1,3]`).

`getTeam(self, gameState)`
Returns agent indices of your team. This is the list of the numbers of the agents (e.g., blue might be `[1,3]`).

`getScore(self, gameState)`
Returns how much you are beating the other team by in the form of a number that is the difference between your score and the opponents' score. This number is negative if you're losing.

`getMazeDistance(self, pos1, pos2)`
Returns the distance between two points; These are calculated using the provided distancer object. If `distancer.getMazeDistances()` has been called, then maze distances are available. Otherwise, this just returns Manhattan distance.

`getPreviousObservation(self)`
Returns the `GameState` object corresponding to the last state this agent saw (the observed state of the game last time this agent moved).

`getCurrentObservation(self)`
Returns the `GameState` object corresponding this agent's current observation (the observed state of the game).

`debugDraw(self, cells, color, clear=False)`
Draws a colored box on each of the cells you specify. If `clear` is `True`, this function will clear all old drawings before drawing on the specified cells. This is useful for debugging the locations that your code works with. `color` is a list of RGB values between 0 and 1 (i.e.`[1,0,0]` for red), `cells` is a list of game positions to draw on (i.e.`[(20,5), (3,22)]`)

Restrictions

You are free to design any agent you want. However, you will need to respect the provided APIs if you want to participate in the contest. Agents who compute during the opponent's turn will be disqualified. In particular, any form of multithreading is disallowed, because we have found it very hard to ensure that no computation takes place on the opponent's turn.

Please respect the APIs and keep all of your implementation within `your_best.py` .

Getting Started

By default, you can run a game with the simple `baseline` that the staff has provided:

```
$ python capture.py
```

A wealth of options are available to you:

```
$ python capture.py --help
```

There are four slots for agents, where agents 0 and 2 are always on the red team, and 1 and 3 are on the blue team. Agents are created by agent factories (one for Red, one for Blue). See the section on designing agents for a description of the agents invoked above. The only team that we provide is the `baseline`. It is chosen by default as both the red and blue team, but command-line options allow you to choose teams:

```
$ python capture.py -r baseline -b baseline
```

In this case, we specify that the red team `-r` and the blue team `-b` are both created from `baseline.py`. To control one of the four agents with the keyboard, pass the appropriate option:

```
$ python capture.py --keys0
```

The arrow keys control your character, which will change from ghost to Pacman when crossing the center line.

Layouts

By default, all games are run on the `defaultcapture` layout. To test your agent on other layouts, use the `-l` option. In particular, you can generate random layouts by

specifying `RANDOM[seed]`. For example, `-l RANDOM13` will use a map randomly generated with seed 13.

Recordings

You can record local games using the `--record` option, which will write the game history to a file named by the time the game was played. You can replay these histories using the `--replay` option and specifying the file to replay.

Final-Project Dry-Run

> 💡 ***There will be a Dry-Run on June 6th. Only students who submit 'studentID.py' before the dry-run will be given intermediate ranks. Dry-Run has nothing to do with the final score.***

There are two files you need to submit.

1. Submit `your_best.py` on Blackboard.
    - You should change `your_best.py` to `Student_ID.py`

      ```
      $ mv your_best.py -> {Student_ID}.py

      # eg)
      # mv your_best.py 201911111.py
      ```

    - Final Submission files:

      `201911111.py`

    > 💡 ***Note. If the submitted file name is incorrect, it will not be scored without any room for mercy.***

Submission

There are two files you need to submit.

1. Submit `your_baseline1.py`, `your_baseline2.py`, `your_baseline3.py`, `your_best.py` on Blackboard.

   - You should change `your_best.py` to `Student_ID.py`

     ```
     $ mv your_best.py {Student_ID}.py

     # eg)
     # mv your_best.py 201911111.py
     ```

   - Final Submission files:

     `your_baseline1.py`, `your_baseline2.py`, `your_baseline3.py`, `201811111.py`

   > 💡 ***Note. If the submitted file name is incorrect, it will not be scored without any room for mercy.***

2. Submit a pdf file containing:

   We expect 2-page(or more, if you need) ***'LaTex format' reports*** and ***csv file***.

   1) Run `capture.py`

      - Running `capture.py` file will return `output.csv` file
      - Number of simulation for grading : 10 times

        ```
        $ python capture.py -r {Student_ID} -n 10

        # eg)
        # python capture.py -r 201911111 -n 10
        ```

	your_best(red)
	<Average Winning Rate>
your_base1	0.1
your_base2	-0.4
your_base3	0.3
baseline	-0.8
Num_Win	2
Avg_Winning_Rate	-0.2
	<Average Scores>
your_base1	10
your_base2	5
your_base3	3
baesline	-14
Avg_Score	1

2) *LaTex* Report including:

- Introduction : brief description about the final project(2/2) and your implementation.
- Methods : Details about your implemented 3 agents.
- Results : Table with results of your agents.
- Conclusion & Free discussion : Ask & Answer your own question about the final project (2/2).

> 💡 ***Note. The LaTex format file (ICML format) will be uploaded in Blackboard.***

To check your understanding, please submit an explanation of your code with a pdf file or comment in the python files.

- Do not compress your result files. Upload them respectively.
- It is okay to make out a report in Korean.

Ranking System

All submitted Facman Agent competes with all other students through a league system.

For the sake of fairness, we score by taking into account all cases of wins, draws, losses, and errors (in code execution).

Details of the scoring rule are as follows:

1. Score of i^{th}match :

   $$s_i = \begin{cases} 3 & \text{win} \\ 1 & \text{draw} \\ 0 & \text{loss} \end{cases}$$

   If an error occurs in a match, that match is not scoring. However, the error rate is calculated and affects the overall score.

2. Final Score :

   $$s = \mathbb{E}_{i \in I}[s_i] \cdot (1 - \mathbf{1}(e \geq 0.1) \cdot (e - 0.1)) \cdot (1 - \mathbf{1}(e \geq 0.3))$$

   e denotes error rate.
   In other words, a mean score of total matches will be your final score.

> 💡 ***Note. An error case of less than 10% does not affect the score, and if an error rate of more than 10% or less than 30% is decayed to score. If the error rate is more than 30%, you get 0 points.***

About Plagiarism

We know that it is easy to find source code for this assignment. However, if we find out that you just copied from one of the source codes, the grade for the assignment will be zero. You can refer to those source codes, but do not just copy and paste them.

Q & A

If you have any questions, please upload your question on discussion board in Blackboard.