Project 2
Computer Science 4013: Artificial Intelligence
Due part 1: 12:01pm March 17, 2009
Due part 2: 12:01AM March 31, 2009
Note: NOT the beginning of class

Introduction

For this project, your task is to use at least two learning techniques (including evolutionary computation) to create an intelligent agent that is able to thrive in the newly defined Spacewar environment. Your project 1 A* agent will likely be useful in this project. You may also use the provided A* code if you do not want to use your project 1 code. Truly intelligent agents will integrate techniques from both learning and search.

For the evolutionary computation aspect of your search, you will need to design an agent that can intelligently choose among a set of high level behaviors. This will require you to design a useful encoding of chromosomes for each individual and design an appropriate fitness function to aid the agent. The chromosome would encode a policy that is a mapping from states (you define these) to high level behaviors. We have provided a list of sample high level behaviors below but you can define your own.

- Move to nearest beacon (AStar code!)
- Move near to another ship's location (presumably to shoot or self-destruct, also uses AStar code)
- Shoot at ship
- Shoot at asteroid
- Lay a mine
- Fire a freezing laser (makes the ship it hits unable to choose actions for 5 time steps)
- Raise shields
- Guide missile (Astar code!)
- Move near to nearest asteroid (presumably to follow it, not to ram it, also uses AStar code)
- Flee from asteroid(s) - you tell it how many asteroids to flee from
- Flee from ship(s) - you tell it how many ships to flee from

You should design a fitness function that rewards the agent for performing well and penalizes it for performing badly. You will need to decide what the criteria for performing well

and badly are but I suggest that you do not make your fitness function overly complicated. For the fitness function, you should focus on your goals for your agent and not on how the agent will accomplish those goals. A good fitness function here tells the agent how successful it is but it does not tell the agent how to accomplish the goal.

Creating your state space is a good spot to integrate your second learning approach. For example, you could use clustering to learn the state space automatically and use evolutionary computation on top of those states. An alternative idea would be to use another learning approach to create some of these higher level behaviors (such as shooting at a ship effectively) and evolutionary computation to choose among them. Many other ideas exist as well!

No matter how you choose to integrate your second learning approach, encoding your chromosome will be critical to the success of evolutionary computation methods. For example, as we discussed in class for tic-tac-toe players, you may create a high-level state space and encode a policy of action responses to each state in your individual. Other encodings are quite possible and may work even better! Even if you choose this approach, you will need to design or learn your state space carefully. We have given you a few ideas in the writeup and in our sample agent but you will need to extend these ideas for your own agent.

The remaining pieces of an evolutionary computation solution are selection, crossover and mutation. You need to pick approaches to each of these using what we discussed in class. The following article on the web may be helpful:

`http://www.tjhsst.edu/~ai/AI2001/GA.HTM`

The following is a list of ideas for state space features. Note that this representation lacks information about your ship's current beacon count or your opponent's health or ... (the list goes on. You can't represent it all in main memory so the problem must become partially observable!).

- distance to nearest 3 asteroids: near, middle, far
- angle to nearest 3 asteroids: 4 values for each
- distance to nearest beacon: near, middle, far
- angle to nearest beacon: 4 values
- health: low, middle, high
- number of ships in game: 1, 2, 3 or more
- distance to nearest ship (other than self): near, middle, far
- angle to nearest ship (other than self): 4 values

Since your second learning method is not specified by the project description, we need information from you quickly. See the rubric discussion below.

Extra-credit opportunities

To keep grades within a fair range, all extra credit will be capped at 10 points. This means that no project can receive a grade higher than 110, no matter if they win the ladder, find great exploits, and are very creative.

Wanted dead or alive: exploits in the simulator

We are reasonably certain that you cannot exploit the simulator (say by directly affecting your opponents health or by directly moving yourself to the goal location). However, any project has bugs and we want to know about them! If you find an exploit to the simulator, you can receive extra credit according to the following scale:

- **5 points:** If you find an exploit and report it to us *using the google code bug reporting system*, you can receive 5 points extra credit upon verification of the report.
- **10 points:** If you find an exploit and give us a fix for it, you can receive 10 points extra credit upon verification of both the exploit and the fix.
- **1 or 2 points:** General simulator bugs are much more likely than exploits. Finding a bug and reporting it using the google code bug reporting system can get you 1 point. Fixing the bug and giving us the fix (you can't check it in directly but you can give it to us in the bug report) can get you two points. Both bug and fix must be verified for any extra credit to be awarded.

Competition ladder

The class-wide competition ladder will start on March 17th (2 weeks before the project is due). For this project, each game will be played with several heuristics, your agent, and two of your fellow classmate's agents (looping through all of them in a round robin fashion).

This ladder will be ranked differently than project 1. Players will be ranked first by the average number of kills (rounded to the nearest tenth). Ties will be broken by an agent's number of deaths with fewer deaths being preferred. Further ties will be broken by the number of beacons collected. **You can change this definition in class by discussion and vote.**

The first place player will receive one extra point for each night that that player wins the ladder up to a maximum of 5 points. To win, you must be ranked above the random player. The second place player will receive 1/2 extra point for each night that the player is

in second place. No player can receive more than 5 extra credit points from the ladder and points will be distributed down the ladder accordingly should the maximum be reached.

Wanted (alive please): creative individuals

Creativity is highly encouraged! To make this real, there are up to 10 points of extra-credit available for creative solutions. Some ideas here include: function approximation (this environment is perfect for function approximation), reinforcement learning, and hierarchical learning (e.g. learning to choose among the high-level behaviors and learning better low-level behaviors such as improved steering, shooting, or obstacle avoidance).

- Document it in your writeup! I can't very well give extra credit unless I know you did something extra.
- You must still be doing either evolutionary computation or reinforcement learning
- Remember that by being creative I am referring to the algorithm and not to the ability to creatively download code. All project code must be written exclusively by you except for the sample code that we provide.

Implementation details

We have implemented a few of the high-level behaviors using the provided A* search algorithm but you are VERY welcome to use your own code! The agent chooses among the high-level behaviors randomly - you need to add the intelligence!

A learning agent needs to store the learned values somewhere and we have given you a sample agent that uses xml to save the values across games. The agent has this ability already using xml (Xstream). You can choose to use xstream or you can implement your own input/output capability but you should load your knowledge file in initialize() and save it out in shutdown(). The example agent shows how to do this. Also note that isAlive() in ImmutableShip tells you when your ship is alive so you don't learn when it is dead.

Those pesky details

1. **Part 1:** Make a plan for how you will integrate two learning techniques into your spacewar agent. Turn this plan in along with a rubric for grading your second learning method (evolutionary computation is already specified) by **Tuesday March 17 at 12 noon**.

2. **Part 2 from here down.** Update your Spacewar code from project 1. We have changed the set of available actions by your request and you must update to obtain these changes! In addition, you may need to do a clean or else eclipse (or netbeans or whatever you are using to compile/edit) will be confused.

3. Download the *unpublished* code from D2L. Please do not distribute this code as it is intended for this year's class only. It contains solutions to project 1 (AStar code). Install this code in your src directory. When unzipped, it should create a directory called 'nonpublish'. You should use the example RandomGeneticAgent in src/nonpublish/agents to start your project.

4. Run the ant build target 'project 2' to see the sample project 2 agent working. You will need to copy the RandomGeneticAgent to your 4x4 directory and the clientinit.xml in examples/project2. You will need to modify TWO lines in clientinit.xml now. First,

   ```
   <clientClass>nonpublish.agents.RandomGeneticAgent</clientClass>
   ```

   The previous line should point to your 4x4 and your agent's name. If you name your agent, 4x4.MySpacewarAgent then change this package line to that.

   ```
   <data>../../src/nonpublish/agents/knowledge.txt.gz</data>
   ```

   The previous line should be modified ONLY to point at your 4x4. Leave the ../../ part alone as it will be necessary for the ladder. For example, I would make mine read:

   ```
   <data>../../src/my4x4/knowledge.txt.gz</data>
   ```

5. Create a ship that acts intelligently using learning as described above.

6. Ensure that your player runs on linux on the CSN class machines. You can ssh into these machines or go into the lab personally to verify this. Java should be in your path by default. If it is not, java lives in:

   ```
   /opt/java/bin/java
   ```

7. Submit your project on codd.cs.ou.edu using the submit script as described below.

(a) Log into codd.cs.ou.edu using the account that was created for you for this class. Your username is your 4x4 and your default password is cs#4x4.

(b) Make sure your working directory contains all the files you want to turn in. Using a similar submit command to the previous projects, you would use the following command to turn in your code.

```
/opt/ai4013/bin/submit CS4013 Project2 *.java clientinit.xml
```

(c) After the project deadline, the above command will not accept submissions. If you want to turn in your project late, use:

```
/opt/ai4013/bin/submit CS4013 Project2Late *.java clientinit.xml
```

Point distribution

- 50 points for correctly implementing evolutionary computation including fitness functions, chromosome/individual representations, selection, crossover, and mutation. Sample point distributions are given below:
  - 45 points if there is only one minor mistake. An example of a minor mistake would be incorrectly implementing selection, off-by-one errors in state calculations, incorrectly configuring your configuration files, and other minor coding errors. In the fitness function, a minor error would be correctly designing a useful fitness function but failing to implement it correctly (such as putting a minus sign in the wrong place or having an off-by-one error).
  - 40 points if there are several minor mistakes.
  - 35 points if you have one major mistake. An example of a major mistake would be accidentally starting your population over from scratch each generation (rather than using selection), failing to crossover individuals, using a completely not useful fitness function, and other major errors.
  - 25 if there are several major mistakes.
  - 10 points if you implement a learning algorithm other than evolutionary computation or RL that at least moves the agent around the environment in an intelligent manner.
  - 5 points for an agent that at least does something other than random movements.

- 30 points for your second learning approach. **You must provide a rubric of your own for this learning approach by March 17, 2009 at 12 noon.**
- 10 points: We will randomly choose from one of the following good coding practices to grade for these 10 points. Note that this will be included on every project. Are your files well commented? Are your variable names descriptive (or are they all i, j, and k)? Do you make good use of classes and methods or is the entire project in one big flat file? This will be graded as follows:
  - 10 points for well commented code, descriptive variables names or making good use of classes and methods
  - 5 points if you have partially commented code, semi-descriptive variable names, or partial use of classes and methods
  - 0 points if you have no comments in your code, variables are obscurely named, or all your code is in a single flat method.
- 10 points for your writeup. A full-credit writeup will describe your choice of fitness function and why you chose that function, your chose of knowledge representation (the individual, the state representation, any actions, etc) and why you chose that representation, your choice of learning parameters (selection, crossover, mutation, and any required parameters to your chosen function), your description of your second learning approach, any creativity that you implemented and why it worked or did not work. 5 points of this write up will be given for a learning curve with time (games) on the x-axis and performance on the y-axis. To get all 5 points, this curve should demonstrate actual learning (meaning performance improves over time). 3 points if you can at least give the curve and explain why you think it is not learning. Additional learning curves measuring different forms of performance are welcome.