**Project 2**
**CS4013/5013: Artificial Intelligence**
**Due 11:59PM March 24, 2011**

---

# Introduction

For this project, your task is to use genetic algorithms/evolutionary computation to create an intelligent pacman agent. Your agent should avoid the ghosts (newly upgraded to follow the original pacman logic!) and successfully clear any board your agent is given. This project will require you to design a useful encoding of your chromosomes, potentially to design a set of high-level/abstract actions for the agent, and to design an appropriate fitness function to guide the learning of your agents.

Learning will most likely be more successful with high-level actions rather than only UP/DOWN/RIGHT/LEFT. A sample list of high-level actions is provided below but this list is not exhaustive and you should define your own.

- Move to nearest energy pellet (uses A$^*$ or other search code)
- Move to nearest extra points piece (like the cherries)
- Chase down nearest ghost if in power mode
- Run away from nearest ghost

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.

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 as well.

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://en.wikipedia.org/wiki/Genetic_algorithm`

The following is a list of ideas for state space features. Note that this representation lacks a great deal of information. You can't make a general learning agent that uses Pacman's current x,y coordinates given that the boards will vary and you want him to play well in any board. You also can't represent every possible board state for every board in main memory so the problem must become partially observable!

- Distance to Inky, Blinky, Pinky, and Clyde
- Distance to nearest power pellet
- Number of power pellets left
- Number of regular pellets left

# 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: bugs or exploits in the simulator

We are reasonably certain that you cannot exploit the simulator. However, any project has bugs and we want to know about them! If you find a bug or an exploit, you can receive extra credit according to the following scale:

- **5 points:** If you find an exploit and report it to us you can receive 5 points extra credit upon verification of the report. Note, we know of two exploits for which you cannot get extra credit as they are already discovered. Since both are both extremely difficult to fix and extremely difficult to implement, I'm not listing them here.
- **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.
- **1-3 points:** General simulator bugs are much more likely than exploits. Finding a bug and reporting it 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 three 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 11th (the day after the project is handed out). As with project 1 and 2, you will be ranked by your pacman agent's average score on several different pacman boards.

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).

- 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
- 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. Update your aiproject code from project 3. We have made changes to the pacman API and you will most likely will not be able to directly run your earlier agents (but you can fix them with a quick examination of the new function calls in ImmutablePacmanState and ImmutablePacmanCharacter) In addition, you may need to do a clean or else eclipse will be confused. To clean your project, select the clean target from the ant build menu. You should only need to do this once.

2. We have provided a RandomGeneticClient in edu.ou.pacman.client that demonstrates how to use the simulator to create a generation of agents and how to save this knowledge to a file. Note that this agent has several supporting files in edu.ou.pacman.client.utils. Copy this agent (and any of the helper classes that you want) into your 4x4 directory and edit the pacmaninit.xml file in examples/hello to point to your new agent. Note that you will need to edit TWO lines in pacmaninit.xml now. First, the line below:

   ```
   <clientClass>edu.ou.pacman.clients.RandomGeneticClient</clientClass>
   ```

   needs to be edited to point to your 4x4 and your agent's name, as you did with the previous projects. The following line:

   ```
   <data>../../src/edu/ou/pacman/client/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>
   ```

3. Create a pacman agent that moves around the board intelligently using genetic algorithms as described above.

4. 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
   ```

5. 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. Don't forget to turn in your knowledge file too! Just list it on the command line below as a regular file.

   ```
   /opt/ai4013/bin/submit CS4013 Project4 *.java pacmaninit.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 Project4Late *.java pacmaninit.xml
   ```

6. As with the previous projects, you can test the performance of your agent on the test ladder using the following command. Don't forget to submit your knowledge file too! Just list it on the command line below as a regular file.

   ```
   /opt/ai4013/bin/submit CS4013 Project4TestLadder *.java pacmaninit.xml
   ```

# Grading rubric

The vast majority of your points go to correctly implementing evolutionary computation. These are broken out as follows. NOTE: CS 5013 students must implement a second approach for one of these categories and compare the results. For example, a CS 5013 student could implement two selection approaches or two fitness functions or two chromosome representations and compare the results in the writeup.

- 20 points for correctly implementing a useful chromosome representation that enables your agent to learn some form of intelligent behavior
  - 15 points if there is only minor mistake. This could include an error in actually representing your state such as an off-by-one error or using the wrong variable in the state.

  - 10 points if there are several minor mistakes or one major mistake. This includes a state representation that crashes some of the time or that simply is not useful for learning at all.
  - 5 points for several major mistakes
  - 0 points for only re-using the state representation we give you in the sample code. You need to change it!

- 20 points for correctly implementing a useful fitness function. This function must guide your agent's learning and should not tell the agents how to solve the problem.
  - 15 points for a minor error, such as 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).
  - 10 points for one major error or several minor errors such as making your fitness the opposite of what you want (e.g. making pacman behave worse rather than better!)
  - 5 points for several major mistakes
  - 0 points for using the provided fitness function (which returns 0 all the time)

- 10 points for correctly implementing a non-random crossover function
  - 8 points for a minor error such as accidentally not using both parents all of the time
  - 5 points for a random crossover function or for a function with a major error

- 10 points for correctly implementing mutation in a way that helps learning
  - 8 points if your mutation rate is WAY too high or if you have a minor error (such as trying to flip a bit but not actually doing it)
  - 5 points for a minor error

- 10 points for correctly implementing a non-random selection function.
  - 8 points if you have one minor mistake in selection (such as accidentally using the wrong individual sometimes)

  - 5 points for a major mistake. An example of a major mistake would be accidentally starting your population over from scratch each generation (rather than using selection) or implementing random selection, either on purpose or by accident.

- 10 points for implementing at least one high level action. Pacman will likely not learn well with only the low-level UP, DOWN, RIGHT, and LEFT actions so you should implement high level actions. This also provides amble opportunity for creativity in your solutions!

- 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), 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 curves measuring different forms of performance are welcome. Note, CS 5013 students MUST compare their results of the two different methods (see above) in their writeup for full credit.