Assignment 3 - Group Challenges
DD2380

GROUP3

Mart Kartasev
19920421
kartasev@kth.se

Henrique Machado
19940105
hfm@kth.se

Abstract

Games have long been a focus of Artificial Intelligence research. From chess to real-time strategy games, they prove an interesting idealized testing scenario for algorithmic approaches that can be later be applied in different fields. In this project we implement an agent for the Pac-Man Capture the Flag game, which was originally developed by University of California, Berkeley. We apply an approach of search with heuristic game-state evaluation functions, coupled with a forward pass filtering algorithm to deal with uncertainty. Using these well-established methods we implemented a capable agent, achieving reasonable results with plenty of improvement potential.

1 Introduction

Pac-man capture the flag is part of the UC Berkley's Introduction to Artificial Intelligence course. It is a modification to the original 1980's arcade game Pac-man where, instead of playing against the computer and it's ghost agents, there are two teams with two agents each competing to capture and return the greatest amount of dots. In this project we were responsible of implementing an AI capable of controlling the agents and playing the game, and later compete against other student's AIs in a tournament. Following is a brief explanation of the rules of the game which can be seen more in depth on the UC Berkeley's AI course's website [8].

1.1 Rules

The Pac-man map is divided into two halves, one red and one blue, each one corresponding to a team. Each team controls two agents, which are ghosts on their half of their map and pac-mans on the opponents half. When in pac-man form, the agent is capable of eating dots, and when he crosses back to his half of the map, the dots are added to the scoreboard. However, if the pac-man is touched by a ghost before delivering the dots, he returns to his starting position and the dots he was carrying are dropped back onto the board. There are also power capsules, which when eaten makes the enemy ghosts vulnerable to opposing pac-mans, causing them return to their starting positions if eaten by a pac-man.

The observation of enemy agents is uncertain if they are farther than 5 squares away. This means that the position given by the program could not correspond exactly to where an enemy agent is, and could vary by six squares in any direction. The winner is the team which returns all but two or their opponents dots or the team with the highest number of returned dots after 1200 moves.

1.2 Outline

We explore the relevant research and available material on different aspects of strategy games, in relation to the problem at hand in section 2. Even though it's arguable if this is a strategy game, we treat it as such as all the methods are equally applicable. As most of the methods used are well-known applications of established ideas, we aim to give an explanation of

how and why they are used in the methods section (3) as well as describing any differences from normal practice which we had to apply for our problem. As this solution is part of a general competition between groups, the results of all the groups, including ours, are listed in section 4, which also includes the tournament specification and analysis of results. The summary of our work will be included in section 5, with a final evaluation and ideas for future improvements.

2 Related work

A general overview of strategy game related techniques is given by Glen Robertson and Ian Watson in a Review of Real-Time Strategy Game AI [5]. As the paper discusses, there are multiple options that could be suitable for our problem. Given the relatively concrete structure and a small number of agents, we decided to try solving the problem with search algorithms.

Minimax is a widely used and studied algorithm for adversarial games, with a long history of use and different formulations and optimizations [3] [4]. The algorithm consists of going through a tree which is generated by considering every possible move from each game state until it arrives at a terminal board state where the utility of that node is computed. It then considers if the agent making that move is looking to minimise or maximise the scoring, and these values are backed up though the tree as the recursion unwinds. As this tree's branching factor can be very large for more complex games, the use of fixed depth for the search, and heuristics for the utility computation is very common [7].

Expectimax or expectiminimax is a modification to the traditional minimax algorithm. Normally used for games where some actions are taken by chance, the utility of the nodes where the next step is generated by chance is calculated to be the average of the utility of all child nodes. Since in our project we know that the opponent agents are not using the same evaluation function as our agent and will most likely not take the action predicted by our search, we decided to use expectimax and consider that our opponents were choosing actions at random.

Dealing with uncertainty in readings is a well established problem - in both literature and practice - being taught as part of many AI courses in universities and also used in industry at large. A lot of our information regarding dealing with the fuzzy observations comes from Artificial Intelligence, A Modern

Approach [6], in the form of what is commonly known as filtering. Filtering can be used to reduce uncertainty for many different problems. In robotics, its more common to use Kalman Filters as they are intended for use in continuous spaces [1]. Though the principles are similar, Kalman Filters are not applicable in discrete space-state models such as this. The Forward Pass algorithm can be a very computationally efficient means of dealing with the sort of uncertainty we are presented with. Most of the available research is regarding more advanced techniques from language and speech modeling to intrusion detection, which shows how versatile HMMs can be, however is mostly irrelevant for the problem at hand. A potentially viable application could be predicting the movements of the enemy based on a the Baum Welch algorithm, as suggested in [2], but in our solution we decided to try and model enemy behavior using search algorithms.

3 Our method

3.1 Expectimax and heuristics

As discussed before we decided to use the expectimax algorithm and consider that our opponents actions as random. Our search tree is generated starting with a maximising move from our agent followed by two random moves from each of our opponent's agents, and so on. The maximum branching factor of this tree is five, since the possible moves are going to any of the possible four directions or standing still. However, usually not all of these moves are possible. After some testing we realised that the maximum depth our algorithm could get to without exceeding the amount of time given by the game for computation was four. Because of that, good heuristic functions would be needed to get our agents to perform well. We decided to focus on creating two different heuristic functions: one defensive and one offensive. That way we could have one of the agents exclusively on defensive duties while the other focused on capturing dots.

The defensive agent was supposed to stay on our side of the field. Because of that we heavily penalised states where our agent was a pac-man. To make sure our agent would pursue enemy pac-mans we also penalised the state depending on the distance between the closest enemy pac-man and our agent, incentivising our agent to get as close as possible to enemies. To make sure our ghost would eat the enemies we also added a negative score based on the number of enemy pac-mans on the map. Finally we made sure to invert

the distance score if out ghost was scared, because in that state he would be unable to actually eat the enemies.

For our offensive agent we actually had two different modes: one for retrieving dots and one for bringing them back to our side. The agent starts out trying to retrieve dots, but once it gets to a threshold which depends on the number of dots it is carrying and the distance to the closest ghost, we switch it to the return mode. On the return mode the heuristic heavily incentivises increasing the distance to the closest enemy so we avoid getting eaten, and decreasing the distance to the agent's starting position, which is in our base. On the retrieving mode the agent tries to minimise the number of foods on the map and its distance to the closest food while trying to keep its distance from the closest enemy if that distance is smaller then four squares. That way it will try to capture food while being weary of close-by enemies. We also try to minimise the distance to scared enemy ghosts.

3.2 Forward pass for uncertainty reduction

Given the structure of the problem, we have to deal with the uncertainty of the observations of the enemy agents. In our solution, we use a probabilistic model based on a first order Markov assumption of conditional independence. This means, that we treat each observation of as a fuzzy reading of a hidden state, which is only based on the previous hidden state. In order to estimate the real values of our hidden states, we use what is known as the Forward Pass algorithm or Forward Filtering, as it is sometimes called. By extension this also makes our solution an HMM based solution.

A probabilistic filtering model in a discrete state space often contains 3 things:

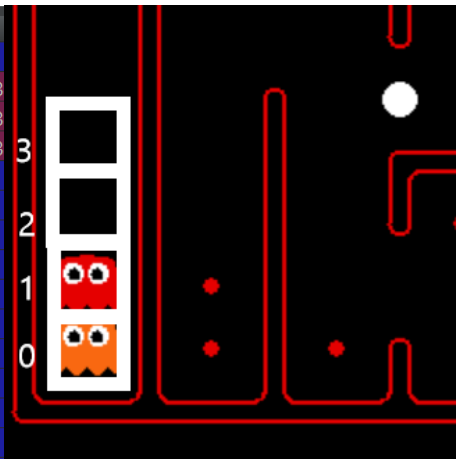
- An initial probability estimation vector for states.
- A transition matrix, that determines the assumed probabilities for transitions between states.
- An observation emission probability matrix, giving the probabilities for each hidden state given each observation.

In our solution, we build the transition matrix based on the set of legal states. In this game world, a legal state is any position that is accessible by a pac-man. We have no need to represent wall positions in the transition matrix, as we would never access them or never move onto them. Essentially this means that we can reduce the transition matrix by the amount of positions that

are walls in the game, instead of calculating over the whole game grid. We know that at every time-step a pac-man can only move to an adjacent square. Therefore, creating a transition matrix becomes a problem of mapping each state as being able to transition to states that are of manhattan distance 1 from it, or the given state itself (standing still is allowed). Values in the matrix are generated to be row stochastic or in other words "normalized by row", with equal probability for each such state transition. A game world and transition matrix representation example is given in Figure 1.

	0	1	2
0	0.5	0.5	0.0
1	0.3333333333333333	0.3333333333333333	0.3333333333333333
2	0.0	0.3333333333333333	0.3333333333333333
3	0.0	0.0	0.3333333333333333
4	0.0	0.0	0.0
5	0.0	0.0	0.0
6	0.0	0.0	0.0
7	0.0	0.0	0.0
8	0.0	0.0	0.0
9	0.0	0.0	0.0
10	0.0	0.0	0.0
11	0.0	0.0	0.0
12	0.0	0.0	0.0
13	0.0	0.0	0.0

(a) Matrix representation



(b) Game grid

Figure 1: Game world to transition matrix

Pre-computing an emissions matrix would not be as viable or even necessary. In our problem the observation is given as a manhattan distance between the agents, taken from an uniform distribution of $\pm$ 6 of the real value. This gives us a hollow diamond shape of width 13 in the game world, with each legal position in that shape being equally likely to have generated this observation. We could then pre-compute a matrix which gives the probability estimation for all states given these observation. However, we quickly see that we need to generate multiple matrices for each possible position that we as an agent make the observation from as we also move around ourselves and the distribution changes as we move around.

Instead we use the knowledge of how the generate this distribution and instead of generating a pre-computed matrix, create the emission distribution in real-time. This also allows us to account for a few heuristics which can change during game-play. To summarize, we can use the following pieces of knowledge about the world for determining the emission distribution.

- The probability of being within 5 spaces of our agent is 0 otherwise we know the position exactly.
- The probability of an enemy being on a space with food from our team, is 0 otherwise it would be eaten.
- The probability of an enemy being on a capsule on our side is 0, otherwise it would be eaten.
- All the other spaces within manhattan distance ± 6 of the agent have equal probabilities of containing the agent.
- We can create a "certain" distribution with a state having probability 1.0 when food or capsules get eaten.
- We can create a "certain" distribution, when we see an enemy.
- When an enemy gets eaten, we can reset the belief to the original belief distribution, with probability 1 for the starting state.

As we know the starting position of the enemy team, we can easily create an initial distribution with 100% accuracy. This will be our initial belief state. As we have the transition matrix and the emission distribution for the observation at each time-step, we can update the current belief state using the forward algorithm. In our model we use the state with the highest estimated probability from our current belief distribution as our guess of the enemy position.

This algorithm has a very convenient quality for us. We can simply calculate the next state based on the previously calculated state, which reduces computational complexity considerably. For each observation, we simply need to add it on top of previous calculations, without iterating through all of the observations up to a point. In addition, as we are keeping a belief state of the enemy agent's position, we essentially get two observations for each move of the enemy agent. This allows further improvement of the prediction accuracy, especially if our own agents are spread out in the game world. When we have less of an overlap in our observation probabilities, we can narrow down the position of the agent faster.

	Wins	Total score
Group 2	22	364
Group 14	22	311
Group 11	17	234
Group 3	16	161
Group 13	10	173
Group 4	5	35
Group 1	0	0

Table 1: Finals

	Wins	Total score
Group 11	7	64
Group 14	7	44
Group 13	6	38
Group 4	5	19
Group 2	5	17
Group 3	3	8

Table 2: Semi-finals

4 Experimental results

4.1 Experimental setup

The tournament, and thereby the test of our agent, was run in two sessions - a semi-final and a final. The sessions were run two days apart to let teams adjust their strategies after seeing how their opponents play.

4.2 Results

As we can see on table 2, our performance on the semi-finals was poor. This was due to problems and bugs we had on the night before the competition when we merged the uncertainty reduction code with the expectimax and heuristics part.

However, for the finals we managed to fix our code and our performance was much better, finishing at fourth place, only one win behind Group 11, as seen on table 1. Using search heuristics to evaluate the board state proved to be a successful idea: from the three groups that finished above us, two used approaches similar to ours. Group 2 also used minimax and Group 11 used an exclusively heuristic algorithm evaluation, similar to a minimax with depth of one. Group 14 differed from us and decided to go for a decision tree implementation.

5 Summary

We were able to successfully implement a search algorithm with a heuristic evaluation function of the board state. Coupled with a HMM based forward pass filtering algorithm we ended up with an agent that can reliably predict the position of the enemy agents while evaluating its next move. Ultimately, we determined that the game space was too large for in depth searching, which limited our long term behavior, narrowing our ability to plan moves into the future. This made executing some more complex behaviors impossible. Non-the less we were able to compete, not falling far behind the teams that beat us.

5.1 Further improvements

The teams that did best focused more on the heuristics and behavior functions of their agents, having more elaborate behaviors and accounting for more scenarios. This suggests that our approach could benefit from additional heuristics such as detecting dead-ends etc. As we saw, that our depth of search was limited, we might try to implement some type of pruning in order to reduce the size of the tree, ignoring some less relevant states. This might help us plan our movements farther ahead, allowing for some more complex maneuvers. Our agent might also benefit from being able to switch between behaviors, or more generic heuristics which allows to evaluate both offensive and defensive behavior and find the best balance given the situation. We might also benefit from further collaboration in our behavior heuristics. Right now, the main type of collaboration between the agents is in determining the enemy positions.

As the filtering method works rather reliably, there are not that many things that can improve it in it's current state. It might be valuable to try to train/learn a more specific transition model with something like the Baum-Welch algorithm. We assume currently that there is no underlying process to the behavior of the agents when doing our estimations, but this is clearly not the case. At any rate, this can backfire as the agent's behavior changes depending on different factors. It could be possible to try learning different models for agents that are carrying different amounts of food for example. It might be interesting to experiment with such things, but our priorities would be to improve the agent, rather than the inference.

References

[1] Hamzah Ahmad and Toru Namerikawa. Extended kalman filter-based mobile robot localization with intermittent measurements. *Systems Science & Control Engineering*, 1(1):113–126, 2013.

[2] Mauricio Ayala-Rincón, Eduardo Bonelli, and Ian Mackie, editors. *Proceedings 9th International Workshop on Developments in Computational Models, DCM 2013, Buenos Aires, Argentina, 26 August 2013*, volume 144 of *EPTCS*, 2014.

[3] Murray Campbell and T. Anthony Marsland. A comparison of minimax tree search algorithms. 20:347–367, 07 1983.

[4] Ronald Rivest. Game tree searching by min/max approximation. 34, 04 1995.

[5] Glen Robertson and Ian Watson. A review of real-time strategy game ai. 35:75–104, 12 2014.

[6] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach*. Number ISBN 978-0-13-207148-2. Pearson Education, 3rd edition, 2010.

[7] Claude E Shannon. Xxii. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314):256–275, 1950.

[8] UC Berkley. Contest: Pacman Capturethe Flag. `http://ai.berkeley.edu/contest.html`, 2015.