Overview of eiisolver Pacman/Ghosts

Louis Verhaard, 2012

During the spring of 2012 I participated in the Ms Pac-man vs Ghosts Competition organized by Philipp Rohlfshagen, David Robles and Simon Lucas from the University of Essex, see http://www.pacman-vs-ghosts.net. The competitors are computer programs that play either the role of Pacman or of the ghosts in the classical Pacman arcade game.

This document describes the design of my participant, eiisolver, that played in both competitions.

Background

I had originally no intention of joining, because I know when I start with this kind of stuff I can get completely absorbed, and spend way too much time. But when I saw the ongoing games on the web site I thought "hmm, it should be easy to finish high up in the list". And it was a long time ago since last time that I indulged in useless, absorbing programming activities, so suddenly one evening I found myself programming the first lines of code.

Because I wanted it to be as fun as possible, I deliberately did not downloaded or looked into source code released during previous competitions and I did not read anything about how previous participants approached the problem; I wanted to invent everything myself.

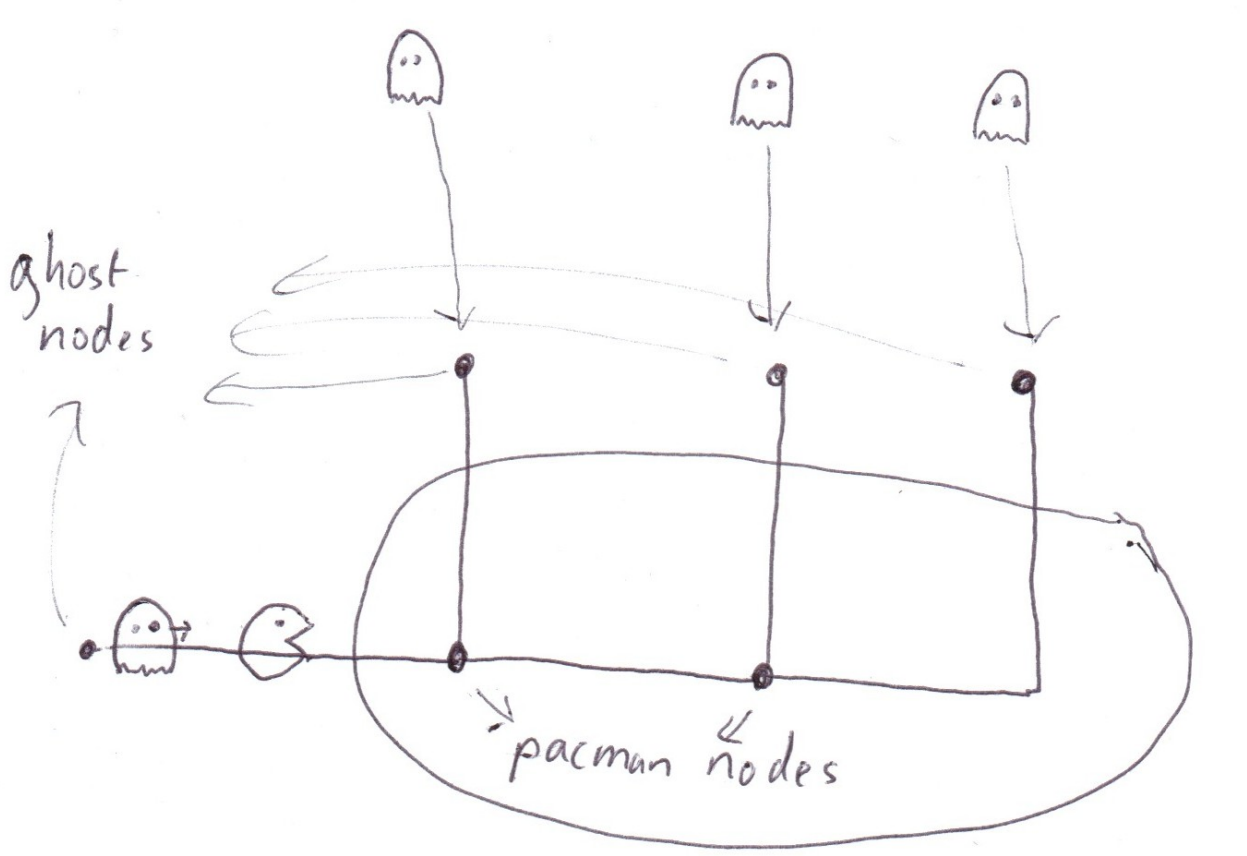
I have programmed games like chess before, and after a bit of analysis I determined that I would use a similar approach in Pacman, vizually the search engine would be based on alpha-beta minimax search.

In the game of Pacman, the number of moves per ply is limited (a ply is a half-move, i.e. a move for pacman, or a combination of moves for the ghosts, 2 ply is 1 complete move). So it should be possible to search quite deep, even with the allocated 40 milliseconds per move.

Static Evaluation

At end nodes and also at many nodes that are internal in the search tree, a static evaluation is performed. The goal of the static evaluation is to determine if Pacman is safe or not. The static evaluation can also give a result meaning “Pacman is safe, but it is forced to take a power pill”. This is important because we want to save power pills to be used for the right moment.

The static evaluation starts with building a “border edge graph”. It calculates which nodes are closest to Pacman (called pacman nodes) and which nodes are closest to the ghosts (called ghost nodes). Edges that have one ghost node and one pacman node are called border edges. The result is a graph (a sub-graph of the total board). For every ghost node in the graph it is calculated which ghosts are closer than pacman.



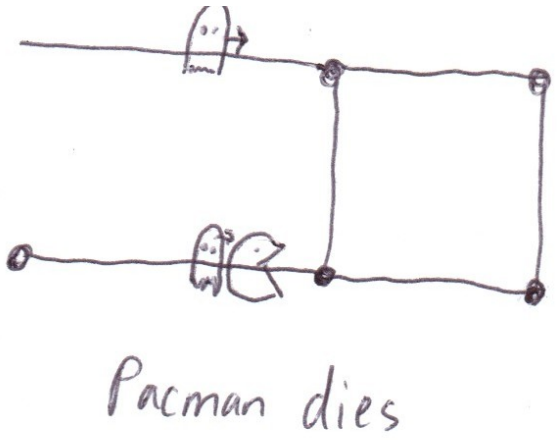
When we have calculated this graph, we perform a match: we try to assign ghosts to border edges such that as many border edges as possible are covered by a ghost. If all border edges are covered, and the graph does not contain any circle, then we know that Pacman will die: every ghost will just walk into its assigned border edge and from there on walk towards Pacman, there is no way that Pacman can escape, unless Pacman can reach a power pill.

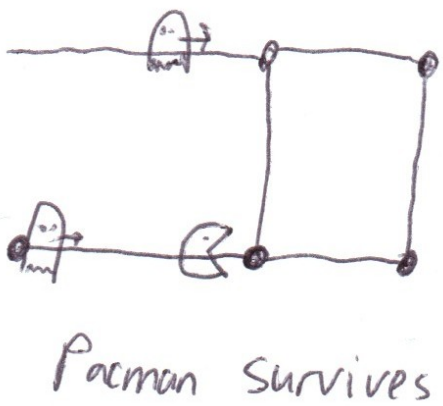
If there is a match but the graph does contain a circle, the situation is more unclear. It could be that Pacman can escape. Eiisolver cannot analyze this situation very well (in fact it does not even recognize this situation correctly) and just uses some heuristics. Fortunately, circles in border edge graphs are much more seldom than trees.

Tree Match

When the border edge graph is a tree, and we found no match, a “tree match analysis” is done.

Consider the two situations in the figure below:





In the left situation, Pacman dies. When it reaches the nearest junction, it must choose to go up or to the right. At this moment, the upper ghost has not yet reached its closest ghost node, so it can wait with its decision to go down or right after Pacman has chosen.

On the other hand, in the right situation, Pacman survives. It can walk to the closest pacman node, and wait for the upper ghost to get to the closest ghost node. If the ghost goes down, Pacman goes right, if the ghost goes to the right, Pacman goes up.

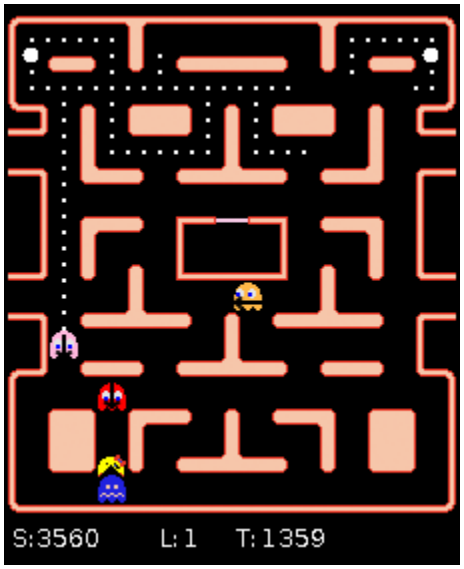
We see from this example that a single ghost can sometimes cover more than 1 edge. The tree match algorithm calculates whether pacman can escape or not for border edge graphs where the number of border edges exceeds the number of ghosts.

In general, the tree match algorithm checks for ghosts that are assigned to multiple border edges, and then calculates pairwise when the ghost must choose which one to cover, and also calculates how long Pacman can wait until it must chose which border edge to follow. The ghost can cover both edges if pacman must make its decision before the ghost.

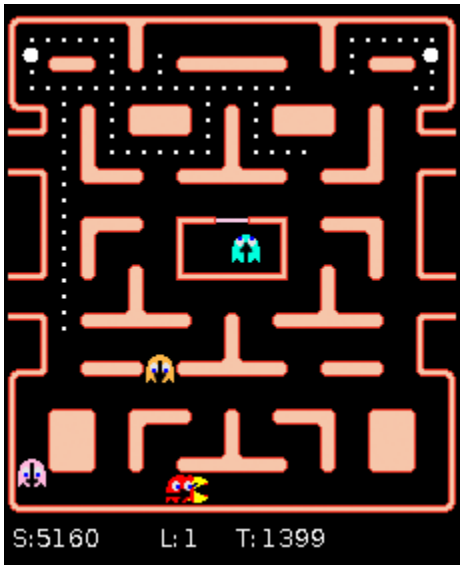
I found it quite hard to implement this correctly, and the current implementation is not 100% accurate; there are some situations where multiple ghosts cover multiple border edges where it thinks pacman survives, even though there is some combination of assignments that cover all edges.

But mostly the tree match works very well, and has helped to reduce the many “accidents” in the bottom of level 1 where it previously died many, many times.

An example of the static analysis in action: below picture is from a game between an old version of eiisolver (=pacman) against Kajikami (= ghosts). It was this game that finally made me understand the dynamics of tree matching. In the picture, the tree match analysis reveals for the first time that pacman will die.



Some moves later, the situation is easier to grasp:



Pacman has 4 border edges, but cannot escape because the brown ghost covers 3 of them. Two single ghosts capture pacman, I found this quite amazing.

A static evaluation is a very expensive operation that usually takes several microseconds to compute. Because the same situation is often encountered many times (because of iterative search deepening, and because we have seen many situations already during the search of a previous move), static evaluation results are stored in a cache, and reused when we encounter the same situation again.

Alpha-beta search

A standard iteratively deepening alpha-beta search is used as the search engine. The search is of variable depth: if there is no choice (i.e. pacman or ghosts can only make 1 move) at some node, the node is “free”, i.e. does not count.

For alpha-beta search it is important to have a good move ordering. The more often the best move is

investigated first, the fewer nodes are required for searching. Two mechanisms are used in eiisolver to improve move ordering: transposition tables and “killer moves”

By far the most important of these two is the transposition table. A transposition table (commonly used in chess programs) is used to store search results. Please check some description of chess programs for details how such tables work. eiisolver only uses a basic implementation. The transposition table greatly improves search depth, because we can sometimes skip searching whole sub-tree. Often the search result of a stored transposition table entry cannot be used directly, for example because it was obtained with a low search depth. In that case we will investigate the move stored in the transposition table first. With a very high probability it will also be the best move during the new, deeper search.

“Killer moves” is a another mechanism for move ordering, inspired by a heuristic with the same name that is used in many chess programs. The implementation in eiisolver is quite simple. Suppose pacman is at some junction somewhere during a search, and the search result says that pacman should go to the right. Then next time during the search where pacman is at that same junction, first the move to the right is investigated, even though the ghosts may be at completely different locations. Chances are good that it is still the right move even in this situation. I have not measured scientifically how effective this mechanism is, my feeling was that search depths were marginally improved.

Selective searching

Initial versions of eiisolver considered all pacman moves and all ghost moves. This gives 100% accurate search results, but the obtained search depth is not so great and results in weak play. By not considering all possible moves we can search much deeper, but at a risk: accidentally it might be the case that the best move was one that we did not consider, basically making the search result useless.

Eiisolver never considers any pacman move towards a ghost on the same edge (except during the first move in a search) unless the ghost is edible, or the edge contains a power pill. I was not able to come up with many cases where moving towards a ghost on the same edge would be significantly better than moving away from the ghost. The only case I know of that is currently not detected by eiisolver: if the edge contains the very last pills of the board.

Also pacman reversals are often not considered. If, during a search, pacman's last move was to the left, then usually the search will not consider a pacman move to the right. There are some exceptions where reversal moves are considered anyway:

- Pacman is at a junction, or pacman's previous location was a junction. It is very important in certain situations (see “tree match” section for a discussion) to be able to “wait out” a ghost in order to escape. Optimally, pacman should consider “neutral” moves at junction nodes. But eiisolver's implementation does not support this (lack of time), so instead reversals after junctions are considered.
- If some ghost's previous location was at a junction.
- If pacman is exactly on the middle between 2 junctions, or at a distance of 7 from a junction.

I think there are some cases where eiisolver incorrectly skips pacman reversals, and most likely this part can be improved somewhat. But overall these heuristics gave a large boost in playing strength.

Extended Search

At end points in a search, a static evaluation is made as explained earlier. If everything looks safe to

pacman, no further searching is conducted. But if the result of that evaluation looks dangerous (for example there are 5 border edges, and 3 different ghosts that are closer to any of these border edges), the search is continued, but now the search becomes much more selective.

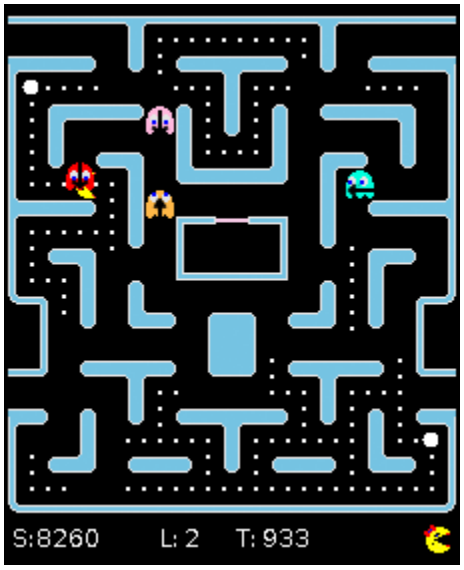
During an extended search, also ghost moves are skipped. The results of the static evaluation are used to set up targets to the ghosts. One of the results of the static evaluation is an assignment of ghosts to targets. An example of such an assignment could be: “Inky should move to junction A, Blinky should move to junction A or B, Pinky should move to junction C, Sue has no assignment”.

- a ghost that has no targets assigned will just try to move closer towards pacman (in the example, Sue will just move towards pacman)
- a ghost that has one or more targets assigned will move towards any of these targets. When it has reached a target it can choose to pursuit pacman, or move to another target.
- a ghost that has reached its target will continue to chase pacman

The extended search is not 100% accurate, but it increases the search depth enormously and gave big improvements in results.

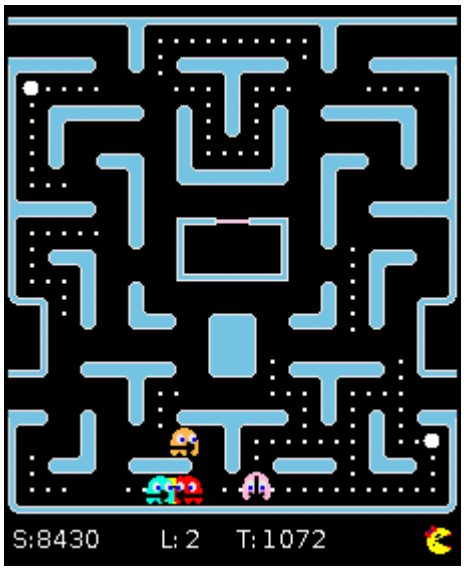
Example

An example from a test game played at home: in the below situation, pacman sees that it will loose.



Static analysis sees that pacman will loose in the below situation:

At the moment of death:



All in all, eiisolver searches quite deep. Peak searches are usually more than 100 moves (200 ply) deep, and together with the static analysis it will almost always discover 100-150 moves (200-300 ply) in advance that pacman will die, and in sometimes even earlier.

The Ghost Magnet

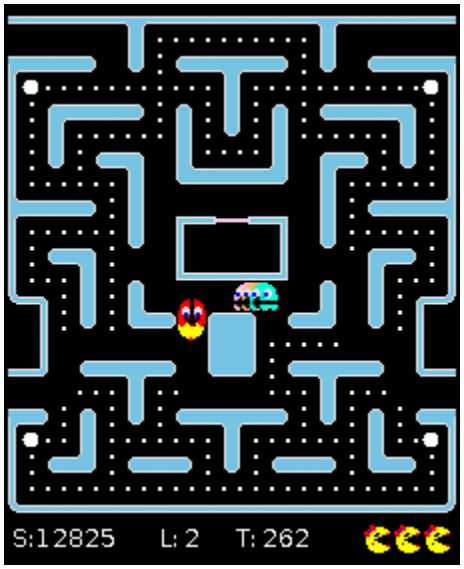
It turned out that for the Pacman competition, the most important thing that mattered was to score as high as possible against the weaker ghosts. I spent only very little time on this because I found this aspect of the game much less interesting than working on the search engine. First 2 days before the end of the competition I really sat down and tried to improve this, because especially maastricht and the ICEP team had taken “ghost eating” to an art and showed consistent monster scores against weak ghosts, thereby taking the two top spots.

I had little hope to beat these two formidable competitors, but also some other controllers had been surpassing eiisolver's ghost eating capacity (haimat, ghostbuster, sir macelon) and I had always been

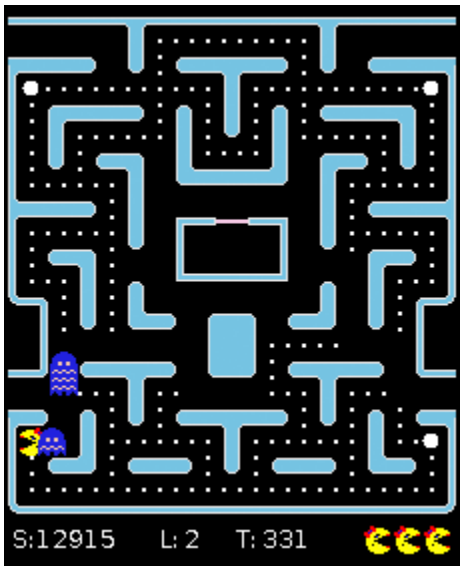
behind the much admired Memetix. So: I almost desperately sat down and tried to improve the ghost eating capabilities of my pacman.

I did a lot of tuning and found numerous bugs related to ghost eating.

A more funny feature implemented during these days was the ghost magnet. The ghost magnet comes into action when all ghosts are very close to pacman and there is a power pill that is closer to pacman than to any ghosts. See for example picture below.



When the magnet is activated, Pacman will first play neutral moves until the closest ghost is at minimal distance, and then it will walk to the power pill.



If pacman is lucky, the ghosts follow pacman obediently to the power pill and then die a horrible death. But if a ghost deviates from the "ghost train" as Bram Ridder (morloth) calls it, the ghost magnet is immediately deactivated.

My original idea was a little bit more advanced; in the above pictures I wanted pacman to follow a path along the bottom of the board towards the lower left power pill because the segment power pill - junction is longer, which on average would lower the time required to eat up all the ghosts.

So I tried to influence the search so that it would encourage pacman to follow the desired path. However this turned out to be too difficult to implement within the little time I had left. So in the end I settled for a solution where pacman just runs along the shortest path to the power pill. Actually no searching is done at all when the ghost magnet is active. There is no risk that pacman can die because of moves made while the magnet is active, because we know pacman is closest to a power pill.

Against weak Pacmans

For the ghosts competition it was important to obtain good scores against starter pacmans. I admit that I use an opening book containing 7 positions that often help to kill of the first two lives of starter pacmans. I am not particularly proud of this, but on the other hand I feel not guilty either...

Results

The challenge that I set for myself was to create a pacman that could survive against any ghosts. I failed miserably. It took considerable effort to create a pacman that could survive against the starter ghosts. And the starter ghosts do not search at all, they just use a few very simple heuristics. And even the latest version of eiisolver gets many times per game against starter ghosts into situations where it sees that it would lose if the ghosts would play correctly.

Surviving during 16 levels against a bit more advanced ghosts that do a little bit of searching and have some heuristics that aim at surrounding pacman is almost impossible, even if my pacman completely out-searches such ghosts. And against the top ghosts my pacman (nor any other pacman) does not stand a chance.

I found this a bit disappointing, and to my taste it made the competition a bit less interesting. I ended up with a pacman that searched much deeper than I imagined when I started, but the effect of all this work

was quite marginal. Instead I should have spent much more time on heuristics to improve results against weak pacmans/weak ghosts. But it does not matter much, it was great fun to try to come up with creative solutions.

Acknowledgements

Thanks to the organizers, they did a great job and were very responsive to bug reports and improvement suggestions.

And thanks again to my wife Anna for her understanding of my mental absence the last 6 weeks.