

Eutopia Pacman Contest documentation

Version 1.1

November 20, 2024



Contents

1	Introduction	2
2	Rules of Pacman Capture the Flag	3
2.1	Layout	3
2.2	Scoring	3
2.3	Eating Pacman	3
2.4	Power Capsules	3
2.5	Observations	3
2.6	Winning	3
2.7	Computation Time	3
3	For students	4
3.1	Setting up the agent and contest frameworks	4
3.2	Getting Started	4
3.3	Building your agent	5
4	For participating organizations	6
4.1	Running a contest	7
4.2	Visualization of the results	8

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 and Universitat Pompeu Fabra (UPF) are participating organizations. UPF is also the tournament organizer, which hosts and runs the tournaments in the HDTIC cluster.

The project is based on the material from the CS188 course *Introduction to Artificial Intelligence* at University of California, Berkeley, 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). UPF has refactored the RMIT and UoM code. All the source code is written in Python language.

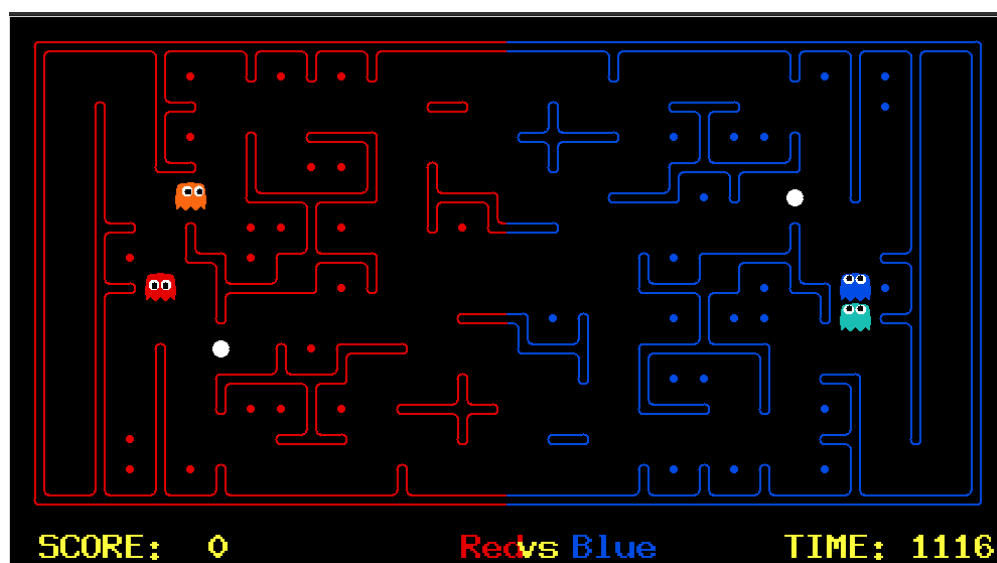


Figure 1: Berkeley's Pac-Man environment in action.

The expected prerequisites for a participating student include programming skills, knowledge of data structures and algorithms, search algorithms, and probability, all of them at the level of an introductory course in Artificial Intelligence.

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:

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 tournaments that are run at UPF according to prespecified dates between all the tournament participants. A different mode with results continuously being updated is left for future versions of the framework.

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). Additionally, an agent 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 `registerInitialState` 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.¹

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 environment.
`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/`

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`.

¹Commands expected to be used in an UBUNTU operative system and tested for version UBUNTU 22.04.

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`.

4 For participating organizations

Participating organizations need to make available the following information to the UPF. Figure 4 shows a diagram illustrating the organization of a tournament.

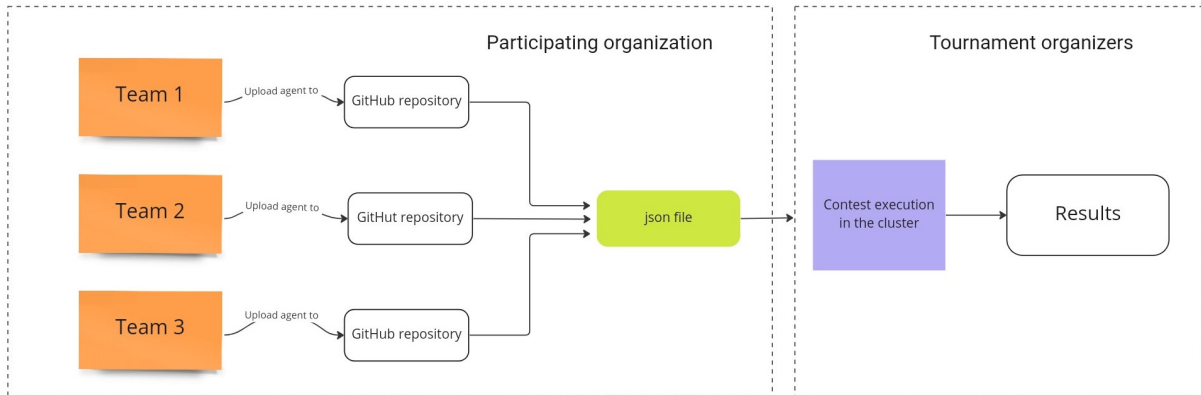


Figure 2: Diagram of a tournament: The teams of a participating entity upload their agents to a GitHub repository. When all agents are ready, the entity generates a `.json` file with the required information. The `.json` file is shared to the organizers of the tournament, who will run the games and return the results in the form of a website.

The participating organization prepares a `.json` file that is copied in the DTIC cluster at UPF by the tournament organizers and is used to automatically run a contest and make public the results. The contents of the file are:

1. Individual identification number for each team.
2. Team members information (member id and name).
3. Team name.
4. Repository GitHub URL.
5. Last commit code from GitHub repository.
6. Configuration flags: `loading_error`, `syntax_error` and `updated` defined as `false` by default.

This setup favours scalability, as adding a new team member only implies adding a new entry in the `.json` file.

The following example contains two teams (team 1 and team 2), each of them composed of three students. Team 1 is composed of Anna, Bob, and Claire and Team 1 is composed of Charlie, Mary, and Jane. There is one repository URL for each team containing directories with the names of the students.

```
{
  "teams": [
    {
      "id": 1,
```

```

    "last_commit": "3cf7c4575f34acc1887aba1b2061c10ce1289747",
    "loading_error": false,
    "members": [
      {
        "id": "123",
        "name": "Anna"
      },
      {
        "id": "125",
        "name": "Bob"
      },
      {
        "id": "139",
        "name": "Claire"
      }
    ],
    "name": "firstTeamName2022",
    "repository": "https://github.com/user/pacman-agent.git",
    "syntax_error": false,
    "updated": false
  },
  {
    "id": 2,
    "last_commit": "3cf7c4575f34acc1887aba1b2061c10ce1289747",
    "loading_error": false,
    "members": [
      {
        "id": "335",
        "name": "Charlie"
      },
      {
        "id": "742",
        "name": "Mary"
      },
      {
        "id": "201",
        "name": "Jane"
      }
    ],
    "name": "secondTeamName2022",
    "repository": "https://github.com/user/pacman-agent.git",
    "syntax_error": false,
    "updated": false
  }
}

```

4.1 Running a contest

Running a contest is currently done by UPF only. At a pre-specified date, UPF will run the tournament with the currently available `.json` file from all the participating organizations.

4.2 Visualization of the results

Once a tournament has been executed, an open web site is made available with information about the results. There are three types of results shown: ranking, disqualified teams, and matches. Figures 3,4, and 5 show an example of each result.

Position	Team	Points %	Points	Win	Tie	Lost	TOTAL	FAILED	Score Balance
1	thirteenthTeamName2022	51.0%	20	4	8	1	13	0	55
2	sixthTeamName2022	48.0%	19	4	7	2	13	0	55
3	thirdTeamName2022	48.0%	19	3	10	0	13	0	37
4	tenthTeamName2022	48.0%	19	3	10	0	13	0	37
5	secondTeamName2022	43.0%	17	2	11	0	13	0	19
6	eighthTeamName2022	41.0%	16	3	7	3	13	0	37
7	fifthTeamName2022	41.0%	16	2	10	1	13	0	19
8	firstTeamName2022	41.0%	16	2	10	1	13	0	19
9	twelfthTeamName2022	38.0%	15	2	9	2	13	0	19
10	eleventhTeamName2022	38.0%	15	2	9	2	13	0	19

Figure 3: Ranking example showing the teams ordered by score

Team	Log file
upf-ai22_teamPytorch2022.log	UPF-AI22_TEAMPYTORCH2022.LOG
upf-ai22_fourthTeamName2022.log	UPF-AI22_FOURTHTEAMNAME2022.LOG

Figure 4: Disqualified teams. A .log file is available with information.

Team 1	Team 2	Layout	Time	Score	Winner	Score file	Replay file	Log file
fifthTeamName2022	twelfthTeamName2022	defaultCapture.lay	0:00:40	0	None	MATCH_494.JSON	MATCH_494.REPLAY	MATCH_494.LOG
firstTeamName2022	thirdTeamName2022	defaultCapture.lay	0:00:39	0	None	MATCH_443.JSON	MATCH_443.REPLAY	MATCH_443.LOG
secondTeamName2022	eleventhTeamName2022	defaultCapture.lay	0:00:40	0	None	MATCH_463.JSON	MATCH_463.REPLAY	MATCH_463.LOG
eighthTeamName2022	secondTeamName2022	defaultCapture.lay	0:00:37	18	secondTeamName2022	MATCH_460.JSON	MATCH_460.REPLAY	MATCH_460.LOG

Figure 5: Matches. One row is generated for each match with a score, a file to replay the match visually, and a .log file with log information.