CS512: Artificial Intelligence

Term Paper
AI in Pacman Ghosts

Abhaysinh Sunil Patil (2018CSB1063)

Gurpreet Singh (2018CSB1092)

Introduction

Pacman is a maze arcade game released in 1980. The maze contains a number of pellets which have to be fully eaten by the Pacman (controlled by a human player) to win. However, its movement is obstructed by a few impervious walls and four ghosts. The key task of the player is to eat all pellets while preventing itself from being eaten by four ghosts. In the original Pacman, all four ghosts have unique personalities. One ghost directly chases the Pacman, while the other two try to position themselves in front of the player. The last ghost switches between chasing and fleeing from the Pacman.

Motivation and Problem Definition

The scenario of the Pacman game can be found to exist in many real life cases. Any situation which involves multiple agents chasing another agent (who may or may not be collecting some reward spread across the area) can be fitted to the description of the Pacman game with slight modifications. For example:

- robot cops chasing an offender
- trying to contain fire in a warehouse/closed area
- nano-robots coordinating an attack against a foreign particle in human body

The main focus of our work is to devise efficient search techniques to improve the chase by the agents.

In terms of the Pacman game, we aim "to devise different search methodologies for the ghosts to chase the Pacman".

The efficiency of a chasing approach can be defined by the reward collected by the Pacman, number of moves or amount of time taken to capture it.

Literature Review

A number of papers related to Pacman, agents exploring a grid to capture a target are available. Work has also been done to apply evolutionary algorithms on the agents. Some Artificial Neural Networks based approaches can also be found.

In the paper available here, the authors have implemented A* algorithm on Pacman ghosts for navigation mesh pathfinding and examined it with Unity 3D game engine. They have drawn the comparison of A* with Dijkstra Algorithm. NavMesh is a tool for representing 3D environments in 2D-like form. This transformation simplifies the path finding in complex spaces.

Another paper discusses the Pacman game as a tool to teach search algorithms. The main effort of the authors is in visualisation of the game to teach depth first and breadth first search.

Most of the papers related to multiple agents and moving targets do not involve coordination among multiple searching agents. The agents are usually dealt with independently. Also, the target is considered to be static or randomly moving. However, coupling the multiple agent target searching with the game Pacman ensures that the target has a well functioning 'mind'. We also implemented one version of the scenario where the target is specifically interested in collecting rewards spread all over the grid. To control the difficulty, we have added a periodic timeout interval for which the pursuers stop the chase.

Preliminaries

Here our decision making agents are the pacman ghosts, which are chasing the pacman, which is controlled by the user.

The core part of the implementation was related to the search algorithm in a graph. A* along with distance heuristic was used for faster search functionality.

Pellet is a small bead like particle, which on collecting, increases the score. Collecting all the pellets results in the completion of the current level. A pellet can also be termed as a reward for the Paman.

The ghosts don't chase the Pacman all the time, they also take breaks in between. This is called a timeout. These two modes are called **chase mode** and **scatter mode (timeout phase).** This will be the duration for which the ghosts will not chase the pacman, and even on contact, the game won't end.

The pacman can move wherever it wants except into the walls. But the ghosts can move only in forward, left or right directions, i.e., they cannot take a U-turn.

Python was used to implement the game, using the **freegames** library for GUI part. And GitHub was used for remote collaboration.

Properties of the game's environment:

1. Fully Observable: All the agents know each other's location and maze topology.
2. Multi-agent: It includes 4 collaborative agents and 1 target agent.
3. Deterministic: The next state can be determined by the current state and the action.
4. Sequential: Current action has an influence in future.
5. Dynamic: The environment changes before the ghost take current action
6. Discrete

The ghost agents are model-based utility-based agents. They evaluate the current state they are in. From the list of the possible actions, they choose the one which provides maximum utility. The total path cost of the next move serves as the parameter of utility here. The agent's objective is to choose the next move which minimizes the total path cost.

In the coordinated A* part, there are two types of ghosts. Leader ghost (darker color) and follower ghost (lighter color). The leader ghost will first run A* and decide its path and then leave a heat map of penalty. The follower ghost will then decide its own path according to the previous modifications. In case of 4 coordinated attack, the darkest ghost has 3 followers. The second darkest has 1 leader (darkest one) and 2 followers (other 2 lighter ones), and so on.

Timeout determines whether ghosts will be in chase mode or scatter mode.

Overlap cost is added to the path cost of a ghost if a position in its path is already occupied by another ghost. This is done for every ghost to ensure that the ghosts do not converge into 1.

Intersection Penalty is added to the path cost of a ghost if a position in its path already exists in the path of the leader ghost.

Decay factor determines the intersection penalty that will be levied on the ghosts. For a state nearest to the Pacman, the intersection penalty is the highest. However, this decays for positions farther from Pacman.

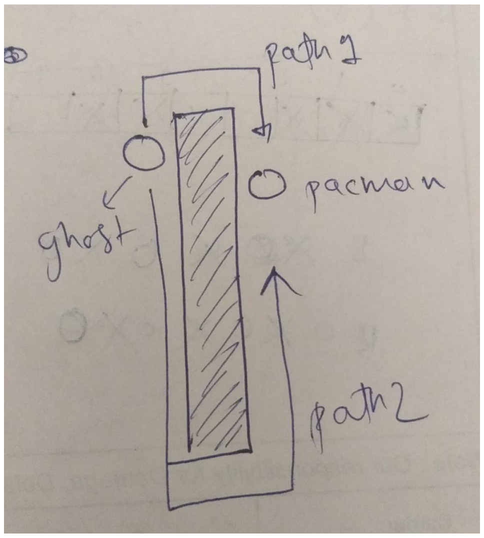
Discount is the difference of the regular step cost and step cost of a pellet containing cell.

Solution/Proposed Model

The basic search algorithms used were A* and GBF (greedy best first). Among all the 4 possible positions where the ghost can move (up, down, left, right), GBF will choose the one which moves it closest to the pacman.

$f(n) = h(n)$, where $h(n)$ can be Euclidean or Manhattan distance

But GBF has some limitations which can be illustrated using the following diagram.



Here we see that the GBF will choose path 2, but actually path1 is more optimal.

There are 5 parts in the code, and the user can input which part to run. Each part has increasing difficulty (except 5th which is quite different from 1-4, hence cannot be directly compared with 1-4). **Part 1 is based on GBF** and **parts 2-4 are based on A*** search, with modifications being smarter and smarter as we move from 2 to 4. In the **5th part**, we made an attempt to do some **prediction of the pacman's steps** by using the fact that the Pacman will eat the remaining food sooner or later.

The first part simply uses GBF search which uses distance heuristic to decide the direction. The results are good but not that great.

The second part uses simple A* search with distance heuristics to determine the best direction which will give the shortest distance to the pacman's position, and make a move in that direction.

The third part uses some extra smartness. In this part we pair 2 ghosts together and **introduce some coordination** between them. This coordination is done by restricting both the ghosts from attacking the pacman on the same side. This can be done by **penalizing overlapping attack strategies** of the two ghosts. This penalty will be added to g(x) for the second ghost. The first ghost will do a normal A* search and the second ghost will make a search on the grid with some modifications. The tiles which are a part of the first ghost's route will be given higher g(x), and hence the second ghost will try to avoid travelling over this route.

It could be possible that this overlap between the two paths might not make a difference, eg: if intersection takes place far away from the pacman's position, i.e., the destination. Hence a **decay function** was used which will penalize more to the tiles near the destination tile (pacman's location). We have used an **exponential decay function**, with a **constant decay factor**.

This coordination is especially useful to **trap the pacman** from two sides of a tunnel and essentially trapping it completely.

In the previous part the pacman can still escape if there is a T-point or an intersection of two paths. Hence in the fourth we used coordination between all 4 ghosts. There is a leader ghost whose path is decided first. Then, the path of the second ghost will be evaluated by applying the overlapping cost. Then, the third ghost and so on. This will make the game quite difficult, as in many cases there will be ghosts present on all the escape routes. To make the game playable we also implemented a **timeout feature**. So that the ghosts switch to scatter mode after some time, and this allows the pacman to escape.

The fifth part uses **A* along with prediction** on pacman's moves and uses that to make its moves. This prediction involves a simple fact that pacman will consume all the pellets sooner or later. Hence we use the **count of remaining pellets** to do prediction. Here the g(x) function for tiles along the paths with pellets will be lesser than the tiles without food. This will induce a bias for the ghosts and it will tend to move on the track with more food, and since the goal for pacman is to eat pellets, hence the ghosts are expected to move along pellet paths. Even if this method doesn't make it easier to catch the pacman, will **introduce breaks in the collection streak**, and increase the playtime for a level.
Initially, g(n) = #steps

Weighted Path Cost:

$$g'(n) = 1*(\#\text{non-pellet steps}) + (w)*(\#\text{pellet steps}); \quad 0<w<=1, w=1-\text{Discount}$$

In this case the heuristic also needs to be discounted. As directly taking the heuristic distance will overestimate the cost if all tiles are having pellets.

Experimentation

The game will start with all the ghosts in 4 corners of the map, and pacman spawned at a fixed position. The game will end in two cases. First, when the ghosts will catch the pacman, and second, when the pacman collects all the pellets.

There are different parameters which will affect the decision making of the ghosts. These parameters are tweaked and observations are made.

There needs to be a **good balance between the chase mode time and scatter mode** time. From multiple tries of different plays over time, one good balance was found. Set the **end_time = 50** and **mid_time = 15**. This means for the first 15 steps will be scatter mode and the remaining 50 - 15 (=35) steps will be chase mode. This distribution seems fair and playable.

If **overlap cost is over 100** then it will be equivalent to infinity as there can't be any path with length > 100 (as grid is 20x20, max path length = 40 for a typical Pacman grid). Hence when this penalty is added to a path, it will be the path with the highest cost. And taking this path means there is no other path available. So as soon as a different path is available (at the next intersection, T-point), overlapping pacmans will split their directions irrespective of the cost of the new splitted paths. Overlapping agents is equivalent to one less agent, hence overcoming overlapping is the top priority. Hence its penalty is infinity.

Intersection penalty decides the penalty of the tile prior to the pacman's position in an A* search algorithm. If this value is kept very high then the number of tiles which are significantly penalized will be large, which is not what we want. If it is too low then the follower ghost may not prefer to attack the pacman from the other side.

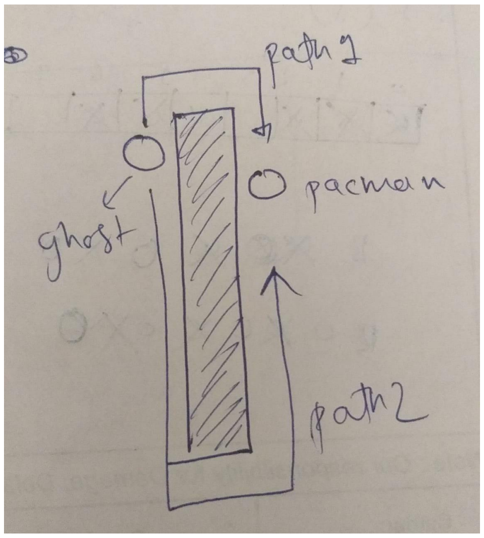
Decay factor decides how far is the effect of intersection penalty significant. The tile closest to the pacman will be penalized according to the value of the intersection_penalty variable, and will slowly reduce (decay as we move away from the pacman). If this factor is too low then the effect of penalty will have a farther reach, and will disrupt the decision making of the follower ghost. And if this value is too low then the follower might not prefer a longer router which approaches pacman from the other side.

Discount decides how much the prediction making ghost prefers the pellet tiles over normal tiles. If this is very low then there won't be significant changes, whereas if its value is very high, then the ghost will prefer only the pellet tiles, which is not something we want.

Results and Discussions

The first algorithm GBF is pretty simple, and works fine, but the ghosts can be easily fooled to follow a single trail, and all the pacman would need to do is proceed on a path without intersecting itself for some distance. This is similar to the snakes game, where you have a trail behind the head, and we find a path without biting ourselves. Here the pacman is the head and the trailing ghosts are similar to the trailing body.

Next comes simple A* which works pretty similar to the previous GBF scenario, except that it takes the most optimum path. One such case where GBF and simple A* differ, is shown below.



The A* will choose path1, whereas GBF will choose path2. In both these cases it was not difficult to complete the game.

The 2 ghosts coordination **worked best** when the **follower and the leader were on the opposite side** (video link) of the pacman. And didn't perform well when both were approaching from the same side (video link). This also happens when the leader is far away (video link). When both were on the same side, the follower generally took a totally different route than that towards the pacman, as both had a common goal to achieve, hence there was a large overlap between the two. Whereas when both were on the different sides, their path hardly overlapped hence it performed good.

The 4 ghost coordination worked best among all these variants. And the problems mentioned above were also observed here (video link). When the leader ghost is far away from the pacman, if the path of the leader coincides with that to the ghost just a few blocks away from the pacman, the closest ghost changes its path and goes to some other way. This is due to the fact that the leader ghost doesn't consider other ghosts while deciding its path.

However when the **leader ghost is close to the pacman**, the **other 3 coordinated very well** (video link) and almost always caught the pacman. One solution to this problem is selecting the leader ghost greedily every time a move is to be made.

The moves by all the entities from one tile to another had one intermediate state as well. Hence in two animations, all entities can move to their adjacent tile.

Intersection penalty = 100 and **decay factor = 0.7** worked very well as compared to some other cases tried. Since we have two steps between each tile, hence for adjacent tiles the decay factor becomes 0.5 (0.7 * 0.7 = 0.49). And with intersection penalty of 100, and decay factor 0.5, the penalty becomes insignificant after about 6 tiles. Which seems pretty reasonable for a 20x20 board. These values are based on our testing experience and knowledge about the game, as testing the environment using automated scripts was not feasible.

For the prediction part, there exists a tradeoff between the number of moves to capture the Pacman and the reward collected by it. This approach reduces the reward collected by the Pacman, but this might come with increased time to capture it.

Conclusions and Future work

This term paper focuses on semi deterministic search algorithms (as ties are broken randomly), but non deterministic learning algorithms can be used for better learning the player pattern, and then making the moves. This move prediction can be based on the behaviour of the player (whether he first prefers to eat bulk of pellets and then clean the scattered pellets or clear the map section by section).

This solution can be further extended along with HMM, when the exact location of other agents might be known at all times. In this scenario a belief state can be maintained which becomes distorted over time. And consolidate the state when next time it is directly visible.

Other modifications like limiting the view of a ghost can be done to increase complexity. For example, this paper relaxes the condition that the environment is fully observable to the ghosts.

The paper available here aims to eradicate the need to recalculate strategy for ghosts everytime Pacman moves.

References

1. https://emitter.pens.ac.id/index.php/emitter/article/view/117/56
2. https://www.scitepress.org/Papers/2016/58646/58646.pdf
3. https://www.aaai.org/Papers/AIIDE/2008/AIIDE08-009.pdf
4. https://webdocs.cs.ualberta.ca/~jonathan/PREVIOUS/Papers/Papers/mamtslong.pdf
5. Pac-Man - Wikipedia
6. https://www.youtube.com/watch?v=l7-SHTktjJc