Games & Theory of Mind in Pac-Man (April 2020)

Wei Xin, Vishwarajsinh Sodha, Sajjan Kumar, Amit Sharma

Abstract—**This Paper introduces the topic of Games & Theory of Mind in Pac-Man domain. As for games and theory of mind, it is about how we represent the intentions and goals of others to maximize our action(s) to reach the final goal. There are several steps that we need to think of. First, the goal set. We definitely need to know what the final goal set is. The goal can be single or multiple but, in the Pac-Man domain, the goal is only one which is the Pac-Man agent should eat all the dots without being eaten by one or more ghost(s). Second, the assumptions (heuristics). The agent should make decisions on its own based on various of factors – Current location, Available actions, Opponent's location(s), Dots' location(s), and how many dots left, etc. To reason the above factors, in other words, the agent should basically evaluate its current state and the opponent's current level state. Now, the concept of "Level" is mentioned.[1] The agent will evaluate the opponent's level in the level beyond this level, the opponent will evaluate the agent's level in the level beyond this level, and so on ad infinitum. Therefore, in order to finish the assumption, the level should be separated into two parts. These parts are called Min and Max. In the Pac-Man domain, the Pac-Man is determined to be at Max level while the opponent(s) is(are) determined to be at the Min level. However, there is no reason to say that Max level is always optimal while Min level is not. Besides, the agent may or may not choose what the expected action, in other words, it will choose with probability. What this paper is trying to illustrate is the agent's Max level can reach to an optimization (i.e., perfect model) and this level is one level beyond the opponent's level. The verification is displayed by Minimax agent. In the future implementation, reinforcement learning will be added just to train agents and the opponents to give a better look on the verification. Finally, we implemented analyzer agent which tracks and displays performance of Pacman through the course of multiple games at different reasoning levels and gives final analysis in console.**

I. Introduction

THIS paper introduces the idea of why Pac-Man performs better than its opponent and how to divide this problem into multiple test cases. During the proof, we use Minimax and Expectimax agent to test different levels of performances and reinforcement learning method to better finish the verification. Pac-Man domain is a classic Artificial Intelligent problem, so it is necessary to use different technical strategies to display the results. In the original project, we use regular code to present Minimax. On top of that we choose Minimax action using 0.8 probability. Also, we created another user-friendly way to prompt the result and analysis that is based on the preferences. Whichever way the user chooses, the verification will always hold that the optimal reasoning level of Pac-Man is always one level beyond the opponent.

II. Technical Approach

Games & Theory of Mind in Pac-Man domain is set up as comparing different levels of agents and ghosts' performances. In other words, under this topic, the optimal reasoning level should be found just to verify that the agent is thinking better than ghost. To obtain the performances, we built upon Project 2 in the course which focuses on different type of agents. Minimax and Expectimax agents have similar approaches but Expectimax has already had a way to make the agents choose with a certain probability but Minimax agent has not. Therefore, we use Minimax agent and form a certain probability on the result actions. Furthermore, the agents share the same evaluation function while doing the adversarial thinking against the ghosts. However, our program made a few changes to compare this with the original agents' performance which, in other words, regular Minimax without probability and agents will do two different evaluation functions. We have implemented analyzer agent as well, which keeps track of multiple games that our agent plays, analyzes it's wins and time it takes to win based on which analyzer shows results whether our Pacman is overly reasoning, optimally reasoning or underthinking.

Furthermore, we deployed Reinforcement Learning as an additional approach to verify that the optimal reasoning level is one level beyond the opponent's level.

A. Minimax Agent with Probability

Here we are using two ways to see our agent behavior using minimax algorithm. One is Minimax Algorithm without probability using different layout such as minimax classic, trapped classic and the other is Minimax Algorithm with probability of 0.8 for decided action and 0.2 probability for random actions. We have tested on several other layouts as well.

This article is finished on May 1st, 2020. The work was not supported, only the four authors.

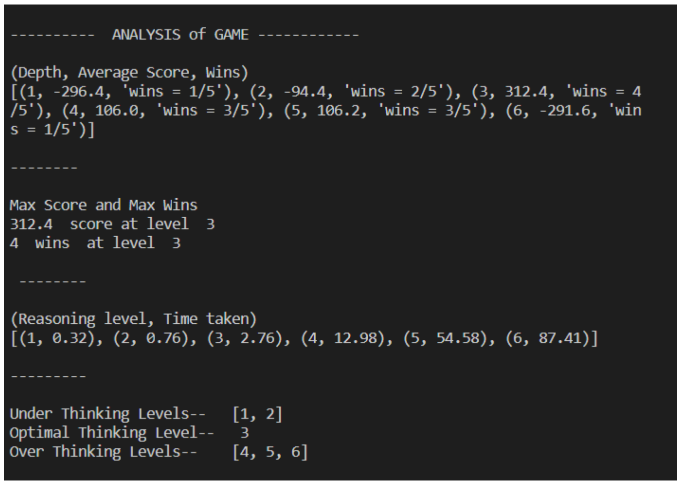
Wei Xin is with the Arizona State University, Tempe, Arizona 85281 (e-mail: wxin4@asu.edu).

Amit Sharma is with the Arizona State University, Tempe, Arizona 85281 (e-mail: apsharm3@asu.edu).

Vishwarajsinh Sodha is with the Arizona State University, Tempe, Arizona 85281 (e-mail: vmsodha@asu.edu).

Sajjan Kumar is with the Arizona State University, Tempe, Arizona 85281 (e-mail: skuma175@asu.edu).

For trapped Classic layout we see that Minimax agent is getting killed because it sees ghosts from both sides. With lower reasoning level the agent takes longer route but sometimes avoids dying as other ghost is far-off and might not be optimal. With higher depth the agent can see the far-off ghost and will think that it's a trap and try to get killed with minimum penalties. We are giving options to the user in order to check the behavior of the agent. Once user chooses max-depth level, our algorithm plays a constant number of games for each level starting from level 1 to the chosen level. There's one point where our agent achieves optimal win rate without taking too long. This is the level which we are using to separate the overthinking and underthinking. We have observed that the best optimal level is 3 or 4. If we keep increasing our level, then the agent will take more time to decide its actions and win rate stays almost constant. In this case we can say that our agent overthinks. For level less than optimal level Pacman takes its action with limited reasoning therefore it is underthinking and underperforming. Below are the results for 5 games each for different levels for minimax Classic layout. A list with maximum depth, time taken by agent and number of games our agent won out of 5 games can be seen in Figure 0.



```
----------  ANALYSIS of GAME ------------

(Depth, Average Score, Wins)
[(1, -296.4, 'wins = 1/5'), (2, -94.4, 'wins = 2/5'), (3, 312.4, 'wins = 4
/5'), (4, 106.0, 'wins = 3/5'), (5, 106.2, 'wins = 3/5'), (6, -291.6, 'win
s = 1/5')]

--------

Max Score and Max Wins
312.4  score at level  3
4  wins  at level  3

 --------

(Reasoning level, Time taken)
[(1, 0.32), (2, 0.76), (3, 2.76), (4, 12.98), (5, 54.58), (6, 87.41)]

---------

Under Thinking Levels--   [1, 2]
Optimal Thinking Level--   3
Over Thinking Levels--    [4, 5, 6]
```

Figure 0. Analysis Result for Minimax Agent

B. *Expectimax Agent with Probability*

For expectimax we use the average of the adversarial nodes in the game tree to maximize the game score for pacman. The main thing here is that the agent will always try to maximize the score by achieving the goal state. Here Pacman agent knows that the ghost can take random actions since ghosts are not always taking the actions to eat the agent. Therefore our agent will play optimistically and try to go where max average rewards are available. We are using the probability in this algorithm where the agent will move in the decided direction with 0.8 probability and take random action with 0.2 probability. The depth level plays an important role in our algorithm. With given depth our agent will look up till that level and decide the actions from a set of actions. The agent chooses the action which will increase the chance to maximize the total score. We are taking consideration of overthinking and under thinking in our algorithm.

C. *Reinforcement Learning*

After implementing Minimax Agent and observe the performance, we still cannot see a perfect model for agents against ghosts. To better perform this, Reinforcement Learning should be used because the agents may perform much better after careful training in different levels. For example, when we train Level-1 agent and let it play against Level-0 ghost, the winning rate will be higher than the Minimax Algorithm, even with the probability added into it.

As mentioned, Minimax algorithm takes agent as Max and the ghost takes Min nodes so even with probability layer, we still consider that agent and ghost are playing adversarial games. Therefore, we need to use the Cognitive Hierarchy Theory (CHT) in order to achieve the goal that evaluates agents and ghosts separately. In CHT, agents are at Level 1/3/5 while ghosts are at Level 0/2/4, and so on. Reinforcement Learning makes the Pac-Man game clearly separated and easy to train different parts just to observe the result in a better way.

III. Results

We used a lot of different layouts and levels to compare the performance of the agents. In order to show clear conclusions, we will not only display the command line results but also some graphs. As the previous knowledge given, we know that the Minimax agent will perform poorly without probability while Expectimax agent will perform a little bit better, and the trained agent will definitely present almost a “Perfect Model”.

A. *Minimax Agent Results*

Since the result of all tests are for scores that agents do better than ghosts, we created ANALYZER agent that tracks Pacman at all games at each level and finds out best performance among all the reasoning levels, tracks if Pacman underthinks or overthinks based on the time it takes to play set of games. Finally in Console it shows which set of Reasoning levels take exponentially more time and win rate remains constant considering it as overthinking. We used MinimaxClassic layout as it is smaller and doesn't take much time with higher levels of reasoning as well. We added time elapse of each level of game and the average score of each level and plotted them as a pair of graphs. Figure 1 shows that how the agent performs against the opponent. Our principle is that once the agent win rate becomes good, we consider it as optimal thinking, as reasoning more than that also takes exponentially more time and win rate also doesn't drastically improve. However, since ghost is randomly performed, the winning rate will never be the same. We will show a range that the agent reaches its optimal level and whenever the agent is performing worse than that level and the time elapse is high-rising, then it is going to be overthinking, and the underthinking will be the other way around. Besides, since we already mentioned that the agent is in 0.8 probability to choose the correct direction and 0.2 probability to choose randomly, the agent is somewhat performing randomly as well. Therefore, the performance, in other words, the graph, may display in multiple "peaks". Here we only consider that whether the agent's winning rate is greater than the ghost's wining rate so that we can come up with a conclusion that agent is thinking better than its adversary.

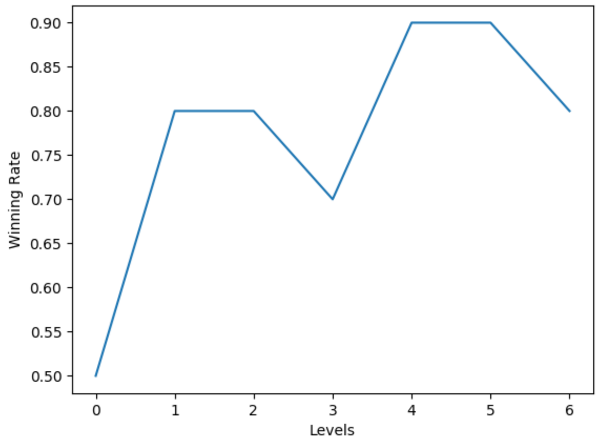


Figure 1a. Minimax Winning Rate of Agent

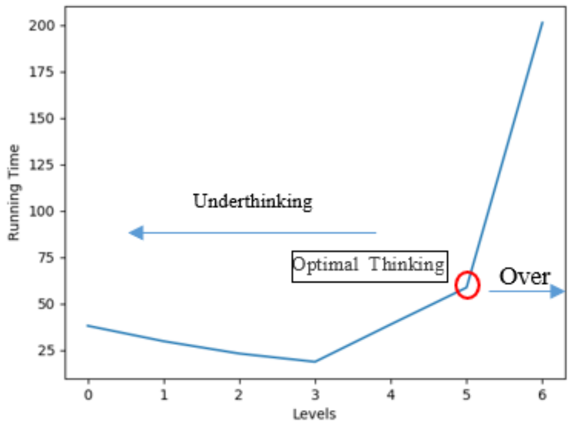


Figure 1b. Minimax Time Elapsed

As Figure 1a shows, the winning rate reaches to a peak level when depth is at 4 and 5 and that means the agent is thinking the best while below 4 is underthinking and over 5 is overthinking. Besides, overthinking may cause running time to rise super quick as Figure 1b shows. Therefore, in order to let user test faster, we include a user prompt and the depth level should be within 7.

B. *Expectimax Agent Results*

The reason why we include Expectimax agent is that the way we implement Minimax agent with probability is more or less like Expectimax agent. Our Minimax agent's probability is shown above, and we want to compare against Expectimax result. Figure 2 shows that the Expectimax performs better since the winning rate is over 0.7 above all. The only same part is that when the agent thinks deep down to the maximum level, then the time complexity goes up because both the agent and ghost are thinking and calculate too much.

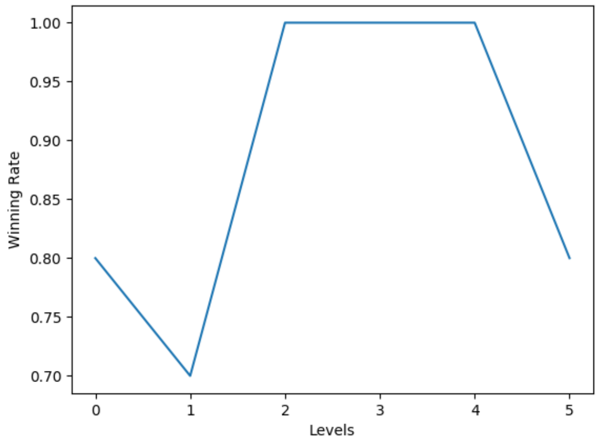


Figure 2a. Expectimax Winning Rate of Agent

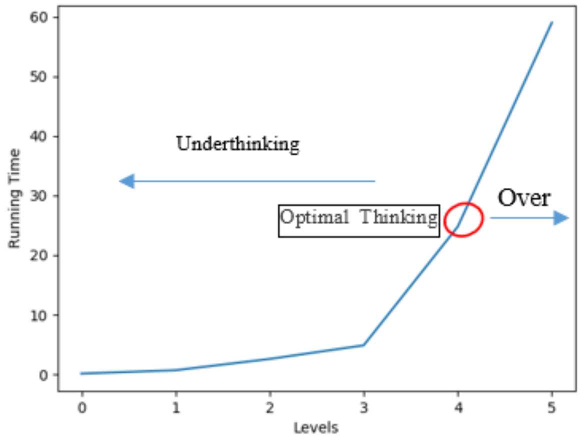


Figure 2b. Expectimax Time Elapsed

As we can conclude for now is that even if Minimax agent is not performing as well as Expectimax, we can still say that the optimal reasoning level is one level beyond the opponent. As Figure 1a and Figure 2a show, the opponent's optimal reasoning level is at level 3 and level 1 respectively and Pacman's optimal level is at level 5 and level 4 respectively.

Furthermore, we reevaluate the result with different evaluation functions by changing the parameter in the __init__ function of MultiAgentSearchAgent class in multiagent.py file. There are two evaluation functions which are regular one and the better one. As for the graphs show, these are all based on the better evaluation functions, but this can be changed by modifying the "evalFn" to evaluation function only. There is very slightly different because the only thing that changed is the scoring. Therefore, we do not show another set of results. The results presented here are for specific layouts only. Underthinking, Optimal and overthinking reasoning levels may depend on game state such as number of ghosts, layout, Pacman Agent, etc. So while running you might get different analysis based on what command you run.

C. *Reinforcement Learning Results*

As we all know, the agent does not perform perfectly so the proof for now is not done yet. We introduced reinforcement learning method just to train the agent (and the ghost for the future) so that the agent will perform as good as it can get. We use a dotted graph to show level 1 agent that has been trained. The Q-Learning function trains the agent and it learns by trial and error from interactions with the environment through its update of the state, action and reward. The Q-values are computed using the Bellman equations to maximize rewards which is shown in Figure 3. [2]

$$Q(s_t, a_t) = \sum_{i=1}^{n} f_i(s_t, a_t) w_i$$

$$w_i \leftarrow w_i + \alpha f_i(s_t, a_t) d$$

$$d = \left(r_t + \gamma \max_a Q(s_{t+1}, a_t) \right) - Q(s_t, a_t)$$

Figure 3. Bellman Equations

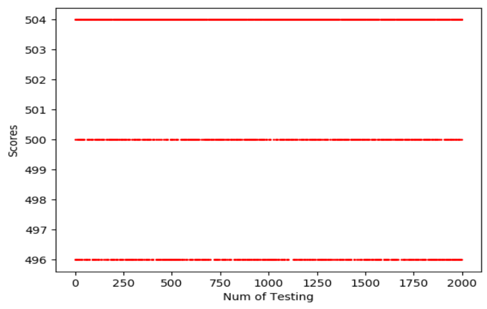
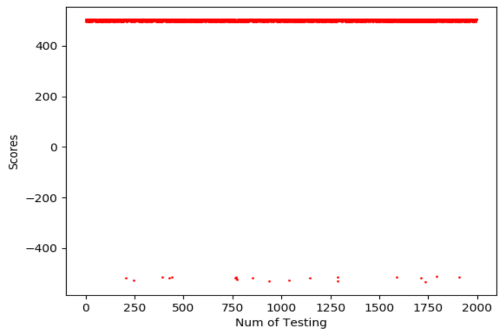


Figure 4. Pac-Man Level 1

As Figure 4 shows, after 4000 times of training, the Pac-Man at Level 1 can play extremely well against Level 0 ghost. This can be compared with the Minimax agent with at most 90% of the winning rate and Expectimax agent with 100% winning rate but only at optimal level. This chart shows 2000 times of testing and winning rate is 100% which can be known as "Perfect Model".

We figured out another way to show how agent performs when the reinforcement learning is for training the opponents. The ghost can be trained at Level 2 and play against Level 1 Pac-Man. However, there is a technical difficulty that the ghost agents are using random algorithm so they cannot be achieved with the current framework due to the sparse rewards for ghosts, lack of cooperation between the ghosts and insufficient features. Additionally, approximator function for Q-values is linear which limits its ability to include more detailed information about the game encapsulated in the features.[3] It is still enough to say that even ghost is optimal, the agent can still win almost all the games. We plotted a graph to show that the agent is more organized.



We used 2000 trainings for ghost and the other 2000 for testing. As we can see, only around 20 are losses so the winning rate is 99% which is still optimal against the Level 1 ghost. This can be compared to Minimax Agent and Expectimax Agent and the proof of the optimal reasoning level is one level beyond the opponent.

IV. Conclusion

Increasing the level of depth at which the agent thinks increases the optimality and amount of successful operations it can do. However it was observed that if the depth is increased to more than a certain level then it becomes excessive and time consuming which can be considered as the classic example of overthinking.

Agent operates at 3 levels of optimality, when there is a reduced amount of win ratio it becomes the case of underthinking. To sum, it can be assumed that adding few levels of up to certain limit ensures satisfactory functioning of agent without going to adverse levels.

V. Team Effectiveness

We in a team 4 collaborated equally and effectively towards this project. We used GitHub to collaborate remotely and do code reviews. Also, our team followed some of the aspects of Agile development model where we used to discuss the roadmap of the project in daily scrums and ensured that nobody in the team is blocked. As each of our team member had knowledge of several different domains, we could combine all of it to constructively build the code for this project.

Towards the end our team collectively portrayed the individual contributions in this project report. Overall it was a good learning experience and we believe we achieved good amount of exposure to various search algorithms and how it can be tweaked to achieve better suited performance and result.

Contribution

Expectimax and Minimax Skeleton	Amit, Sajjan
Probability algorithm implementation	Wei, Vishwaraj
Evaluation function	Wei, Vishwaraj, Sajjan, Amit
Analysis and reasoning	Wei, Vishwaraj
Testing and Review	Amit, Sajjan, Wei, Vishwaraj

References

[1] L. Bom, R. Henken, and M. Wiering, "Reinforcement learning to train Ms. Pac-Man using higher-order action-relative inputs," in IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), Apr. 2013, pp. 156–163.

[2] Yoshida, W., Dolan, R., &amp; Friston, K. (n.d.). Game Theory of Mind. Retrieved May 01, 2020, from https://journals.plos.org/ploscompbiol/article?id=10.1371%2Fjournal.pcbi.1000254

[3] K. Ranjan, A. Christensen, and B. Ramos, "Recurrent deep Q-learning for Pac-man," 2016.