CS181 Final Project
Spring 2011

Writeup due Wednesday, May 4th, 5pm
Tournament on Monday, May 2nd, 1 - 3pm in MD G125.
Agents due 11:59pm Sunday, May 1st

1 Introduction

In this project, you will design an agent for playing a simple game. The game has been designed so that all the techniques learned in the course are applicable to it, but no one technique alone will provide a complete solution. Also, there is no such thing as a perfect solution to this game. The game is challenging, and it will require creativity to come up with a program that works well.

You should try to work in groups of two. If you prefer to work by yourself that is also OK. Please let us know who your group is by Monday Apr 26th (we'll put up a google form for this). If you have trouble getting into a group, send us email and we will be happy to help you.

The writeup is due at 5:00pm on Wednesday, May 4th. There will be a tournament on Monday, May 2nd, 1-3PM in MD G125. We will need your agents submitted the night before, by 11:59pm on Sunday, May 1st. All agents, from CS181 and CSCI-E181 will compete in the same tournament.

2 The Game

The game involves a herbivorous robot Xavier who lives in a two-dimensional world. The world is an infinite grid, and the robot begins at the center of the grid. The robot may move right, left, up or down. It cannot stay in the same place, though it may return to a place they have been before. Each location in the world may contain a plant. When a robot moves to a location containing a plant, it may choose to eat it.

The robot's moves are stochastic. At each turn, when it tries to move, a robot chooses one of the four directions. The location in which the robot ends up is determined stochastically based on that. With some probability it goes forward in the chosen direction, and with some probability it goes to either side.

The robot needs energy in order to live. It starts out with a certain amount of energy, and uses up one unit of energy each move. Some of the plants are nutritious, others are poisonous. The robot gains energy from eating a nutritious plant, and loses energy from eating a poisonous plant. A robot dies when its energy is less than or equal to 0.

When a robot sees a plant, it may ask for an observation of the plant. An observation is a 6 by 6 pixel image, with each pixel being 0 or 1. Getting an observation costs one unit of energy. Asking for an observeration is optional, and the robot may ask for more than one observation if it likes. Observations are noisy.

The robot knows only what it has experienced. That is, the full state of the environment is unobservable. Not only do you not see the true type of a plant in your current location, but you don't know whether a plant exists in another location until you visit that location. When the robot is in a location, it knows whether or not there is a plant at the location. When the robot eats a plant, it finds out whether or not it was nutritious or poisonous.

Formally, this is a Partially-observable MDP environment and also includes the need for learning: you don't know the transition model for your robot and you don't know the probabilistic model for how plants are distributed in the world, or for the way that noisy images are generated from poisonous and nutritious plants.

The initial energy, bonus for eating a nutritious plant, and penalty for eating a poisonous plant, are parameters to the game.

The default values are 100 for initial energy, 20 for the bonus and 10 for the penalty. These may be changed a bit for the final tournament for time management reasons.

On the day of the tournament, pairs of robots will compete on the same instance of the world. That is, if the robots make the same observation and movement decisions, the robots will be given the same score. The robot that survives for a the longest number of rounds (before running out of life) will be declared the winner. The tournament will face off winning pairs in subsequent rounds until there is a single robot remaining.

3 Running the program

The support code for the final project can be found on the website. The game is implemented in a compiled Python module. We have compiled the module for different platforms: `mac, nice, linux32, linux64`. To use a particular platform, cd into the `project` directory and run:

```
cp {platform}/* .
```

We did not compile Windows binaries, but please email `cs181@fas.harvard.edu` if this is a problem for you.

To run a game, use `run_game.py`. `run_game.py --help` gives you the different options for running the game. To code your custom behavior for determining whether to view images and where to move in the game, you will implement your own `get_move` function in `player1/player.py` and `player2/player.py`. `get_move` takes in a `PlayerView` object whose definition can be found in `python_game.h`.

This is a `C++` header file, but we use SWIG to generate Python code from this, so you can call the methods defined in `python_game.h` on the view passed to `get_move`. Example usage can be found in `common.py` and `test_game_interface.py`.

To ensure that the tournament runs reasonably quickly, we are limiting the amount of time taken to 1 second per turn during the course of the game.

You will submit a directory that contains a `player.py` function that implements the `get_move` method. You may use any other functionality you like (e.g. writing code in C and wrapping it with Python), but all of this functionality must be included in your directory and run on `nice.fas.harvard.edu`. We should be able to drop your directory into a `player1` or `player2` directory and run `run_game.py`.

4 What you need to do

There are three basic decisions that need to be made: whether to request an observation of a plant, whether or not to eat a plant at the current location, and where to go next. You will need to implement an agent that makes all of these decisions.

The design of the controller is completely up to you. All of the methods we have studied could reasonably be applied to this project. **A minimum requirement for the project is that your program should incorporate at least two methods that we have studied**, and any others you think are appropriate. The two methods should apply to different aspects of the problem, or approach the same aspect of the problem in fundamentally different ways. Thus, implementing two different classification methods for classifying the plants from in-game data would not be adequate. However, using one method for learning from in-game data, and another for learning between games, would be fine, as would combining a clustering and a classification method in the plant recognition. As us if you have any questions.

This is a minimum requirement. A project that fails to meet this requirement will not receive a good grade. Implementing additional methods can potentially help your grade. There are different ways to earn a good grade. In general, a good project will use a creative, well-thought out design. It is possible to earn a top grade for the project by implementing just two methods, if they are implemented in a thorough, insightful manner, where the methods are carefully chosen and evaluated. Alternatively, a project that contains many good ideas can also earn a top grade, even when the individual ideas are not as carefully worked out, as long as the ideas are well-motivated.

What consitutes a well-motivated and well-evaluated method? Motivating a method requires thinking about the problem domain and task to be solved. A method should be chosen whose properties are appropriate to the task. Different alternatives should be considered before choosing a method. In implementing a method, you may have a number of design decisions to make. These should be carefully worked out. In addition, your method and design decisions should be evaluated.

A good method could be one that we have studied in the course, or another method that is creatively worked out in response to the problems posed by the domain. (Two of your methods have to come from the course.) Beware of engineering hacks — tricks that happen to improve the performance of the player for no particularly good reason. While you are welcome to implement them to improve performance, they should not be the basis of your project.

Feel free to discuss any ideas you have for the design with the teaching staff.

In addition to your controller, you should also submit a writeup of your project. In the writeup, you should discuss the design, explaining why you chose your methods, how you applied them to the problem, and how you evaluated them. Also discuss whether or not your design decisions actually worked as expected. Were there unexpected difficulties, or surprising benefits?

5 Grading

AI is different from many areas of CS, in that one cannot simply talk about "correct" implementations. Obviously, an agent that makes decisions randomly would be "correct" in that it wouldn't crash the system, and always provide an answer to every question, but it would be a poor solution. One could try to judge an implementation solely on the basis of the performance, but that wouldn't be fair — sometimes a good idea doesn't perform as well as one would expect, for unforseen reasons. We won't punish a good idea, as long as you document it well, and try to understand why it did not do as well as you hoped. On the other hand, since performance is the ultimate arbiter, we will reward agents that perform well.

To summarize, your grade will be based on the following factors:

- The design of your agent. This is the most important component.
- Whether or not you implemented your design correctly.
- Evaluation of your design.
- The quality of the writeup.
- Performance of your agent. This will be a relatively small component. We will not penalize well thought out designs that happened not to work as well as you hoped. However, the teams that do well in the tournament will receive some extra credit.

  The tournament will use the default image model, and the default geographic distribution of nutritious and poisonous plants. Once we get your agents on May 1st, we'll run some experiments to see whether we need to adjust the starting life and the nutritious bonus and poisonous penalty to make sure that we can finish the tournament in the time allotted. You should design your agent so that it is robust to changes in these values. (For example, they can be passed as parameters).

Good luck, and have fun!

6 Submission

When submitting, please create a `submit` directory with the following:

- writeup in `writeup.pdf`,
- your drop-in replacement for `player1` or `player2` in a directory named `player`,
- all other code you wrote in a directory named `code`.

Once this directory structure has been created, execute the following statement (on `nice.fas.harvard.edu`):

```
/usr/local/bin/submit cs181 6 `pwd`/submit
```