Notes on Assignment 1 (AI2016)

Julian Togelius

October 5, 2016

Abstract

Below are some notes on assignment 1 for the AI course, more specifically on how to use the Ms Pac-Man framework, how to implement the various algorithms and how to write the project report.

1 The Pac-Man source code

The following are the most important classes and methods to look at in the Ms Pac-Man framework.

1.1 pacman.entries.pacman.MyPacMan

The class template you will use to implement your Pacman controller. Implement the method *public MOVE getMove(Game game, long timeDue)*, which receives the game state and how long do you have to make a move, and returns the move pacman will make (for example: MOVE.LEFT to go left, MOVE.NEUTRAL to remain with the same direction).

1.2 pacman.game.Game

This is the main class, which contains information about levels, the Pac-Man, ghosts and pills. This is the class you will clone and use to simulate your game. **We suggest you check all the methods in this class and what they do.** If you have a *Game game* object, some useful functions are:

game.copy() copies and return a Game instance.

game.advanceGame(pacManMove, ghostMoves) Simulate one step of the game: the Pacman moves according to *MOVE pacManMove*, and the ghosts follow *EnumMap<GHOST,MOVE> ghostMoves*.

game.getPossibleMoves(nodeIndex) Receives a *int nodeIndex*, which is the index of Pacman's position, and return all possible moves he can make (MOVE[]).

game.getPacmanCurrentNodeIndex() Return where PacMan is.

So, for example, if you want to return a random move, you could do:
MOVE[] moves = game.getPossibleMoves(game.getPacmanCurrentNodeIndex());
return moves[new Random().nextInt(moves.length)];

1.3 pacman.Executor

The main class of the project. It contains several methods that run the game, such as *runGameTimed(Controller<MOVE> pacManController, Controller<EnumMap<GHOST,MOVE>> ghostController, boolean visual).*

pacManController is the controler you implemented.

ghostController can be any GhostController already implemented in the framework in the package **pacman.controllers.examples** (such as StarterGhosts or AggressiveGhosts).

visual defines whether it will render the interface.

2 Notes on implementing the algorithms

The first thing to note is that your algorithms do not all need to play the game you have chosen well. In fact, several of the algorithms will not do well on any of the games in their "naive" or "vanilla" form. For example, an unmodified depth-first algorithm will almost certainly time out on all of the games. It is up to you whether to improve each algorithm so that it performs well on the game you have chosen. I will expect you to have at least 2-3 algorithms playing your game to a decent standard, but it's perfectly fine to explain that some other of your algorithms could not play your game at all. This will not affect your grade, as long as you have tested the algorithms and *explain why they underperform.* In general, I am very keen on you explaining both successes and failures. On the other hand, it is perfectly fine to try to make all these algorithms play well on your game by modifying them. All algorithms can be made to play Pac-Man decently well with fairly minor modifications.

There are always multiple way of implementing each algorithm in a game, and there are often multiple versions of an algorithm. You have some freedom here (more on certain algorithms than on others), but the important thing is that you make it clear what your design choices are and why.

However, let me take this opportunity to reinforce that you when you implement tree search algorithms, you should *always be searching in game state space, not just in physical space.* This is important, so I'll put it in bold as well. **You should always be searching in game state space, not just in physical space.** This means that you expand a tree node (find successor nodes) by taking actions in copies of the game state. Each node corresponds to a particular game state. If your tree search algorithm does not include copying of the game state and taking actions in these copies, you are doing it wrong. If the core of the algorithm includes getting the graph of the level, you are most likely doing it wrong (if you are accessing this graph inside an evaluation function it's another thing–that is a perfectly sensible thing to do).

When you implement your algorithms, try to not build on any of the example agents. Write your own code from scratch, while conforming to the interface.

2.1 Uninformed search

To illustrate how to implement basic tree-search algorithms, below is somewhat abstracted Java code for a depth-first search algorithm. To use this, you need to replace the method calls (copy(), value() etc) and object names (Move, State etc) with the appropriate class names and method calls for the Ms. Pac-Man framework. See the previous section for what those classes and methods are called. No, you can't just copy-paste the code, because I did not want to simply give you the solution to part of the assignment.

```
move dfsRoot (State state) {
  int best = 0;
  Move bestMove;
  for (Move move : state.moves ()) {
    copy = state.copy ();
    copy.makeMove (move);
    copy.advance ();
    int value = dfsRecursive (state);
    if (value > best) {
      best = value;;
      bestMove = move;
    }
  }
  return bestMove;
}

int dfsRecursive (State state) {
  int best = 0;
  for (Move move : state.moves ()) {
    copy = state.copy ();
    copy.makeMove (move);
    copy.advance ();
    if (copy.terminal ()) {
      int value = copy.value (); // won or lost?
      if (value == WIN) return value;
      if (value > 0) best = value;
    }
    else {
      return dfsRecursive (copy);
    }
  }
  return best;
}
```

Even when you integrate this with your Most likely, it will time out because the algorithm times out without coming to a win state. If you want to make it perform decently on your game, try making it depth-limited and use a state evaluation heuristic, for example the score. Start with a low depth limit (maybe 10 steps?) and see how far you can take it.

You will also need to implement breadth-first search (again, what's your depth limit?) and iterative deepening.

2.2 Best-first search

Best-first search (A*) requires a heuristic that gives an admissible estimate of the distance to a goal state. But what is a goal state? In Pac-Man, it could be having eaten all the pills; the distance could simply be the number of remaining pills. (Make sure that the multiplier is so small that it would never take fewer number of steps to eat all those pills if they were in a straight row.)

2.3 Optimization and evolutionary algorithms

The various optimization algorithms (hill climber, simulated annealing, evolution strategy and genetic algorithm) can be used in different ways, but the way you should use them for this assignment is to evolved action sequences. So each genome in the population is a sequence of a number of actions (maybe 10 or 20). Mutation changes one or several actions; crossover simply combines part of one action sequence with part of another. The fitness function is an evaluation of the game state resulting from each action sequence. Just like for the tree search algorithms, you then take the first action of the best sequence, before searching for a new sequence next time step.

Note that this goes for all the optimization algorithms, including the hill-climber and simulated annealing. The chromosome/individual/search point is not a point in space or a game state, but a sequence of actions.

The difference between evolution strategy and genetic algorithm is not super clear cut. Both are evolutionary algorithms; traditionally, the genetic algorithm is heavily reliant on crossover and uses fitness-proportionate selection, whereas the evolution strategy only uses mutation (no crossover) and rank-based selection. It's pretty much up to you what exactly you implement here. The important thing is that you implement two different evolutionary algorithms.

3 The report

The assignment is to be done individually and should be handed in via NYU Classes.

You should hand in a one-page report in pdf format. The report should include your name and NYU email id (the one on your ID card, e.g. jt125). It is strongly suggested you write your report in LaTeX using Overleaf (overleaf.com), using their Sample Paper template. You just click on "new project" and then on "sample paper". This very document is written using that template. You can also use something like Word to write the report: if you do that, please use this document (which is written using the standard paper template in Overleaf) as a guide to margin sizes, font size etc.

The assignment report should detail how you implemented all the algorithms. There are several implementation choices for each of the algorithms, and you should motivate all your choices. You should also provide benchmark figures for all your algorithms—what score did they get? Try to explain why some algorithms did better than others.