Assignment III: Decision-making for Teams

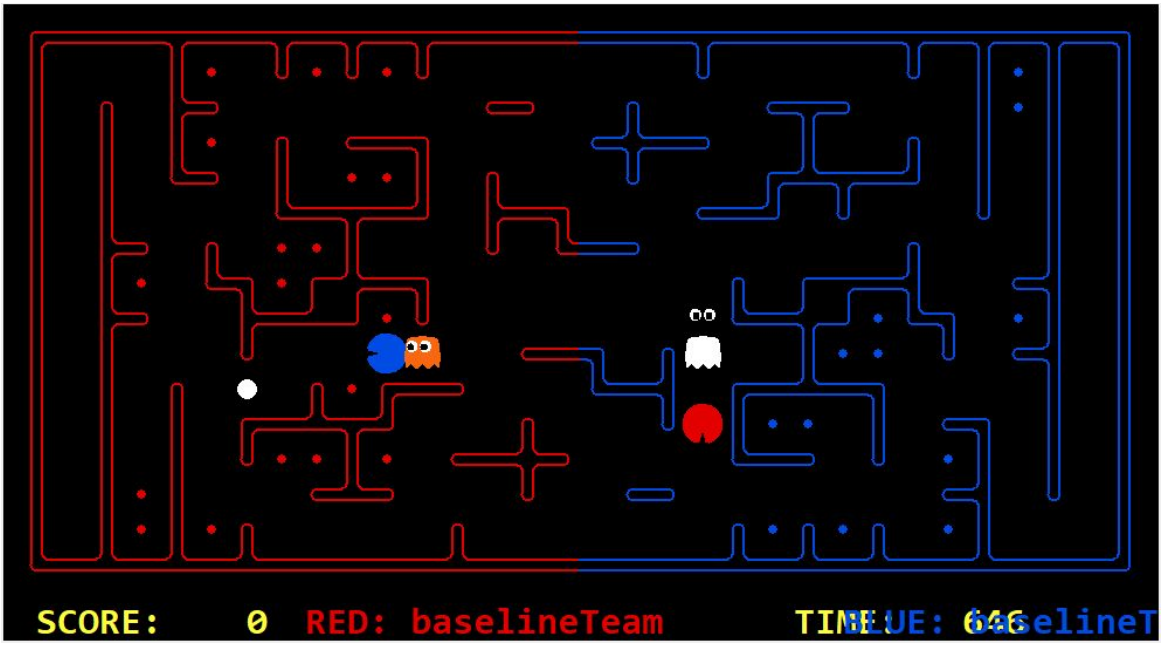
1. Overview

- This assignment explores planning and decision making for teams. The setup involves two teams where each team tries to safeguard its resources (food pellets) and tries to capture resources held by the other team. You will design AI agents that control Pacman and ghosts in coordinated team-based strategies.
- This assignment involves two teams (named *red* and *blue*) present in the 2D map 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. Each team (*red* and *blue*) consists of two agents each.
- Your agents are in the form of ghosts on your home side and Pacman on your opponent's side. If you are a ghost then you are permitted to consume the opponent. If Pacman is eaten by a ghost before reaching its own side of the board, he will explode into a cloud of food dots that will be deposited back on the board. Additionally, the map includes power capsules which will be explained later. Further, there is an added time limit of computation which will add new challenges.
- The AI agent must reason about trading off trying to capture resources versus trying to defend its own resources and effectively functioning both as a ghost and a Pacman in a team setting.

2. Getting Started

- Please download the starter code for the assignment from this link and create a fresh setup.
- As before, please use Python 3.6 for this assignment. You may use the same conda environment as for assignment 2, or you may create a new environment, following the instructions as in assignment 2.
- You will be placing all your implementation in **myTeam.py**. Please respect the API provided in myTeam.py and do not modify the interface. Only **myTeam.py** is to be part of the submission.
- You can invoke the Pacman game as follows. A simple example of a baselineTeam agent is provided in the package and is selected as a default for both the red and the blue teams.

```
python capture.py
```



Screenshot of agents as ghosts in their own territory and as pacmans in the enemy's

- There are four agents. The agents 0 and 2 are always on the red team and 1 and 3 are on the blue team. The agents for the red team and the blue team can be selected as follows. The **-r** flag selects the agent for the red team and the **-b** flag selects the agents for the blue team respectively. Here, both teams are created from **baselineTeam.py**.

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

- To control one of the four agents with the keyboard, pass the appropriate **--keys** option. The arrow keys control your character, which will change from ghost to Pacman when crossing the centerline.

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

- By default, all games are run on the **defaultcapture** layout. The layout can be specified with the **-l** option. Random layouts can be generated by specifying **RANDOM[seed]**. For example, **-l RANDOM13** will use a map randomly generated with seed 13.
- You may record the games on your local machine by specifying the **--record** option. This will write the game history to a file named by the timestamp when the game was played. The recorded game history can be replayed using the **--replay** option and specifying the file to replay.
- Adding the **--help** flag will display options available.

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

3. Rules

- **Layout:** The Pacman map is divided into two halves: *blue* (right) and *red* (left). The Red agents (which all have even indices) must defend the red food while trying to eat the blue food. When on the red side, the red agent is a ghost. When crossing into the blue territory, the agent becomes a Pacman. The layouts directory contains several map layouts.
- **Scoring:** As the 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 he is carrying, earning one point per food pellet delivered. The scores for the Red team are positive, while the scores for the Blue team are negative. If the 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, the agents on the opposing team become *scared* for the next 40 moves, or until they are eaten and re-spawn, 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. Please see the GameState section for more details.
- **Winning:** A team wins when they return all but two of the opponents' dots. Games are *limited* to 1200 agent 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:** There will be an initial start-up allowance of 15 seconds (use the **registerInitialState** function. 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 (i.e. the other team will win).

4. Designing Agents

- **Baseline Team:** As a default, a team of two simple baseline agents, defined in **baselineTeam.py** is provided in the package. The **OffensiveReflexAgent** simply moves toward the closest food on the opposite side. The **DefensiveReflexAgent** wanders around on its own side and tries to chase down entrants from the opposite team.
- **File Format:** Your agent must be completely contained in **myTeam.py**. Please adhere to the provided API. Do not change the API interface. This is necessary for evaluation.
- **Interface:** Please review **GameState** in **capture.py**. Note the methods such as **getRedFood**, which gets a grid of food on the red side. The grid is the size of the board but is only true for cells on the red side of the food. The team's indices can be listed with the **getRedTeamIndices** function. The team membership can be tested with **isOnRedTeam**. The code in **distanceCalculator.py** can be used to compute shortest path maze distances.
- **CaptureAgent Methods:** To start the process of designing your own agent, please subclass the **CaptureAgent** class. This class provides several useful functions. Some of the methods are listed below.

def getFood(self, gameState):	Returns the food you are 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 in the form of a number that is the difference between your score and the opponent's 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 to the 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)]**)

- **No Multithreading:** Multi-threading is not allowed. Agents which compute during the opponent's turn will not be considered.

5. Leaderboard on Gradescope

- In this assignment, an option has been provided for evaluating your agent against a couple of TA-agents. This will give you a sense of how your agent is performing with respect to some standard strategies. These agents are hosted on Gradescope. See the section on 'Gradescope' under 'Submission Instructions' for details on how to submit.
- Once you submit, it will play your agent against multiple TA-agents and compute an average score. These scores will be used to create a leaderboard which will be publicly visible. The purpose of leaderboard is to give you a sense of how agents submitted by other students are performing against the TA-agents.
- Please note that your score on this leaderboard is only for your own assessment and will NOT be used in grading (which is described below). Note that submissions on Moodle will be used for final evaluation. Please see below for more details.

6. Grading

The evaluation for this assignment will be done out of **100 points** in the following ways.

a. You will be awarded **0 points** if your submission semantically matches with implementation of the **baselineTeam** provided in the assignment packet: the comparison will be done by the plagiarism detection software (MOSS).

b. **Winning rate against TA-Agents [20 pts]**: Your agent will play N matches against 4 TA-Agents and an **average score** will be computed along with your winning rate.
   i. 5 points for over 51% winning rate against "TA-Agent 1".
   ii. 5 points for over 51% winning rate against "TA-Agent 2".
   iii. 5 points for over 51% winning rate against "TA-Agent 3".
   iv. 5 points for over 51% winning rate against "TA-Agent 4". Note that TA Agent 4 is the same as the provided baseline agent in **baselineTeam.py**.

c. **Relative performance on the final Leaderboard [20 pts]:** Average score computed in part b, will be used to create a **final** leader-board. Note that it will be different from the leader-board on the gradescope. Students will receive a relative credit based on the "**average score**" metric. A minimum of 5 points will be awarded if your average score is positive.

d. **Tournament [60 points]:** We will play 3 tournaments on different layouts. Each tournament is worth **20 points**. The winner from a tournament will not participate in the subsequent tournaments and will be awarded full 60 points. **The tournament will be conducted as follows:**
   i. A tournament will have multiple rounds. The number of rounds will depend on the total number of submissions. For N submissions, the number of rounds would be approximately log(N).
   ii. In each round, We will create groups of 4 or 5 submissions. The group formation will be based on the scores obtained in the previous round. For the 1st round, scores from the final leaderboard will be used to create groups. From each group, only 2 teams will qualify for the next round. Grading for each submission would depend on the number of rounds it plays in a tournament.
   iii. **Grading of a tournament (each 20 points):**

Each tournament will be graded using the following rubric. Please note that there could be minor changes to this rubric which may evolve over time, but the essence would remain the same.

- Winning any tournament: 60 points total for all 3 tournaments combined.
- Losing in finals: 20 points
- Losing in semi-finals: 18
- Losing in quarter-finals: 16
- Losing in pre-quarter finals: 14
- Losing in pre-pre-quarter finals: 12
- Teams that do not qualify to the pre-pre-quarters (and have a functional submission) will receive between 5 and 10 points, based on the number of wins and their score margin. However, if a team timed out, or sent a wrong move, etc., they will NOT get their functionality credit of 5 points.

7. Submission Instructions

This assignment is to be done individually or in pairs. The choice of continuing with a partner from A2, working with a new partner for A3 or working by yourself is entirely your choice and does not affect the grading.

Gradescope:

- Submit a single file named ***myTeam.py*** over gradescope. Autograder will return the results after evaluation after some time.
- You may see a max score of 100 on gradescope, which is a default score. Please ignore it.
- To avoid the load on the gradescope servers, we request you to restrict the number of submissions per team per day and use the evaluation facility on gradescope judiciously. Most likely, this may become an issue as the deadline approaches. As a result, we may start monitoring the number of submissions per team per day and we may have to put an upper limit to it. We will communicate it through Piazza if it becomes an issue. Teams violating the set upper limit may receive some penalty.

Moodle:

- Submit a single zip file named ***<A3-EntryNumber1-EntryNumber2>.zip or <A3-EntryNumber>.zip***. Upon unzipping this should yield a single file named - ***myTeam.py***.
- **For final grading, the assignment is to be submitted on Moodle.**
- **The submission deadline is 5pm on Sunday December 27, 2020.**
- **No deadline extensions are possible as the grading process is to start immediately.**
- This assignment will carry 16% of the grade.
- This is the final assignment for the course.
- There are no buffer days. Please submit by the submission date.
- Your code will be graded using evaluation scripts. Please do not change the names of any of the provided functions or classes within the code. Carefully follow the format. Only make changes to the specified file (myTeam.py) and in the correct functions. Your code should use only standard python libraries. Do not include any dependencies on third party libraries.

- No credit provided if you modify other functions which you are not supposed to.
- Please only submit work from your own efforts. Do not look at or refer to code written by anyone else. You may discuss the problem, however the code implementation must be original. Discussion will not be grounds to justify software plagiarism. Please do not copy existing assignment solutions from the internet: your submission will be compared against them using plagiarism detection software.
- Copying and cheating will result in a penalty of at least -10 (absolute). The department and institute guidelines will apply. More severe penalties such as F grade will follow.
- Queries if any should be raised on Piazza. Enrol on **piazza.com/iit_delhi** for the course **Fall 2020** term of **COL 333: Artificial Intelligence** using access code **col333.**
- Remember that the autograder given to you tests your solution on default test cases. The final evaluation may be done on new unseen test cases. Therefore, you may still not receive credit even if the autograder passes on the default test cases.
- Requests for **late submission** on the grounds of **medical emergencies** must be accompanied with a medical certificate from a qualified doctor indicating that you were unwell in the period of submission and a proof of prescription. Additional proof may be requested to verify the submitted documents. Any **other requests** on other grounds such as law and order, basic infrastructure must be accompanied with written proof. Formal proof and verification is necessary for consideration of any such requests. Any decision will be taken on those as per institute guidelines and consideration of the documents. The documents may be submitted in the Department as per institute guidelines.

8. Undertaking

The Pacman project was developed by Dan Klein at the University of California, Berkeley and is used for AI education in several universities. In order to support use of the Pacman framework for teaching in several universities, each student enrolled in COL333 and COL671 must take the following undertaking: “The Pacman project is freely available for educational use. Since the framework is used at multiple universities for AI education, it is mandated that we do not distribute or post solutions to any of the projects. Any redistribution of the code or release (e.g., on a Github account) by any student taking the course will be considered as a violation of the Honor Code in the class and will lead to penalties”. A penalty will be applied if a student makes the solution available online.

9. List of Files in the Pacman Project

Files to edit:	
myTeam.py	What will be submitted to Gradescope. Contains all of the code needed for your agent.
Files to look at:	
capture.py	The main file that runs games locally. This file also describes the 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.
Files to ignore:	
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

10. List of command line options available in the Pacman Project

Also available through python capture.py --help

Command line	Description
python capture.py	starts a game with two baseline agents
python capture.py --keys0	starts a two-player interactive game where the arrow keys control agent 0, and all other agents are baseline agents
python capture.py -r baselineTeam -b myTeam	starts a fully automated game where the red team is a baseline team and blue team is myTeam

Option	Description
-h, --help	show help message and exits
-r RED, --red=RED	Red team [Default: baselineTeam]
-b BLUE, --blue=BLUE	Blue team [Default: baselineTeam]
--red-name=RED_NAME	Red team name [Default: Red]
--blue-name=BLUE_NAME	Blue team name [Default: Blue]
--redOpts=REDOPTS	Options for red team (e.g. first=keys)[Default:]
--blueOpts=BLUEOPTS	Options for blue team (e.g. first=keys) [Default:]
--keys0	Make agent 0 (first red player) a keyboard agent
--keys1	Make agent 1 (second red player) a keyboard agent
--keys2	Make agent 2 (first blue player) a keyboard agent
--keys3	Make agent 3 (second blue player) a keyboard agent
-l LAYOUT_FILE, --layout=LAYOUT_FILE	the LAYOUT_FILE from which to load the map layout; use RANDOM for a random maze; use RANDOM<seed> to use a specified random seed, e.g., RANDOM23 [Default: defaultCapture]
-t, --textgraphics	Display output as text only
-q, --quiet	Display minimal output and no graphics
-Q, --super-quiet	Same as -q but agent output is also suppressed
-z ZOOM, --zoom=ZOOM	Zoom in the graphics [Default: 1]
-i TIME, --time=TIME	TIME limit of a game in moves [Default: 1200]
-n NUMGAMES, --numGames=NUMGAMES	Number of games to play [Default: 1]
-f, --fixRandomSeed	Fixes the random seed to always play the same game
--record	Writes game histories to a file (named by the time they were played)

--replay=REPLAY	Replays a recorded game file.
-x NUMTRAINING, --numTraining=NUMTRAINING	How many episodes are training (suppresses output) [Default: 0]
-c, --catchExceptions	Catch exceptions and enforce time limits