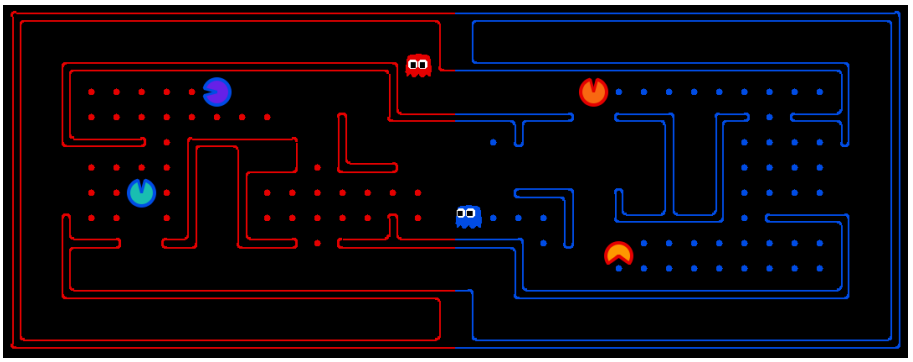
Contest: Pac-Man Capture the Flag



Enough of defense,
Onto enemy terrain.
Capture all their food!

Introduction

The course contest involves a multi-player capture-the-flag variant of Pac-Man, where agents control both Pac-Man 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. The contest code is available as a zip archive.

Key files to read:

capture.py	The main file that runs games locally. This file also describes the new capture the flag GameState type and rules.
pacclient.py	The main file that runs games over the network.
captureAgents.py	Specification and helper methods for capture agents.

Supporting files:

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.
distanceCalculator.py	Computes shortest paths between all maze positions.
graphicsDisplay.py	Graphics for Pac-Man
graphicsUtils.py	Support for Pac-Man graphics
textDisplay.py	ASCII graphics for Pac-Man
keyboardAgents.py	Keyboard interfaces to control Pac-Man
layout.py	Code for reading layout files and storing their contents

Academic Dishonesty: While we won't grade contests, we still expect you not to falsely represent your work. *Please* don't let us down.

Rules of Pac-Man Capture the Flag

Layout: The Pac-Man 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 Pac-Man.

Scoring: When a Pac-Man eats a food dot, the food is permanently removed and one point is scored for that Pac-Man's team. Red team scores are positive, while Blue team scores are negative.

Eating Pac-Man: When a Pac-Man is eaten by an opposing ghost, the Pac-Man returns to its starting position (as a ghost). No points are awarded for eating an opponent. Ghosts can never be eaten.

Winning: A game ends when one team eats all but two of the opponents' dots. Games are also limited to 3000 agent moves. If this move limit is reached, whichever team has eaten the most food wins.

Computation Time: Each agent has 1 second to return each action. Each move which 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 forfeit. There will be an initial start-up allowance of 15 seconds (use the `registerInitialState` function).

Observations: Agents can only observe an opponent's configuration (position and direction) if they or their teammate is within 5 squares (Manhattan distance). In addition, an agent always gets a noisy distance reading for each agent on the board, which can be used to approximately locate unobserved opponents.

Play Balancing: Over the semester we will be improving the game. Several likely changes are: (1) power pellets, (2) a start-up time allowance, and (3) ongoing level redesign.

Submission Instructions

To enter an agent into the nightly tournaments, create a subdirectory in the `teams` directory with the same name as your agent, and put the code for your agent in it. Then properly fill out `config.py` with your team name, agents, and other options, and place it in the directory along with the rest of your files. After this, you can `submit` under the assignment name `contest`. For your reference, we have provided a sample `config.py` configured for the `BaselineAgent`. The `BaselineAgent` directory itself is inside the `teams` directory. Make sure to pick a unique team name!

Getting Started

By default, you can run a four-agent game with simple `BaselineAgents` that the staff has provided:

```
python capture.py
```

A wealth of options are available to you:

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

There are six slots for agents, where agents 0, 2 and 4 are always on the red team and 1, 3 and 5 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 agents available now are the `BaselineAgents`. They are chosen by default, but as an example of how to choose teams:

```
python capture.py -r BaselineAgents -b BaselineAgents
```

which specifies that the red team `-r` and the blue team `-b` are `BaselineAgents`. To control an agent with the keyboard, pass the appropriate option to the red team:

```
python capture.py --redOpts first=keys
```

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

Game Types

You can play the game in three ways: local games, ad hoc network games, and nightly tournaments.

Local games (described above) allow you to test your agents against the baseline teams we provide and are intended for use in development.

Ad Hoc Network Games

In order to facilitate testing of your agents against others' in the class, we have set up game servers that moderate ad hoc games played over the network.

```
python pacclient.py
```

Teams are chosen similarly to the local version. See `python capture.py -h` for details. Any agent that works in a local game should work equivalently in an online game. Note that if you violate the per-action time limit in an online game, a move will be chosen for you on the server, but your computation will not be interrupted. Students in the past have struggled to understand multi-threading bugs that arise from violating the time limit (even if your code is single-threaded), so stay within the time limit!

Named Games

By default, when you connect to the server for a network game, you will be paired with the first unmatched

opponent that connects. If you would like to play with a buddy, you can organize a game with a specific name on the server:

```
python pacclient.py -g MyCoolGame
```

Which will pair you only with the next player who requests "MyCoolGame".

Coordinating With Other Teams

Finding an opponent for network games may be difficult because it requires someone else to be online and looking for a game at approximately the same time. It may be a good idea for your team to get together with other teams to compete or practice over the network. Alternately, you could run two separate instances of `python pacclient.py -g MyCoolGame` on a single computer, and play your agents against themselves.

Official Tournaments

The actual competitions will be run using nightly automated tournaments, with the final tournament deciding the final contest outcome. To enter an agent into the nightly tournaments, make sure to properly fill in `config.py` and then `submit` under the assignment name `contest`. Be sure to pick a unique name for your team. Tournaments are run everyday at midnight and include all teams that have been submitted (either earlier in the day or on a previous day) as of the start of the tournament. Currently, each team plays every other team in a best-of-3 match, but this may change later in the semester. The results are updated on the website after the tournament completes each night.

Designing Agents

Unlike project 2, an agent now has the more complex job of trading off offense versus defense and effectively functioning as both a ghost and a Pac-Man in a team setting. Furthermore, the limited information provided to your agent will likely necessitate some probabilistic tracking (like project 4). Finally, the added time limit of computation introduces new challenges.

Baseline Agents: To kickstart your agent design, we have provided you with two baseline agents. They are both quite bad. The `OffensiveReflexAgent` 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.

Directory Structure: You should place your agent code in a new sub-directory of the teams directory. You will need a `config.py` file, which specifies your team name, authors, agent factory class, and agent options. See the `BaselineAgents` example for details.

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`.

Finally, you can access the list of noisy distance observations via `getAgentDistances`. These distances are within 6 of the truth, and the noise is chosen uniformly at random from the range [-6, 6] (e.g., if the true distance is 6, then each of {0, 1, ..., 12} is chosen with probability 1/13). You can get the likelihood of a noisy reading using `getDistanceProb`.

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

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

```
def getFood(self, gameState):
  """
  Returns the food you're meant to eat. This is in the form
  of a matrix where m[x][y]=true if there is food you can
  eat (based on your team) in that square.
  """

def 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[x][y]=true if there is food at (x,y) that
  your opponent can eat.
  """

def getOpponents(self, gameState):
  """
  Returns agent indices of your opponents. This is the list
  of the numbers of the agents (e.g., red might be "1,3,5")
  """
```

```
def getTeam(self, gameState):
  """
  Returns agent indices of your team. This is the list of
  the numbers of the agents (e.g., red might be "1,3,5")
  """

def 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.
  """

def getMazeDistance(self, pos1, pos2):
  """
  Returns the distance between two points; These are calculated u
  distancer object.

  If distancer.getMazeDistances() has been called, then maze dist
  Otherwise, this just returns Manhattan distance.
  """

def 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 - this may not include all of your
  opponent's agent locations exactly).
  """

def getCurrentObservation(self):
  """
  Returns the GameState object corresponding this agent's
  current observation (the observed state of the game - this
  may not include all of your opponent's agent locations
  exactly).
  """
```

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 tournaments. Agents which compute during the opponent's turn will be disqualified. In fact, we do not recommend any sort of multi-threading.

Contest Details

The contest has two parts: a qualifying round and a final tournament.

- **Qualifying:** Every night, we will post the results of a round robin tournament among all submitted agents, including a qualifying "Staff Agents" team. To qualify for the final tournament, you must first submit your agents according to the instructions. Then, you must be ranked ahead of Staff Agents in one of the nightly tournaments before the qualification deadline. The earlier you submit your agents, the more chances you have to qualify!
- **Tournament:** (details subject to change) A final double-elimination tournament will be run in the basement of Soda hall on the evening before the last day of class (Wednesday 12/2). The final lecture the next day will include replays of important matches. The final tournament will be similar to the defaultCapture layout.

  Important dates (subject to change):

Monday	9/21	Contest announced and posted
Tuesday	11/10	Qualification opens
Thursday	11/26	Tournament layout revealed
Monday	11/30	Qualification closes
Wednesday	12/2	Final tournament
Thursday	12/3	Awards ceremony in class

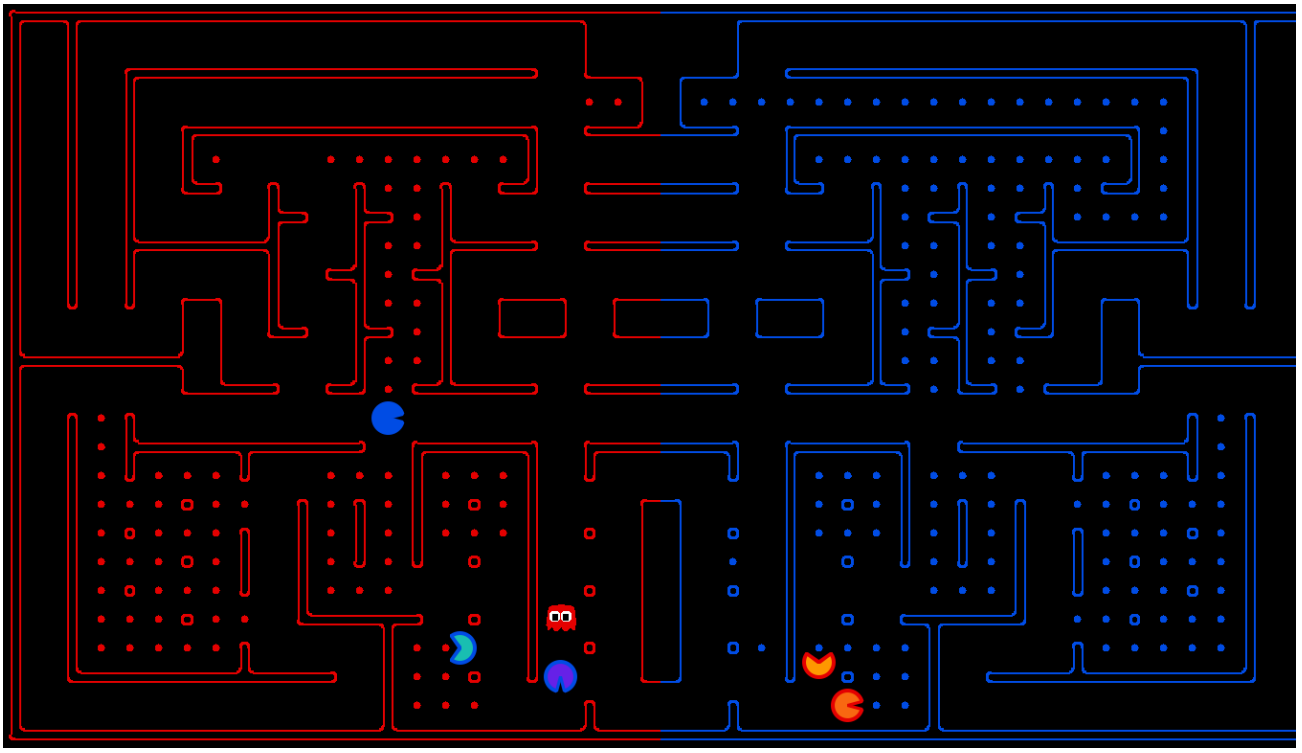
  Teams: You may work in teams of up to 5 people.

  Prizes: The top three teams will receive awards in class, including shiny medals and extra credit points. All teams that qualify for the final tournament will also receive extra credit points.

- First Place: 4% final exam point increase
- Second Place: 3% final exam point increase
- Third Place: 2% final exam point increase
- Qualifying: 1% final exam point increase

Acknowledgements

Many thanks to Jeremy Cowles for implementing the tournament infrastructure. Thanks to Barak Michener and Ed Karuna for providing online networking infrastructure, improved graphics and debugging help.



Have fun! Please bring our attention to any problems you discover.