Abstract

Mastery over games has been a key cornerstone of AI development that pave the way for solutions to real-world problems. Whether be it classical games like chess or video games like Pac-man, the agents need to be supplemented with a combination of fundamental AI functionalities to compete and be the best. Here, Pac-man: Capture the Flag originally developed by University of California Berkeley requires cooperative multi-agents to compete in an adversarial and partial-information environment. A heuristic-based approach is taken to define two agent behaviours coupled with Bayes' inference to acquire complete information. The solution is pitted against 5 other teams in two rounds of competition and interesting insights on static role assignment and more are detailed based on these results.

1 Introduction

Deploying AI agents that can beat humans and master games has been a forefront of artificial intelligence development. Games make for a good playground to evaluate agents as even the simplest of them allow for deep strategies, prediction and adaptations based on how an opponent play. Major breakthroughs in AI are recorded as moments when AI agents managed to out-play reigning game experts at classical games like Go or Chess, and more recently at video games such as Dota 2 or StarCraft. The more recent exploits are more advanced as they are multi-agent systems in cooperative and non-cooperative environments without perfect information, meaning the user does not have full knowledge of the state of the game.

The classical Pac-man game has seen longstanding use for testing applicable common AI methods. However, standard Pac-Man is a perfect information game with a single agent and no room for competition. To make a more useful environment, UC Berkeley developed a Pac-Man variant called Pac-Man Capture the Flag. It has various elements like agent acting as ghost or pac-man based on position in the map, with partial information of opponent locations and with the goal to capture opponent food while defending your own food. This makes it a multi-agent cooperative environment with two teams pitted against each other. These elements vastly increase the number of possible methods one can apply to make viable AI agents and win the game. A detailed set of rules governing the game can be found at UC Berkeley's website for its AI course [1].

1.1 Contribution

In this paper, a functional implementation for the UC Berkeley Pac-Man Capture the Flag game is presented and its performance when pitted against various different strategies is discussed. The implementation is primarily tuned for a two-agent team. A division of roles is considered with different implementations for an aggressive and defensive agent. This approach allows to focus on developing good strategies for the different scenarios in the game, yet also helps explicate some of the downsides with rigid agent roles.

1.2 Outline

This report outlines the development of the undertaken project. As many parts of the implementation are grounded in existing research and methods, Section 2 presents the work on which the project is built. The developed

solution, based on the aforementioned works as well as novel solutions, is presented in Section 3. As multiple teams worked on developing solutions to this problem, multiple approaches have been tried and are able to be compared. In Section 4, the qualitative results are presented relative the other works. Analysis is also provided on the advantages and disadvantages of the different approaches. Finally, Section 5 provides a summary as well as some final conclusions.

2 Related work

A goal of the project was to ground the development in existing research. Given the nature of AI in game theory, there is an abundance of existing literature, of which a small portion is detailed here in relevance to the solution attempted.

The Minimax algorithm bases itself in the similarly named Minimax theorem, first proven by Neumann himself.[2] When playing a competitive game, one wishes to pick the move that maximises our score or chance of winning. In contrast, our opponent will seek to minimise our score. The idea of the Minimax algorithm is that by considering how our opponent can react to our moves, we maximise the minimum which the opponent can impose on us. In a game setting this is often visualised as a tree graph with branches for each possible move at a given point in the game. An issue with Minimax is that the optimal move is only able to be found by searching the full tree down to the end of the game, something which for all but the simplest games is an unreasonably large search. To combat this, the idea of a heuristic was introduced[3]. The Minimax search is only perpetuated to a certain depth, where the value of the state is given by the heuristic, which measures "goodness" of a particular state of the game. This enables reasonable run time, although the performance depends on how good the used heuristic is as well as the depth of search used.

Since a deeper search yields better results for Minimax search, we have employed the use of alpha-beta pruning[4]. This is a method that prunes unnecessary branches of the search tree, terminating search down branches which will not affect the final outcome. By pruning the search tree, one can accomplish deeper searches in the same amount of computational time, thus getting a better result.

A method that was also considered but ultimately scrapped was using

the expanded Expectiminimax algorithm[5]. Expectiminimax builds on the standard Minimax implementation with the inclusion of allowing for chance events to be considered, such as dice rolls. The idea behind the inclusion was to cover for us not knowing the opposing strategy, thus adding some randomness to our predictions. In the end, the idea was not used, and instead Minimax was implemented with the assumption of the enemy strategy being the worst possible for us.

Often with partially observable environments, it is required to infer more information by keeping track of the current and previous state information. An array of methods for obtaining this information is discussed in the Uncertain Knowledge and Reasoning sections of Artificial Intelligence: A Modern Approach [6]. The authors mainly discuss three inference algorithms for temporal models – Hidden Markov Models (HMMs), Kalman filters, and Dynamic Bayesian Networks (DBNs). HMM frameworks deploy a probabilistic model with a single discrete random variable while Kalman filters handle multiple continuous variables. DBNs can be represented as first-order Markov processes with a single state variable and a single observation variable and are often thought of as sparse versions of HMMs[7]. These algorithms are done with specific steps of filtering, prediction, smoothing, computing the most likely explanation, and learning.

Last but not the least, course work detailing the fundamental concepts of state-space search, multi-agent search, probabilistic inference, and reinforcement learning from the AI course [8] at University of California, Berkeley also provided insight on basic set-up for the Pac-Man game.

3 Method

The Pac-Man CTF game consists of a symmetric playing field, with each side corresponding to a team. To gather points, agents must pick up food on the enemy half of the field and return to their own field side. To counter this, agents can kill enemies on their own terrain. The approach to this problem is achieved by assigning two different agent roles with different behaviours - the attacker and the defender. Both roles inherit from a base class with useful common functions. Default available functions are taken advantage of to build the functionality of these two roles. One of the main functions created is choosing the best action to move towards or away from a given target. (The move which increases/decreases the distance most) This function is used frequently to determine the movement of the agents, as the used

approach at each frame determines a point to pursue or evade, and then uses this function to translate it into an action. This point could for example be an enemy, food, capsule, or intercept point.

Section 3.1 explicates the details of the attacker role that focuses on collecting food from the enemy field while avoiding enemy agents. The defender behaviour is described in Section 3.2 and focuses on preventing and pursuing enemies that try to cross the field to steal food.

3.1 Attacker

The attacker primarily focuses on collecting food, but its behaviour branches based on the presence of nearby enemies with each situation using heuristics and state-to-state decisions. The entire flow of decision-making for the attacker can be seen in Figure 1. In the following subsections, the main essence of the mentioned differences is described in detail.

3.1.1 Gathering food

When no enemies are deemed close enough to the attacker, it focuses on gathering food and capsules (larger foods which make the agent impervious to enemies for a duration of time). If the attacker is still on home field, the general heuristic of the agent is to travel to the closest enemy food or capsule. Since the capsules are more powerful, they are also prioritised even if they are a bit more distant. Once the agent starts picking up food, there are multiple heuristics for whether the agent should return. A hard threshold *numInMouth* is set at which point the agent will return home to drop off the food. A sliding threshold is also in place for how far away food can be and still be worth grabbing before turning back. This means that the agent will grab clusters of food near each other but return earlier than the hard threshold if the nearest food is far away. Additionally, the agent will pick food up on the way back if it is close to its path. Once the attacker decides to turn home it determines the closest point on home field and takes the optimal route, given there is no enemy interference.

3.1.2 Escaping enemies

The attacker uses two separate ways to detect enemy agents, due to the setup of the game. When enemies are within a certain range of the agent, their exact position can be determined. When further away, only an approximate position with a large noise factor can be acquired. When agents are outside

of the visible range, Bayes' inference is used to estimate their position. (More details in Section 3.2.1) If an enemy is too close, a vulnerable attacker agent will try to escape. (If the attacker has eaten a capsule and is impervious, it will ignore enemy agents unless they get very close, in which case the attacker will try to kill them.) To avoid or escape enemies, the Minimax algorithm is employed which estimates the best course of action by simulating many moves ahead. The algorithm samples a branching tree alternating between the friendly and enemy agent moves. The move is chosen which gives the best outcome when maximising a given heuristic, and the enemy tries to minimise the heuristic. The heuristic used is given in Equation 1.

$$score = dist_{enemy} + \frac{1}{dist_{safe} + 1} \tag{1}$$

This heuristic primarily increases with the distance between agents ($dist_{enemy}$), and secondarily by shrinking the distance to the closest safe location ($dist_{safe}$). Safe locations are defined as both the boundary of home field, but also the locations of enemy capsules. By using a simple to calculate heuristic as well as alpha-beta pruning, sampling can be done at depth 10 without computation taking more than a second, despite the branching factor being 5 at worst. On top of alpha-beta pruning, two scenarios are checked at every node of the game tree. Firstly, any state where the friendly and enemy agents occupy the same location ceases any deeper search down the branch and returns a large negative score. Additionally, any state that puts the friendly agent on a safe spot returns a large positive score. This overrides the standard heuristic and ensures the agent avoids all choices that end up with getting killed, while always prioritising moves that lead to safety, regardless of the heuristic. For example, this encourages the agent to get closer to an enemy agent, if it in doing so can reach a safe point before being caught. This also prunes the search tree significantly by terminating further exploration beyond such 'terminal' states.

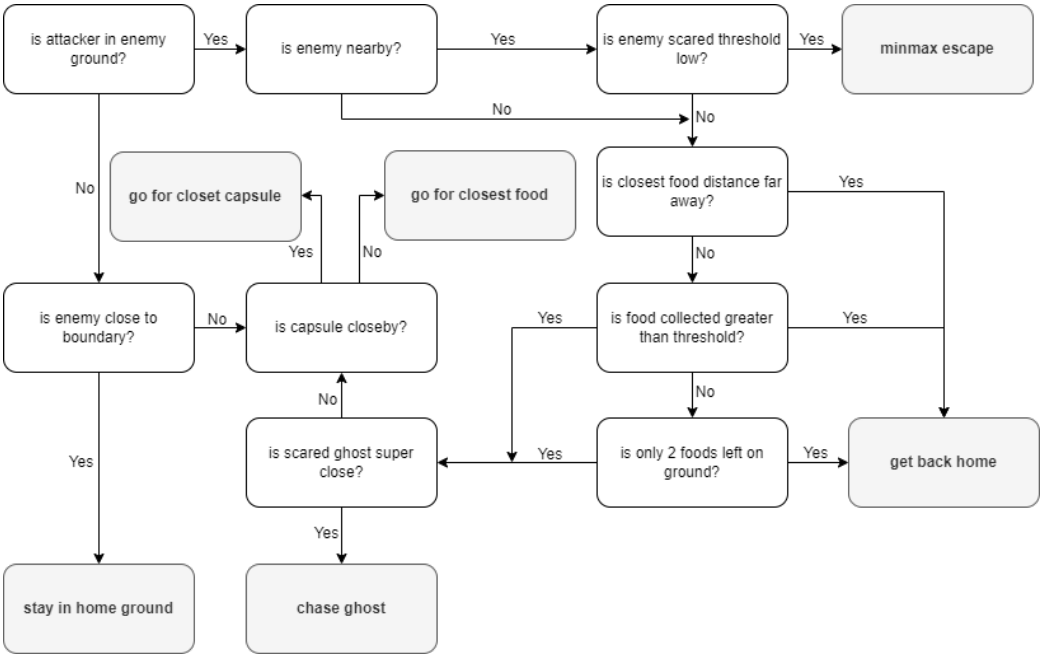

Figure 1: Algorithm for Attacker - Acts as Pac-man

3.2 Defender

In comparison to the attacker, the defender behaviour is more straightforward. It attempts to intercept incoming enemies at the boundary and chase ones that crossed over to the home field. Detection of enemies not in visible range is very crucial here, for which Bayes' Inference is used with other chasing methods which will be detailed in the following subsections. The entire algorithm for the defender is summarised in Figure 2.

3.2.1 Estimating enemy location

Every agent (both defender and attacker) updates a global variable that maintains the enemy agent states and enemy positions when available. It is updated with true or actual position if known, and otherwise with a time elapsed belief distribution. This belief distribution is updated based on previous known locations or beliefs. The time elapsed belief uses the enemy base location and disappearing food location as criteria for updating distribution. Once these distributions are available, approximate positions are estimated based on the high probability of observation. These are then used in place of unknown position information.

3.2.2 Chasing enemies

Chasing enemies in close vicinity is quite easy, as the enemy position is known, but when there is no complete information a mixture of Bayes' inference and case-specific heuristic decisions is used. There are two prominent times when anticipating the positions of the enemy is needed. Firstly, when the enemy is still approaching home field and other is when the enemy is already on home field. With the estimated enemy position and using the path from the enemy base and to the closest food as metrics, a boundary entry point into the home field is predicted. The defender is stationed at this boundary point to await and intercept the enemy before it starts collecting food. The second case is when it is known that an enemy is in the home field but high accuracy estimation of its position is unavailable. In this case, the disappearance of food is used as a target. The defender moves to the area where food is disappearing to spot the enemy and capture it. Under circumstances where the defender is scared due to the enemy eating a capsule, the agent follows the enemy around at a slight distance. This is to ensure it is possible to easily chase once the affliction wears off and to not lose sight of the enemy.

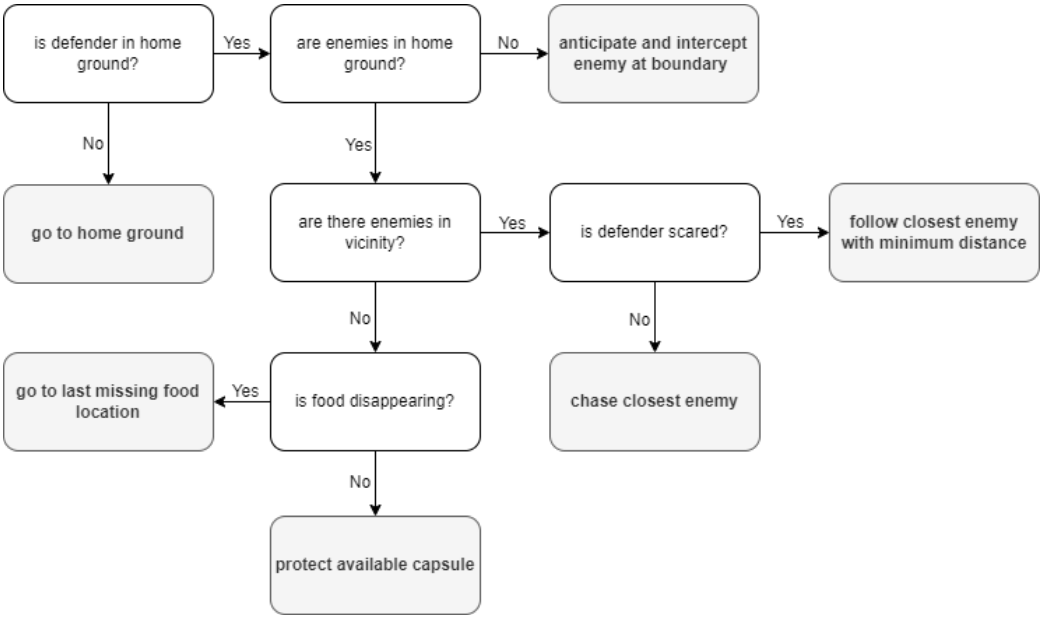

Figure 2: Algorithm for Defender - Acts as Ghost

4 Experimental results

The Pac-Man contest has 5 teams competing against each other and two rounds were held to compare their performances. For each competition, 2 random maps were generated. For each of the maps, every team simulated 3 games against each opponent, for a total of 6 games per map and match-up. The final score of each team was calculated by $score = n_{wins} + 0.5 \times n_{draws}$. Section 4.1 addresses some of the parameters present in our implementation which majorly affected the behaviour of the agents. Section 4.2 and 4.3 detail the results of each competition along with comparative analysis on our team's performance.

4.1 Experimental setup

A key parameter that contributed to the winning performance against certain opponents was the attacker's *numInMouth* which was limited to a max of 7. Other parameters for the attacker were distance measures like *enemyNearby* as 7, *scaredEnemyNearby* as 4 and *capsuleRelativeDist* as 3. Other one was *scaredThreshold* of 5. Meanwhile, the defender's key parameter was only *minDist* of 3 to maintain when scared.

4.2 Pre-finals

Table 1 shows the results of the pre-final games. In this competition our team did comparatively well, placing 2^{nd} in terms of score, only a few points after the first place.

Table 1: Pre-final Pac-Man CTF Results

Group	Time	Rank
G1	41	1st
G3	**36**	**2nd**
G5	31	3rd
G19	13	4th
G6	0	5th

Our approach saw a lot of success in this competition due to the attacking agent performing very well. The high depth Mini-max algorithm allowed our agent to complete most games without ever getting caught by the opponents defending agents. At this stage, our defender did not yet utilise disappearing food or Bayes' inference to pinpoint faraway enemy attackers, so our defender

only chased enemies if they got sufficiently close. Despite this allowing enemies to slip into our terrain unnoticed, our superior attacker behaviour led to winning most of the time.

Compared to Group 1, the only group to outscore us, their implementation was similar to ours in terms of behaviours. Their attacker also chases the closest food, avoids enemy agents, and returns home when having grabbed a certain amount of food. Their defender functioned better than ours, tracking our attacking based on eaten food and strategically placing itself close to where there is a lot of food to prevent an attack in the first place. This meant we got stuck in a deadlock (Our attacker trying to reach a food blocked by their defender) while their attacker could still bypass our short-sighted defender. Group 5 also used Mini-max, but only looked 4 moves deep, and thus sometimes walked into dead ends where we could kill them. It is also worth noting that Group 6 scored so poorly due to an error in their implementation, as neither of their agents left the proximity of their spawning location.

4.3 Finals

Table 2 shows the results of the final games, with us performing worse than in the pre-final.

Table 2: Final Pac-Man CTF Results

Group	Time	Rank
G1	42.5	1st
G19	29	2nd
G6	17	3rd
G3	**15**	**4th**
G5	11	5th

In comparison to the prefinals, our performance worsened in the finals placing 4^{th} despite the implementation of Bayes' inference for far-sight vision. This is predominantly due to the last-minute interfacing of position estimation without proper testing. The defender's logic to intercept enemies based on Bayes suffered due to low accuracy in prediction and that it overrides the disappearing food logic completely. In addition to the enemy base location and disappearing food logic, updating distribution with possible routes could have helped increase accuracy in prediction.

During the competition with Group 1, the disadvantages of having rigid attacker and defender roles were clearly seen. Group 1's agents prioritised

attacking completely in the first half and then chose to defend with both agents when in the lead, while our attacker was not able to amass as many points with a limited *numInMouth* of 7. Playing too cautiously was highly disadvantageous in such scenarios. The other groups had more or less same behaviour as us with most games being close calls. But a major attribution to Group 19's wins was that it went for capsules immediately, no matter how far which meant it could gain those few extra points much easily before deadlocks happened. While our team did make an attempt at escaping the deadlocks by moving away to target new food positions, its effectiveness was still very map independent.

5 Summary and Conclusions

The project was aimed at addressing the Pac-Man: Capture the Flag gameplay as both pac-man and ghost. This was conducted to a reasonable degree with heuristic-based algorithms of specific agenda. Additionally, a probability-based position estimation logic was also incorporated using Bayes' Inference for complete information about the game. From the comparative analysis of the results, insights could be drawn regarding the favourability of the algorithms against opponents using similar or completely different strategies. Overall, better integration of position estimation could have changed the direction of our development. Otherwise, a key observation noted was that while rigid roles often resulted in deadlocks, dynamic role switching was a competitive advantage as seen by Group 1's results. Our algorithms while able to hold its ground against similar solutions could have benefited greatly by parameter tuning and escaping deadlocks.

References

[1] UC Berkeley. *Contest: Pacman Capture the Flag.* URL: `http://ai.berkeley.edu/contest.html`.

[2] J. v. Neumann. "Zur Theorie der Gesellschaftsspiele". In: *Math. Ann. 100* (1928), pp. 295–320.

[3] Claude E. Shannon. "Programming a Computer for Playing Chess". In: *Computer Chess Compendium*. Ed. by David Levy. New York, NY: Springer New York, 1988, pp. 2–13. ISBN: 978-1-4757-1968-0. DOI: `10.1007/978-1-4757-1968-0_1`. URL: `https://doi.org/10.1007/978-1-4757-1968-0_1`.

[4] D. J. Edwards and T. F. Hart. "The Alpha-Beta Heuristic". In: (1963).

[5] Stuart J. Russell and Peter Norvig. "Artificial Intelligence: A Modern Approach". In: Prentice Hall, 2009.

[6] SJ Russel and P Norvig. "Uncertain knowledge and reasoning". In: *Artificial intelligence: a modern approach. Prentice Hall, New Jersey* (2010), pp. 480–684.

[7] Padhraic Smyth, David Heckerman, and Michael I Jordan. "Probabilistic independence networks for hidden Markov probability models". In: *Neural computation* 9.2 (1997), pp. 227–269.

[8] John DeNero and Dan Klein. "Teaching introductory artificial intelligence with pac-man". In: *First AAAI Symposium on Educational Advances in Artificial Intelligence*. 2010.