Universitat Pompeu Fabra Barcelona

24304: Artificial Intelligence 2024

Final Lab Project: Pacman Contest

Deadline: To be determined

This project counts towards 15% of the grade for this course.

This assignment must be done in groups of 2 students.

---

1 Introduction

The Eutopia Pacman contest is an activity consisting of a multiplayer capture-the-flag variant of Pacman, where agents control both Pacman and ghosts in coordinated team-based strategies. Students from different EUTOPIA universities compete with each other through their programmed agents. Currently both University of Ljubljana, Vrije Universiteit Brussel, and Universitat Pompeu Fabra (UPF) are participating organizations. UPF is also the tournament organizer, which hosts and run the tournaments in the SNOW cluster[1].

The project is based on the material from the CS188 course *Introduction to Artificial Intelligence* at Berkeley[2], which was extended for the AI course in 2017 by lecturer Prof. Sebastian Sardina at the Royal Melbourne Institute of Technology (RMIT University) and Dr. Nir Lipovetzky at University of Melbourne (UoM)[3]. UPF has refactored the RMIT and UoM code. All the source code is written in `Python`.

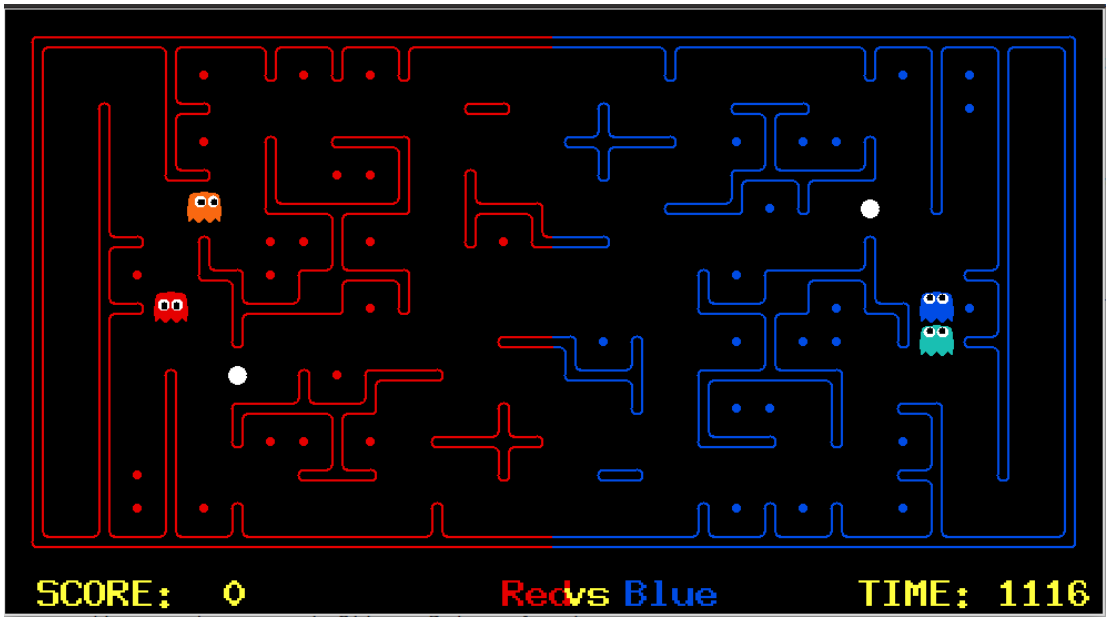

Figure 1: Berkeley's Pac-Man environment in action.

The project code is developed in a modular way, so that users can work at different levels depending on their objective. There are three different modules:

[1] https://guiesbibtic.upf.edu/recerca/hpc

[2] http://ai.berkeley.edu/contest.html.

[3] https://github.com/AI4EDUC/pacman-contest-cluster.

Agent development : the source code of a participating agent is contained in a `github` repository. Each participating team will build a single repository. The Pacman Agent defines a basic template of an agent behavior.

Local tournament : The (Pacman Contest) module contains the scripts needed to run a custom tournament locally, independently of the tournaments organized by UPF.

UPF tournament : The Pacman Eutopia is the module used by tournament organizers at UPF. Participating organizations do not need to contribute to this module.

Currently, the framework supports a running mode based on *events*. This means that tournaments are run at UPF according to prespecified dates between the tournament participants. A different mode with results continuously being updated is left for future versions of the framework. A test run will take place, and based on the results of the test run, students can make changes before the final run around one week after.

2 Rules of Pacman Capture the Flag

2.1 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.

2.2 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 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.

2.3 Eating Pacman

When a Pacman is eaten by an opposing ghost, the Pacman returns to its starting position (as a ghost). No points are awarded for eating an opponent.

2.4 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).

2.5 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.

2.6 Winning

A game ends when one team returns all but two of the opponents' dots. Games are also limited to 1200 agent moves (300 moves per each of the four agents). 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.

2.7 Computation Time

We will run your submissions on the UPF cluster, SNOW. Tournaments will generate many processes that have to be executed without overloading the system. Therefore, 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 `register_initial_state` function). If your agent times out or otherwise throws an exception, an error message will be present in the log files, which you can download from the results page.

3 For students

Students need to first download the source code and install the required dependencies.[4]

3.1 Setting up the agent and contest frameworks

Step by step:

1. Clone the repository to download all the necessary code.
   `git clone git@github.com:aig-upf/pacman-agent.git`
2. Move to the created directory.
   `cd pacman-agent/`
3. Create a virtual environment.
   `python3.8 -m venv venv`
4. Activate the virtual enviroment.
   `source venv/bin/activate`
5. Pull the contest framework.
   `git submodule update --init --remote`
6. Install the contest framework and required python libraries.
   ```
   cd pacman-contest/
   pip install -e .
   pip install -r requirements.txt
   ```
7. Finally, move to the directory containing the main file (`capture.py`) to run a match.
   `cd src/contest/`

[4] Commands expected to be used in an UBUNTU operative system and tested for version UBUNTU 22.04.

3.2 Getting Started

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

```
python capture.py
```

A wealth of options are available to you:

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

The code provides one sample team called `baseline_team`, contained in a python script named `baseline_team.py` in `src/contest` folder. 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 baseline_team -b baseline_team
```

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

Once this last step is working, we can start running games between our custom agents, or between a custom agent and the `baseline_team`. To do this, we will save our agent's directory in the `src/contest/agents/` folder. Inside this folder we will have our directory, which can have any name. As an example, let's imagine that we have two agents, `team_name_1` and `team_name_2`. Each of these folders contains an agent, in a script called `my_team.py`. An executable example is provided in the framework:

```
python capture.py -r agents/team_name_1/my_team.py -b agents/team_name_2/my_team.py
```

We could also compare our agent against the `baseline_team`, by running

```
python capture.py -r agents/team_name_1/my_team.py -b baseline_team
```

There is also an option to record the game and some log info by adding the flags `--record` and `--record-log` to the previous command.

```
python capture.py -r agents/team_name_1/my_team.py -b baseline_team --record --record-log
```

The previous command will save the data in the following files:

- Log: `www/contest_default/logs/match_0.log`
- Replay: `www/contest_default/replays/match_0.replay`
- Score: `www/contest_default/scores/match_0.json`

Finally, a match can be replayed from a `*.replay` file. Using the one generated in the previous execution, we can run it as follows:
`python capture.py --replay=www/contest_default/replays/match_0.replay`

3.3 Building your agent

In the root folder do the following:

1. Create in `my_team.py` a class with the name of your agent that inherits from `CaptureAgent`, e.g. class `ReflexCaptureAgent(CaptureAgent):`
2. In the new class, override the `def choose_action(self, game_state):` function to return the best next action (check the given source code example).
3. (Optional) Add any other functions to the class for reasoning / learning and improving your agents decision which could also use other code sources in the same folder.

If you want to debug your agent, provide the local route to `capture.py`, e.g., `python capture.py -r baseline_team -b ../../../my_team.py` for your agent to play against the `baseline_team`.