Project 2, 2017

Aim

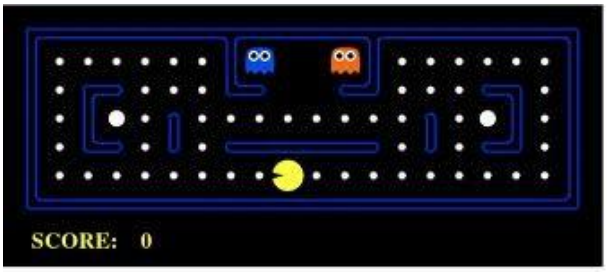
The purpose of this project is to implement a *Pac Man* Autonomous Agent that can play and compete in a *tournament*. To help you develop your solution you must provide:

(i) A PDDL formulation of Pac-Man as classical planning problem (5 marks).

(ii) A working Pac Man Agent that is capable of playing Pac-Man and competing in the tournament. Your Agent can use any technique, or combination of techniques, that you choose. For example using a classical off-the-shelf *FF* planner, Reinforcement Learning, Heuristic Search, Monte Carlo Tree Search or a purpose built decision tree of *your own making* (25 marks).

(iii) A 5-minute oral presentation in your workshop that outlines the theoretical or experimental basis for the design of their agent (that is, why you did what you did), challenges faced, and what you would do differently if you had more time (5 marks).

(iv) A 2 pages report: 1st page describing the approaches implemented, 2nd page experimenting the performance of your agents in several scenarios. (5 marks).



The intention of the project is to follow on from your project 1 submission concerning Berkeley Pac Man as a basis and starting point for project 2, based on http://ai.berkeley.edu/ contest.html.

Your task

1. Axiomatisation of Pac Man in PDDL

Typical applications of planning consist on one or several calls to a planner. The instances are generated *on the fly* by a *front–end* (the pacman engine), and the solutions (plans) are interpreted as executable instructions. As the pacman is not a classical single agent problem, you are required to implement two points of view: The point of view of the pacman, where its goal is to stay alive while eating all the dots of the grid, and the point of view of the ghost, whose goal is to kill pacman.

Assume that the game is turn-based, so at each step an instance is generated with the current state of the world, i.e. the dots and ghosts locations in the grid. From the point of view of pacman, the ghosts don't move, and vice-versa, that is, the environment is static.

At each step the planner would come out with a plan to eat all the dots while avoiding static ghosts, and plans to enable ghosts to kill the static pacman. A simple interpretation of the plans by the pacman engine is to execute only the first action of the plan, ignore the remaining actions, and call the planner in the next step with a new updated instance accounting for the new locations of the ghosts and the pacman.

The axiomatisation should define the state model for pacman using PDDL, and another PDDL for a ghost state model. Explain clearly the assumptions made, e.g. pacman do not move to cell X, when Y holds, etc., and describe several initial states or goals to illustrate interesting situations.

Make sure that your PDDL files can be solved using the online solver in http://editor.planning.domains.

2. Implementation of Pac Man

Produce a working agent that can play Pacman within the Pacman game engine. You *have to use at least 2 of the techniques* that have been discussed in the subjects, and you can combine them in any form. The candidate techniques are:

1. Heuristic Search Algorithms (using general or pacman specific heuristic functions)
2. Classical Planning (PDDL and calling a classical planner)
3. Value Iteration (Model-Based MDP)
4. Monte Carlo Tree Search or UCT (Model-Free MDP)
5. Reinforcement Learning – classical, approximate or deep Q-learning (Model-Free MDP)
6. Goal Recognition techniques (to infer intentions of opponents)
7. Game Theoretic Methods

You can always use hand made decision trees to express behaviour specific to Pac-Man, but they won't count as a required technique. You are allowed to express domain knowledge, but remember that we are interested in Autonomy, and hence using techniques that generalise well. The first 7 techniques can cope with different rules much easier than any decision tree (if-else rules).

While any submission using techniques from the lectures will suffice, perhaps the simplest and most basic way of doing this is to use your models from task 1. Use one PDDL domain file for pacman, and one domain file for the ghost containing the predicates and the actions of the world.

The problem file describes the 'initial' state and goals. Therefore, with a single domain for either the pacman or the ghost, several problems can be generated by only updating the problem file.

By reading the state of the Pacman get from the engine and converting this into PDDL predicates, you can describe the state of the game in PDDL and, at each step that an action is required, call your favourite planner using that state as the initial state. Then, parse the solution in order to choose the best action.

Different domains can be used to encode different strategies. Explain each domain encoded and the differences among them, if needed. Note that the pacman tournament has different rules as it's a two teams game, where your Pac-mans become ghosts in certain areas of the grid. Please read carefully the rules of the pacman tournament.

If you decide to compute a policy, you can save it into a file and load it at the beginning of the game, as you have 15 seconds before every game to perform any pre-computation.

While a classical planning approach is perhaps the simplest way to get a working agent, it is unlikely to do well in the tournament play if not combined with other techniques. That is, you should think about each possible situation that may arise during the game, and use the best technique you know. You do not need to use classical planning for each situation, actually you don't need to use it at all. Just use at least 2 different techniques from the list in section 2.

Deliverables

Exactly one member of your team must submit:

(i) a PDDL formulation of Pac Man, using *PDDL* (*.pddl*) file format,

(ii) a working Pac Man *Agent* that is capable of competing in the tournament in Python (*myTeam.py*),

(iii) You must include a *readme.txt* file which briefly documents the code and outlines the approaches you've used,

(iv) a *group.txt* plain text file listing your group members' logins, one per line

(v) a *report.pdf* of 2 pages, explaining your agents and doing experiments to show how different variations of your agent perform against each other.

In your report.pdf, you must describe the techniques that you have implemented in your Agent for the tournament. Please clearly state any assumptions that you make. Include an analysis of the strengths and weaknesses of your techniques.

The group.txt file must (strictly) include a list of lowercase login names, one per line for all people in your Project group (or just one login for individuals). For example,

```
>cat group.txt
```

>login1
>login2

Individual report [optional]: For those working in teams, each team member is requested to submit a 0.5 page individual report (as a **text file (individual report.txt)**) outlining two things:

(i) What they contributed to the project.

(ii) What they perceive their team members contributed to the project.

This individual report is intended to give everyone the opportunity to inform subject staff of the contributions of their team.

While the project can be done as a team, we reserve the right to assess members of a team individually.

While this report is optional, you will not be given a chance to submit one after the deadline if your mark is affected by one of your team members report.

Keep your report to 0.5 page. Anything longer than 0.5 page will not be read.

Marking criteria

Part (i) – 5 marks

- PDDL model successfully generates good plans suitable to be used for Pacman [2 marks]
- PDDL model is sound (correctly models a deterministic Pacman game) and complete (models all required behaviour: ghost and pacman) [2 marks]
- PDDL model uses the appropriate levels of abstraction [1 mark]
  - Encode the Grid
  - Point of View of Ghost (static Pacman, goal eat pacmans)
  - Point of View of Pacman (static Ghosts, goal eat food and come back to home area)
  - Power Level (just check whether is SuperPacman or not)

Part (ii) – 25 marks

Marks will be given according to final position in the tournament with respect to *staffTeam*:

- Pacman competitor finishes above the *staffTeamBasic* agent [11 marks].
- Pacman competitor finishes above the *staffTeamMedium* agent [8 marks].

- Pacman competitor finishes above the *staffTeamHard* agent [6 marks].
- Final competition place (up to 3 bonus marks).

  1st place in the competition will receive 3 bonus marks, 2nd place will receive 2 bonus marks, and 3rd place will receive 1 bonus mark.

Part (iii) – 5 marks

- A clear presentation of the design decisions made, challenges experienced, and possible improvements [2 marks]
- A clear demonstration and understanding of the subject material [2 marks]
- Ability to answer questions about your design in a clear and convincing manner [1 marks]

Part (iv) – 5 marks

- A clear written description of the design decisions made, approaches taken, challenges experienced, and possible improvements [3 marks]
- An experimental section that justifies and explains the performance of the approaches implemented [2 marks]

1. You may assume that ./ff is version 2.1 of the Metric-FF planner (https://fai.cs. uni-saarland.de/hoffmann/metric-ff.html). If you want to use any other planner or 3rd-party executable please discuss with Tim and Nir before submission.

When submitting a solution, please make absolutely sure you adhere to the following instructions:

1. Your code **must run** on Linux and adhere to **Python 2.7**. Staff will not debug or fix any code.
2. At the very minimum, your code should be **error-free**. If your code crashes in any execution, it will disqualified from the contest. Again, staff will not debug or fix code that crashes.
3. Your code will be copied into a directory called teams/<your_teamname>/ in the contest package. This means that if you import from other files outside myTeam.py they will not be found unless you tell Python to look in your team dir. You can do so by having the following code on top of your myTeam.py:

import sys
sys.path.append('teams/<your _team>/')

4. Your code will be run by the following command:

   python2.7 capture.py -r teams/<team1>/myTeam.py -b teams/<team2>/myTeam.py,

   please make sure your AgentFactory is defined in myTeam.py.

5. You are to **submit just one zip file** with extension .zip (any name is fine). Such zip file should contain all the files needed to run your agent in the *root* directory of the file.

6. Do *NOT* use the current working directory to write temporary files; instead, redirect all output to your own folder ./teams/<your_teamname>/. For example, if you use a planner online, and generate PDDL files and solutions, redirect your planner call, solution outputs, etc., to your own folder. You can use python code to do it automatically, or you can hardcode it assuming that your team will be located in ./teams/<your_teamname>/ folder.

7. If you want to use any other 3rd-party executable please discuss with us before submission.

8. Finally, submit your project **substantially before** the deadline, preferably one day before. Submitting close to the deadline could be risky and you may fail to submit on time, for example due to lose of Internet or server delays. There will be no extensions based on these unforeseen problems.