Lab 6

Overview

This lab involves a multi-player capture-the-flag variant of Pacman, where agents control both Pacman and ghosts in coordinated team-based strategies. You will try to eat the food on the far side of the map, while defending the food on your home side. The code is available as a zip archive (lab6.zip).

You may choose to work alone or with one partner. There is room to bring your own unique ideas, and there is no single set solution. Much looking forward to seeing what you come up with!

Introduction

This project 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 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 you'll edit

File	Description
myTeam.py	What will be submitted. Contains all of the code needed for your agent.

Files you might want to look at

File	Description
capture.py	The main file that runs games locally. This file also describes the new capture the flag GameState type and rules.
captureAgents.py	Specification and helper methods for capture agents.
baselineTeam.py	Example code that defines two very basic reflex agents, to help you get started.

Supporting Files (Do not Modify)

File	Description
game.py	The logic behind how the Pacman 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 Pacman
graphicsUtils.py	Support for Pacman graphics
textDisplay.py	ASCII graphics for Pacman
keyboardAgents.py	Keyboard interfaces to control Pacman
layout.py	Code for reading layout files and storing their contents

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 layout 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 being 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) this is recorded as a tie game.

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).

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 `baselineTeam.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.

File Format

You should include your agents in a file of the same format as myTeam.py. Your agents must be completely contained in this one file.

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:

`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]`).

`def 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]`).

`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 using the provided distancer object. If `distancer.getMazeDistances()` has been called, then maze distances are available. 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).

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

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

Restrictions

You are free to design any agent you want. However, agents which compute during the opponent's turn will be disqualified. In particular, any form of multi-threading 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 `myTeam.py` .

Getting Started

By default, you can run a game with the simple baselineTeam:

```
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 baselineTeam. It is chosen by default as both the red and blue team, but as an example of how to choose teams:

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

which specifies that the red team -r and the blue team -b are both created from baselineTeam.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.

What to submit

1. Source code.
2. Please create a folder called "yourname_StudentID_Lab6" that includes all the required files and generate a zip file called "yourname_StudentID_Lab6.zip".
3. If pair working, save your work in a folder named "yourteamname_Lab6". Both members must submit the same folder.
4. Please submit your work (.zip) to Moodle.