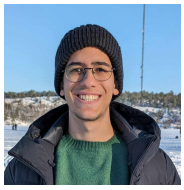
PacMan Capture the Flag

GROUP 8

Saad Guessous
20010928
guessous@kth.se



Peiheng Li
20010315
peiheng@kth.se

Abstract

Multi-agent systems involving teams of cooperative and competitive agents present unique challenges for artificial intelligence. This report addresses these challenges through the Pacman Capture the Flag game environment, where adversarial teams aim to capture pills from opposing territories. We developed a decentralized, utility-based AI system enabling agents to make strategic decisions autonomously based on local observations. The explored techniques, including decentralized decision-making, multi-agent planning under uncertainty, and autonomous cooperation/competition, have widespread applications across multi-robot systems, games, military simulations, and other complex environments. The study yielded promising results, demonstrating the effectiveness of our utility-based solution while also revealing opportunities for further refinement and optimization.

1 Introduction

The field of multi-agent systems has seen rapid growth in recent years, driven by the increasing prevalence of applications that require multiple agents to cooperate and compete in complex environments. Games provide an ideal testbed for developing and evaluating multi-agent artificial intelligence techniques, as they often encapsulate key challenges such as decentralized decision-making, coordination among teammates, adversarial dynamics, and real-time planning under uncertainty.

The Pacman Capture the Flag game presents a multi-agent environment where two teams, each with two or more agents, compete to capture pills from the opposing team's territory while defending their own. The game map is divided into two symmetric halves, with each team spawning on their respective side as ghosts that can kill the opposing team's Pacman agents. When an agent enters the enemy territory, it transforms into a Pacman and can consume pills. If a Pacman returns to its own side with captured pills, they are added to the team's score. The objective is to have more pills on your side than the opponent. Additionally, a super pill allows Pacman to temporarily eat ghosts in the enemy territory. Games typically last between 150 and 240 seconds.

This game presents a rich environment for multi-agent research, requiring agents to navigate the maze, evade opponents, coordinate offensive maneuvers to infiltrate and collect pills, and simultaneously adopt defensive postures to protect their home territory. Solving this game demands an integrated approach that combines path planning, opponent modeling, team coordination strategies, and high-level decision-making capabilities.

Developing effective multi-agent AI solutions for Pacman Capture the Flag has implications that extend far beyond the game itself. The underlying principles and algorithms can be applied to a wide range of real-world scenarios involving teams of autonomous agents, such as multi-robot systems for exploration, search and rescue operations, military strategy simulations, and even multi-agent trading systems in financial markets. As such, this game environment serves as a powerful platform for advancing the state of the art in multi-agent intelligence.

1.1 Contribution

Our primary contribution is the development of a decentralized, utility-based AI system tailored for the Pacman Capture the Flag game. This system enables individual agents to make strategic decisions autonomously based on their local observations and knowledge of the game state. At the core

of our approach is a utility function that evaluates the desirability of each available action by considering multiple relevant parameters, such as the agent's current status, positions of teammates and opponents, remaining pill locations, and strategic priorities like offensive versus defensive posturing.

By continuously re-evaluating the utility of possible actions, our agents can dynamically adapt their behavior in response to the evolving game state. This decentralized approach eliminates the need for a centralized controller, promoting scalability and robustness. Furthermore, our utility function incorporates terms that facilitate cooperation among teammates, allowing agents to automatically coordinate strategies without explicit communication.

1.2 Outline

The remaining sections of this report are structured as follows: Section 2 provides an overview of related work in multi-agent systems and game AI, highlighting key techniques and approaches relevant to our problem. Section 3 details our proposed method, including the implementation specifics of the utility AI system, the parameters considered in the utility function, and the strategies employed for offense, defense, and team coordination. Section 4 presents the experimental results obtained from testing our AI solution against other teams' implementations in a competition setting, analyzing the strengths and weaknesses of our approach. Finally, Section 5 concludes the report by summarizing our key contributions and insights, while also discussing potential avenues for future work and improvements.

2 Related work

The core of The Pacman Capture the Flag game is decision-making, which forms the core of any multi-AI agent system. There are many different approaches to decision-making, such as goal-based, reinforcement learning-based, and so on. Among all of the methods, one of the most robust approaches is the utility-based method. In the related work section, we first briefly introduce the related application of the utility method in multi-agent AI systems, which mainly defines the meaning of this approach. Then we will introduce certain principles, and tricks that help in developing a utility-based agent.

As introduced in [1], utility design in multi-agent systems has been applied in a variety of fields including transportation networks, actions, supply-chain, converting problems, and network games. Thus it is a topic of great

potential that worth discussing.

The key to decision-making using the utility-based method is to calculate a utility score for every action the agent can take and select the highest one. Among many approaches, one of the most common techniques is to multiply the utility score by the probability of each possible outcome and sum up the weighted scores. As introduced in [2], normalizing the probability and selecting the highest score action is called the principle of *maximum expected utility*. A simple mathematical expression can better visualize the intuition:

$$utility\_score = \sum_{i=1}^{n} D_i P_i$$

In this case, D is the utility and P is the probability that the action will occur.

Taking one more step after knowing how to pick one action with utility scores, it is also important to find a way to calculate the utility score. As introduced in [3], the key to utility theory is to understand the relationship between the input and the output and be able to describe the conversion process in a resulting curve. Many ways of calculating the score cans be used, including linear mapping, quadratic, and even logistic function [4].

Even more potential problems may occur during picking an action. One of the most common problems is that among all the actions, some actions that may have a high utility score are not actually suitable choices after all. This could be due to the lack of utility design and flaws in the logic chain. A straightforward way to solve this problem is called *dual utility*, as introduced in [5]. This approach essentially categorizes all the actions into different *buckets*, which are also assigned different weights. The high-weighted buckets are always processed first.

3 Proposed method

This section outlines our decentralized, utility-based AI system for the Pacman Capture the Flag game. We first provide an overview of the approach, followed by details on utility function design, path planning optimizations, and the biggest challenges that we faced during the implementation.

Our approach to solving the Pacman Capture the Flag game revolves around a decentralized, utility-based AI system. At its core, each agent continuously evaluates the current game state from its local perspective using a set of utility functions. These functions consider multiple relevant parameters, such as agent positions, food locations, power pill positions, team scores, and agent status flags, to determine the desirability or utility of each

available action. By dynamically re-evaluating utilities and selecting the highest-valued action, agents can adapt their behavior in response to the evolving game state in a flexible and context-aware manner.

A key aspect of our implementation is the avoidance of collisions between agents and static obstacles. This is achieved by utilizing precomputed breadth-first search (BFS) distances instead of Euclidean distances when calculating utilities involving spatial parameters like agent-to-food or agent-to-agent distances. Before gameplay begins, we perform a BFS from every cell to every other cell on the grid, caching the resulting distances. During the game, agents can efficiently query the precomputed BFS distance between any two cells, inherently avoiding collisions with obstacles while still navigating efficiently.

The utility functions operate on multiple parameters extracted from an agent's local view, including agent positions, food locations, power pills, team scores, and agent status flags. The roles of the key utility parameters are:

- Closest Food/Capsule: Guides offensive food/capsule capturing
- Ally/enemy positions: Enables coordination and threat assessment
- Score difference: Determines offensive/defensive prioritization
- Carried food: Signals when an agent should return to its territory
- Agent state (Pacman, scared): Modulates behavior based on vulnerability

The utility functions associated with each parameter are designed to guide agent behavior towards desirable outcomes. For instance, the utility for the closest food pill is calculated as 1 / (distanceToFood + 1), incentivizing actions that minimize the distance to food. Proximity to allies is handled using a threshold-based function that penalizes actions bringing agents too close to avoid collisions.

Collision avoidance between teammates is also a crucial consideration. Our utility functions penalize actions that would bring an agent too close to an ally by decreasing the utility by a fixed amount, typically -5, when the distance falls below a predefined threshold (e.g., 0.4 units). This simple yet effective mechanism helps agents automatically maintain safe distances from one another, promoting coordinated movement without direct communication.

For the distance to the nearest enemy, we employ four distinct utility functions that account for whether the agent is a Pacman, whether the enemy

is scared, and whether the agent itself is scared. This enables context-aware decision-making based on the threat level and offensive/defensive opportunities. See Algorithm 1 for the calculation of the utility depending on each case.

While agents can theoretically move in all directions, our implementation considers only nine principal actions: up, left, right, down, and the four diagonal directions, along with the option to remain stationary. This discretization simplifies the action space while retaining sufficient maneuverability.

Finally, to prevent agents from becoming stuck in cycles, we maintain a cache of the previous six actions taken by each agent. If a cycle is detected, the agent selects the next action specifically to break out of the cycle.

By integrating utility evaluations, path planning optimizations, and context-aware behavioral modulation, our system enables robust and adaptive agent operation tailored to the adversarial multi-agent dynamics of Pacman Capture the Flag.

Algorithm 1: Calculate Utility for the distToEnemy parameter

Result: utility (Float)

```
distanceToEnemy ← distanceCalculator.GetDistance(potentialNewPos, closestEnemy);
if isPacman then
    if isEnemyScared then
        score += 1/(distanceToEnemy+1);
    end
    else
        if carriedFood > 0 and distanceToEnemy < 5 then
            score -= 100/(distanceToEnemy+1);
        end
        else
            score -= 1/(distanceToEnemy+1);
        end
    end
end
else
    if isScared then
        if distanceToEnemy < 10 then
            score -= 100/(distanceToEnemy+1);
        end
    end
    else
        if distanceToEnemy < 10 then
            score += 100/(distanceToEnemy+1);
        end
    end
end
```

4 Experimental results

In the section on experimental results, we will provide a detailed discussion of our experimental setup, including the configuration of the competition mode and the game mechanics. We will also analyze the outcomes of the competition. Our PacMan agents performed commendably, securing 8th place on Map A and 12th place on Map D. Despite demonstrating high competence, they faced challenges when competing against certain teams. Overall, our

agents met our expectations, yet there remains significant scope for further improvements.

4.1 Experimental setup

The PacMan Capture the Flag game is formed by two teams (red and blue) trying to retrieve as much food as possible from the opponent's side and try to defend the already gotten food. More detailed rules include different food types such as super pills that enable the Pacman to kill ghosts.

The PacMan agents' behavior is based on the PanMan AI script, which can be executed in two different modes of Unity: host and client/server. The features of these two different modes will be introduced in the following sections.

In total, there are three maps used for the competition, Map A, C, and D, respectively. However, we focus on analyzing Competition 3, thus Map A and Map D are mainly used for the experiments.

4.1.1 Host Mode

The host mode is essentially performing a PacMan game in Unity locally. Both the red and blue teams will have the same AI script that empowers them to make decisions. Since the scripts are running locally, the latency in the host mode is really low, making the agent agile and responsive.

4.1.2 Client/Server Mode

The Client/Server mode enables the experiment to be set as running two different solutions as executable clients and competing on one executable server. The execution environment is identical to the host mode. However, this mode has a longer latency for the scripts to run and can potentially lead to unexpected behavior.

4.2 Analysis of Outcome

We analyze our Competition 3 result for Map A and Map D, as shown in figure 1 and figure 2. The first row of the tables is the groups that we competed with, and the second row is the score. A positive score means that we ended up winning, and a negative score means that we ended up losing. The tied game is represented as 0.

At the end of the tournament, our PacMan AI achieved 8th place in the Map A competition and 12th place in the Map D competition. Compared

Group	15	3	16	17	1	4	25	21
Score	83	-43	99	25	-62	18	44	78
Group	5	23	7	26	6	9	24	10
Score	-54	32	-4	33	54	46	-78	79

Table 1: Results for Map A

Group	15	3	16	17	1	4	25	21
Score	32	-24	3	3	-13	-25	35	-34
Group	5	23	7	26	6	9	24	10
Score	-4	22	30	27	14	36	0	50

Table 2: Results for Map D

with other solutions, we outperform certain groups' solutions that are based on single offensive/defensive switching behavior trees. Our agent showed better versatility when making decisions since the actions are decided by the overall utility calculated, and thus not restrained by certain modes our agents are in.

Our agents perform better on a larger map (Map A). By observing the competition process we discovered that our agent makes more aggressive behavior when competing in a smaller and more open area, and more balanced in a larger, more complicated (e.g. more blocks) area. In hindsight, we found that we did more experiments locally on large maps such as Map A and Map C, but we never tuned the weights on Map D, which wasn't released until the day of the competition. A potential solution could be that we try to normalize all the information such as distances, and remove the potential effect coming from the size of the map.

Moreover, we discovered that our agent always ended up with a huge loss against group 1, who proposed a great solution for the dynamic defenders. The effect of their solution is obvious, as we discovered in the match games, after a certain amount of time group 1's agent will start defending their food on their territories. They have a good strategy that can divide the boundary of the map to each of the agents, and hold a great position that always discovers our agents who are trying to "steal" the food in a very quick manner of time, and eats our agent before ours can return. Group 1's agents have exposed many of our solution's drawbacks. Our utility functions make decisions mostly based on distances, and only a few other information is considered. This makes them unable to outsmart defenders in circumstances such as all the opponents are defenders, only crashing into the opponents

and getting eaten when there is no food to defend on our side. Another problem for our solution is that overall our agents are prone to attack, and only perform a few defensive behaviors. It is the case that we didn't have the utility function for circumstances that lead to a stable defense action. Our territory is thus often exposed and not well protected. This problem can be patched by the dual utility method as discussed in the section 2.

5 Summary and Conclusions

In summary, our utility-based multi-agent AI system demonstrated effective performance in the complex and dynamic environment of the Pacman Capture the Flag game, reflecting the solution's robustness in managing decentralized decision-making and agent autonomy. The utility functions in our solution are tailored to the agents' immediate context, such as proximity to objectives and enemy locations, and are adjusted for both offensive and defensive tactics uniformly, facilitating cooperation among team agents. The utility-based approach enabled agents to perform both offensive and defensive roles dynamically, adapting their strategies to the unfolding game scenario.

However, the competition with other groups also highlighted several areas for improvement. For instance, the system's performance varied across different map configurations and against teams with different strategic focuses, particularly against sophisticated defensive tactics when close to the end of game. Further refinement of utility calculations and strategic decision-making processes may improve the system's effectiveness. More advanced ideas can be introduced such the dual utility method that can solve some problems that our solutions are facing.

References

[1] D. Paccagnan, R. Chandan, and J. R. Marden, "Utility and mechanism design in multi-agent systems: An overview," *Annual Reviews in Control*, vol. 53, pp. 315–328, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1367578822000062

[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. USA: Prentice Hall Press, 2009.

[3] D. Mark, *Behavioral Mathematics for Game AI*, ser. Applied Mathematics Series. Course Technology, 2009. [Online]. Available: https://books.google.se/books?id=iJ2pOgAACAAJ

[4] Wikipedia, "Logistic function — Wikipedia, the free encyclopedia," http://en.wikipedia.org/w/index.php?title=Logistic\%20function&oldid=1221464318, 2024, [Online; accessed 06-May-2024].

[5] K. Dill and L. Martin, "A game ai approach to autonomous control of virtual characters," in *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*, 2011.