In this assignment you will experiment with a genetic algorithm as a method for evolving control strategies for Robby the Robot.

**Robby, the Soda-Can-Collecting Robot**

Robby’s job is to clean up his world by collecting empty soda cans. As described in class, Robby’s world consists of 100 squares (sites) laid out in a 10×10 grid. Robby starts in site 0,0. Imagine that there is a wall around the boundary of the entire grid. Various sites have been littered with soda cans (but with no more than one can per site).

Robby isn’t too intelligent, and his eyesight isn’t that great. From wherever he currently is, he can see the contents of one adjacent site in the north, south, east, and west directions, as well as the contents of the site he’s currently in. A site can be empty, contain a can, or be a wall.

For each cleaning session, Robby can perform exactly 200 actions. Each action consists of one of the following seven choices: move to the north, move to the south, move to the east, move to the west, choose a random direction to move in, stay put, or bend down to pick up a can. Each action may generate a reward or a punishment. If Robby is in the same site as a can and picks it up, he gets a reward of 10 points. However, if he bends down to pick up a can in a site where there is no can, he is fined 1 point. If he crashes into a wall, he is fined 5 points and bounces back into the current site.

Clearly, Robby’s reward is maximized when he picks up as many cans as possible, without crashing into any walls or bending down to pick up a can.
when no can is there.

Your assignment is experiment with code that evolves control strategies for Robby the Robot. This code can be downloaded from the class web site:

http://www.cs.pdx.edu/~mm/AIFall2011/GeneticAlgorithm.tgz

Encoding of Strategies

Robby’s strategy is be encoded as a look-up table that gives, for every possible state—i.e., situation he can encounter—the action he should take when in that state.

There are five different sites (north, south, east, west, current), each with three possible types of contents (wall, empty, can). Thus there are $3 \times 3 \times 3 \times 3 \times 3 = 243$ different “possible” situations. Of course this number includes some “impossible” situations, such as those in which Robby’s current site contain a wall (since Robby will always bounce back from a wall). However, you don’t have to filter these out; you can just leave them in the table, since they won’t have much affect on the GA.

Here’s an example of a strategy—actually, only part of a strategy, since an entire strategy would be too long to list here.

<table>
<thead>
<tr>
<th>Situation</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>North</td>
<td>South</td>
</tr>
<tr>
<td>Empty</td>
<td>Empty</td>
</tr>
<tr>
<td>Empty</td>
<td>Empty</td>
</tr>
<tr>
<td>Empty</td>
<td>Empty</td>
</tr>
<tr>
<td>Empty</td>
<td>Empty</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>Wall</td>
<td>Empty</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>Wall</td>
<td>Wall</td>
</tr>
</tbody>
</table>

For example, suppose Robby’s current situation is
To decide what to do next, Robby simply looks up this situation in his strategy table, and finds the corresponding action to take; in this example, the action is MoveWest.

The “chromosome” to be evolved by the GA is just a listing of the 243 actions in the rightmost column of the strategy table, in the order given. The actions are numbered as:

- 0: move north
- 1: move south
- 2: move east
- 3: move west
- 4: stay put
- 5: pick up can
- 6: move in a random direction

Then the chromosome representing the strategy in the example above would be

\[ 0234 \ldots 3 \ldots 6 \]

The GA remembers that the first action in the string (here action 0: MoveNorth) goes with the first situation (“Empty Empty Empty Empty Empty”), the second action (here action 2: MoveEast) goes with the second situation (“Empty Empty Empty Empty Can”), and so on.

Here are the steps in the genetic algorithm.

A. **Generate the initial population.** The GA starts with an initial population of POPULATION_SIZE random individuals (strategies). As described above, each individual strategy is a list of 243 numbers, each gene between 0 and 6, which stands for an action (0 = MoveNorth, 1 = MoveSouth, 2 = MoveEast, 3 = MoveWest, 4 = StayPut, 5 = PickUp, and 6 = RandomMove). In the initial population, these numbers are filled in at random.

B. **Repeat the following for NUM_GENERATIONS generations:**

1. Calculate the fitness of each individual in the population.
The fitness of a strategy is determined by how well the strategy lets Robby do on NUM_ENVIRONMENTS_FOR_FITNESS different cleaning sessions. A cleaning session consists of putting Robby at site 0, 0, and throwing down a bunch of cans at random (each site can contain at most one can; the probability of a given site containing a can is 50%). Robby then follows the strategy for NUM_MOVES actions in each session. The score of the strategy in each session is the number of reward points Robby accumulates minus the total fines he incurs. The strategy’s fitness is its average score over the NUM_ENVIRONMENTS_FOR_FITNESS different cleaning sessions (each of which has a different configuration of cans).

2. **Sort the population by fitness** Assign to each individual in the population an integer: 1 for the individual with highest fitness, 2 for the individual with next highest fitness, and so on. Ties are broken at random. The population is sorted in this rank order.

3. **Apply evolution** to the current population of strategies to create a new population. That is, repeat the following until the new population has POPULATION_SIZE individuals:

   - Select two parent individuals from the current population probabilistically. There are several ways to perform this selection; different options are given in the code.
   - Mate the two parents to create two children. That is, randomly choose a position in each number string; form one child by taking the numbers before that position from parent A and after that position from parent B, and vice versa to form the second child.
   - With a probability MUTATION_RATE, mutate numbers in each child. That is, for each number in the child’s chromosome, with probability MUTATION_RATE replace that number with a randomly generated number between 0 and 6.
   - Put the two children in the new population.

4. Once the new population has POPULATION_SIZE individuals, return to step 2 with this new generation.

Finally, you can use the `calculate-performance` routine to test the generalization performance of the highest-fitness strategy in the final generation. This
routine inputs a chromosome and calculates its average score over 10,000 new cleaning sessions.

**Your assignment**

1. After downloading the genetic algorithm, code, type “make” to compile it. I can guarantee it will work on the CECS Linux machines, but no guarantee for other platforms.

2. Come up with your own hand-devised strategy for Robby, write a program to encode it as a 243-number string, and use `calculate-performance` to calculate its performance. In your writeup, describe your strategy and give its 243-number code and the performance you calculated.

3. Run Robby with the following parameters (you will need to edit the `params` file):

   NUM_RUNS 10
   NUM_GENERATIONS 500
   POPULATION_SIZE 200
   NUM_ENVIRONMENTS_FOR_FITNESS 200
   NUM_MOVES 200
   CROSSOVER_RATE 1
   MUTATION_RATE 0.005
   SELECTION_METHOD “pure rank”
   SHORT_PRINT TRUE
   LONG_PRINT FALSE
   BEST_PRINT FALSE

   (All other parameters should be left at their default values.)

   You will also need to change the RUN_NUM_DIR and the OUTPUT_DIR in the params file to point to your own directory.

   To run Robby, simply type “robbxy”.

   This will perform 10 independent runs with the parameters you have set. Record the average fitness of the best individual in the final generation over these 10 runs.

4. Repeat the previous experiment, but with CROSSOVER_RATE set to 0. I.e., each parent has one “cloned” child, which is then subject to
mutation. Record the average fitness of the best individual in the final generation over these 10 runs.

5. Same as the previous experiment, but increase the population size to 500.

6. Same as the previous experiment, but decrease the population size to 100.

7. Design your own experiment by changing one or more parameter values, or by changing something about Robby’s task (e.g., the reward structure).

8. Finally, for each experiment in steps 3–7 above, choose one of the 10 runs per experiment and plot the highest-fitness in the population as a function of generation number.

What to turn in

Turn in a clearly written and formatted writeup with a summary of your results and discussion. Your write-up should describe your hand-designed strategy, give its code and performance, describe and give the results of your experiments in steps 3–7, comparing these results with the performance of your hand-designed strategy, give your plots, and include a paragraph or two describing your hypotheses about the results (e.g., hypothesize as to why there is a difference between the results of the original GA run and the run without crossover, and so on). Send your final writeup to Josh Hughes (hughesjg@cecs.pdx.edu).

Please don’t hesitate to ask questions (to me or via the class mailing list) if you don’t understand something in this assignment.