Final Report

CS 4365 Artificial Intelligence
Professor Vincent Ng

Team: i_am_groot
Siddharth Naik
Mustafa Sadriwala

Introduction

A game of Pacman Capture The Flag (CTF) involves 2 teams each comprised of 2 agents that are given half of a symmetric map to defend. The goal of the game is to collect food from the opponent's side of the map and bring it back to your own side without being eaten. Whichever team has collected the most food by the end is the winner.

To add further complexity the game also includes power pellets or capsules that, if eaten by the enemy, make all agents of the opposing team "scared" so that they can be eaten/killed by enemy pacmen, forcing them to respawn at their original start point.

It becomes clear from this outline that a team of agents has two distinct goals: to collect food from the opponent's side and to defend their own food from the enemy pacmen. These goals can then further be broken down into sub-goals of defending exit points, moving towards dense regions of food, running away from the enemy, etc. Based on these goals we created a team of agents that prioritized particular sub-goals in situations where they were most applicable.

Theoretical Foundation

Part A: Reflex Agents

Simple reflex agents are agents that rely solely on information about the current state of the game. Reflex agents are built by considering commonly occurring pairs of input and output whose pairing is referred to as a *condition-action rule*. A condition-action rule simply states that if a certain condition arises, then the corresponding action should be taken; these can also be thought of as if-else conditions.

The overall structure of both our agents revolves around reflex agents and there are many "reflex actions" that the agents will take that we will refer to later as "base cases". Reflex agents are useful for making quick decisions when given a certain scenario but are constrained by their lack of prior information and future expectations. Without storing prior information a reflex agent can become especially vulnerable to continue making contradictory decisions and get stuck in a cycle of conflicting actions.

Part B: Goal-based Agents

Goal-based agents use both information about the current state of the gate and some knowledge of a goal state that they hope to reach. The decision-making behind goal-based agents differs from reflex agents in that they take into account future conditions and try to determine which future condition serves them better to reach their goal. Although a goal-based agent can be far less efficient than a reflex agent since it requires time to compute future possibilities, it allows for a lot more flexibility as simply changing the end-goal can change it's entire decision-making for a given state.

Part C: Utility-based Agents

Utility-based agents expand upon the idea of goal-based agents providing a numeric value (utility) for a given state. These numeric values are specified by a utility function that maps a state to a utility. Utilities are especially useful in two main junctures where goals fail: first when there are conflicting goals only a subset of which can actually be achieved then a utility function can help us determine the appropriate trade-off and second when there are several goals that can be reached but none of which are given with 100% certainty. Though utility-based functions clearly provide for more specificity than goal-based agents it is a non-trivial endeavour to create a correct utility function.

Theoretical Foundation continued on next page. . .

Part D: Tabu Search

Tabu search is a local search algorithm that restricts possible actions by eliminating those that are considered "tabu" or forbidden to touch. A tabu search usually involves a data structure known as a *tabu*. The *tabu* keeps track of the forbidden states that are not allowed. Tabu search is used primarily to avoid getting stuck in local maxima/minima. This is accomplished by keeping track of the x most recently visited states and asserting that those should not be revisited, where x is the size of the *tabu*. This also has the consequence that a *tabu* of size x can only detects cycles of up to length x - a cycle of length $x+1$ would still persist and potentially result in the local search getting stuck at a local maxima/minima.

Part E: Heuristics

Heuristics are functions that map state variables to a certain value. Heuristics are, in essence, an abstraction of what we previously mentioned as a utility function used in conjunction with utility-based agents. Heuristics allow for uninformed agents to adapt to domain knowledge and make more rational decisions. They also allow for assumptions to be made about the state of the game, saving on overall computation time. However, heuristics can often times be misleading in edge cases and can lead to sub-optimal decisions being made.

Final Agent Description

Part A: Team Overview

Our solution includes two separate agents, one offensive and the other defensive. We believed that having two instances of the same agent would have led to too much confusion on roles and difficulty in divvying up multiple tasks between the two agents so that they actually do different things. By having two distinct agents we can also know their primary and secondary roles. For example our offensive agent is primarily responsible for gathering food and returning it to our side. However, should we be in a position where gathering food is no longer of the utmost necessity, the agent turns to a secondary task of defense.

Both agents initially began as goal-based agents each with distinct goals. However, as we progressed in advancing the rationality of the agents we realized that they may have different goals in different conditions and so created a modular, decision-making hierarchy based on the concept of reflex agents. We created certain base cases for both agents and then implemented a series of condition-action rules that were determined by the current state of the game. Each of these condition-action rules essentially specified the current goal and allowed us to make specific goal-based or utility-based heuristics pertaining to that particular game state. This enabled us to not have to make complex heuristics or utility functions that took multiple conditions into account and allowed us to build up our agents incrementally by taking into account new, unique conditions.

Part B: Preprocessing

One of our primary and essential pieces of preprocessing involves figuring out where the *exit points* are on the layout. These are the points from which we can enter and exit the opponent's side. We figured that the exit space would always be a contested area during a game so figuring out where the exits were would allow us to use them as a position that was either a goal for us or a goal for the opponent.

Additionally given the distancer class that pre-computed the best distance from any one point to any other point we created a safeDistancer and enemySafeDistancer that computed the best distance from one point to any other point on either our side or the opponent's side. These calculations are made using the same distancer class except we create a new layout from the given maze layout that has walls all along the exit points so that an agent cannot cross sides and the distancer must search for the best distance while remaining on one side.

Part C: Defensive Agent

Overview

The overall goal of the defensive agent is to **prevent the opponent from escaping**, not to chase after them or catch them as soon as possible. To accomplish this we create a set of points we want to defend, these points are comprised of both our exit points that we pre-computed and the power pellets on our side. These points are sorted based on their distance from the enemy so that we prioritize defending points closer to them. Finally, we compare what our coverage for these sets of points will be at a given location, for a point in the set to be considered "covered" we must be able to reach that location before the enemy.

Base Cases

If our defensive agent was scared then our base case was to remove all actions (i.e. make them *tabu*) that would cause the agent to be eaten by the enemy pacman. Thus a scared defensive agent will always maintain at least a distance of 2 spaces between itself and enemy pacmen.

If our defensive agent was not scared then our base case was to remove all actions that would take us to the enemy side.

Final Agent Description continued on next page. . .

Condition-Action Rules

Our top-level condition-action rule was to determine if there were any opponents on our side (invaders).
If there were no invaders then we needed to determine what was the best position to guard from if opponents did enter our side. In this case, we changed our goal to be guarding as many power pellets as possible from the enemy. As such we made the set of points we needed to guard equivalent to just the set of power pellets on our side and determined which position would allow us to cover the most of these points. If two actions resulted in equivalent coverage then we chose to take the action that allowed us to be closer to the enemy.
We also needed to decide which enemy to defend against.
If our friendly agent was on the opponent side, then we would choose the opponent that was farther from our friendly agent by a certain threshold[1] (as the enemy agent closer to our friend was more likely to be an enemy defensive agent chasing our friendly offense)[2].
If our friendly agent was on our side then we choose to defend against the enemy closest to ourselves.

If there were invaders on our side then we would pick an action that allowed us to guard the most points. Our points in this instance include both exit points and power pellets as we consider eating a power pellet a viable exit strategy for an invader. As mentioned earlier, we sort these exits based on how far they are from the invader. For each action, we generate a list of points that we can "guard" from that state. We choose the action from which we can guard higher ranked exits (ones that are closer to the enemy). If multiple actions allow us to guard the same set of points, then we choose the state that takes us closer to the enemy (which eventually allows us to eat them).
If there are multiple invaders and only one defender (i.e. our friendly agent was on offense and is a pacman) then we must choose who to guard against. If our friendly agent is on offense then we defend against the invader closer to us.
If our friendly agent is also on defense then we choose to defend the first invader and know that our friendly agent will defend against the second[3] invader.

Part D: Offensive Agent

Overview

Our offensive agent uses a mix of goal-based and utility-based decision-making depending on differing contexts. Some of our disjunctive goals for the offensive agent were to reach a power pellet or to move towards a dense area of food. While moving towards either of these goals we want to also eat food. In conjunction with these we also want to make sure that we are able to escape from the enemy if we are on their side, this means that we should be able to reach some exit point before any and all opponents.

Base Cases

We remove all actions that would cause us to be eaten by the enemy. If we are on our side this mean not going through an exit point that would be one step away from the enemy and if we are on the opponents side then that means any action that takes us one step away from the enemy.

Condition-Action Rules

Before choosing to play offense we go through a series of rules that tell us whether to play offensively or whether to support our defensive agent. These rules are based on our current potential score which is a combination of the current score and how much food we are currently carrying. We determine whether we

1. We empirically chose this threshold to be 6, which also happened to coincide nicely with our max path length for short and long term planning.
2. This is a heuristic for determining who is more likely to be the "offensive" agent on the opponent side and is only really effective if the enemy makes a similar distinction as us of an offensive and a defensive agent.
3. "first" and "second" here refer to the order of agent indices

Final Agent Description continued on next page...

are winning or could be winning by a comfortable margin, if we have already collected a large fraction of the total food and are winning, and if the next nearest food is too far given the time remaining. We outline our secondary defensive strategy in the appropriately titled section below. Also, we determine if we are stuck in a cycle by using a tabu data structure that tracks our most recent positions, if we detect that we have been stuck in a cycle then we define a "cool down" timer to the size of the tabu and while the coold down timer is greater than 0 we automatically choose to play defensively as the conservative choice to break the cycle.

Once we've decided to play offensively we first check if either of our enemies are scared, this implies that we have eaten a power pellet and we make decisions outlined in the power pellet offense section below.

Then we check if we are on our side of the board or not.
If we are on our side of the board then we want to go towards an exit that will allow us the most "leeway" in terms of how much space we have before the enemy reaches that same exit. If there are no exits with at least a leeway of 3 then we choose to go towards the exit that is farthest from ourselves currently so as to hopefully confuse the enemy until something more optimal appears. Besides exits that provide leeway we also check for exits through which we can reach a power pellet before the enemy.
If we are on the opponent's side of the board we first check whether there is some power pellet that we could reach before all other enemy ghosts and before time runs out. If there is then we move towards it using a utility-oriented strategy we termed long term planning that we discuss further below. Otherwise we pick some other goal.

Next we determine which exits we could potentially escape from, these are just the exit points that we can reach before all the enemy ghosts. We pick the closest of these exits and say it is our best exit for escape. If there are no possible exits then we pick the exit farthest from the enemy in the hopes that it provides more time for the situation to change and an exit to become available.

Then we restrict our set of actions to only those that allow us to escape, this means that we should be able to reach some exit before all the enemy ghosts and before time runs out.
As long as there are some actions that remain after this filtering process then we are safe to attempt eating some food. Since there are no reachable pellets, our goal is to now find some area of densely populated food. We use a dense food heuristic that is outlined below to determine this point. We move towards this point using a utility-based strategy we called short term planning that is also outlined below.
If there are no possible actions then we go towards our best exit that we chose previously.

Secondary Defense Strategy

Our final goal is to win the game. This means that we need an overall positive score, and ultimately that simply means having one more point than the enemy. So long as we have a decent margin of error in terms of a lead then we can stop playing offensively and let both of our agents focus on defense. The goal of the secondary defense strategy is to support the primary defense agent by either covering enemies that they cannot or trying to catch the same invader.

If there are no invaders then the secondary defense tries to defend against the enemy that is farther away since the primary defensive agent tries to defend against the closer enemy. It uses the same strategy to defend as the primary defensive agent.[4]

If there are invaders then we make a choice depending on how many there are.
If there are two invaders then we pick the second invader since the primary defensive agent picks the first

4. It tries to "cover" the maximal set of power pellets

one.[5]
If there is one invader then we simply chase that invader since the primary defensive agent will already try to prevent that invader from escaping.

Power Pellet Offense Strategy

When we are on a power pellet we do not have to follow the normal rules of the game so we outline a separate strategy for this scenario.

First we check if we can still reach our closest exit before time runs out. This is simply because we always want to return to our side by the end of the game in case we are carrying any food.

Then we check whether any enemy ghosts are not scared (because we ate and killed them already). We treat these as normal ghosts, and restrict our moves to those that allow us to escape from this ghost. If there are no potential actions then we check to see if there is another pellet that is reachable. If there is then we use the long term planning strategy. If there isn't a reachable pellet then we move towards the closest exit.

Our base cases in this strategy are to avoid any ghosts that are not scared and to eat any ghosts that are scared and are only one step away from us.

Then we filter our actions further by ensuring that we can always either escape by making it to the closest exit before the enemy is no longer scared or we can make it there before the enemy regardless.

If there are still some actions remaining then we use the short term planning strategy to move towards dense food while also taking into account the minimum scared time between our two opponents.
If there are no available actions remaining then we progress with normal decision-making as outlined in the above offensive agent condition-action section.

Dense Food Heuristic

When it is viable for us to eat food we want to try to move towards a region of the map that is "dense" with food meaning that there is a lot of food near each other. To calculate such a desirable region, for each food position we calculate the average distance to all the other food positions and then we also take into account our distance to that food position. Thus we will move towards a locally dense region of food.

Long Term Planning

This strategy is essentially meant to get us to some end destination no matter what. We always want to reach our goal before the enemy but along the way we want to optimize the food we eat that still allows us to reach our goal. We use DFS to look for paths of length 6 at the end of which we should still be closer to our desired position than our enemy and we want to maximize the utility of the path we choose to take.[6]
The utility we calculate is based on whether we eat food or a pellet along the path. For long term planning we weight food and pellets equally. We also want to see if we reach our desired position along our path, if so we add a much larger utility score that we discount from the longer our path has taken to reach it (i.e. we value reaching our destination as soon as possible). At the end of our DFS we return the first action on the path with the highest utility. If long term planning cannot distinguish some best path then we simply take the action that takes us closer to our desired position.

Short Term Planning

This strategy is similar to long term planning except we do not really care about making it to the desired position. Instead we simply want to move in the general direction of our desired position and maximize our

5. Once again "second" and "first" here refers to the order of agent indices.
6. This max path length is empirically deduced by the amount of computation time available.

local utility. We once again use DFS to look for paths of length 6 from our current position. The paths are given utility scores based on how much food and pellets we can collect along them and if they require us moving away from our desired position. We add a small negative weight to moving away from our desired position because ultimately if there is nothing else we still want to move towards that desired position and don't want to take unnecessary moves that take detours away from that position. Additionally we added another parameter to account for how long the opponents have been scared for, this is because during short term planning, if our opponents are recently scared then we want to attribute large negative utility to eating a power pellet as this would be wasting the functionality and overall time we could make the opponents scared. This negative utility gradually becomes a large positive score as the amount of time the enemy is scared decreases to 0. At the end of DFS we return the first action on the path with the highest utility. If short term planning cannot distinguish some best path then we simply take the action that takes us closer to our desired position.

Tabu

We created our own tabu data structure that can be initialized using a max size. The tabu is comprised of a list and a set. Each time a new move is taken the corresponding position is stored in the tabu (appended to the end of the list and added to the set).

Before inserting into the tabu list we check if the position element already exists in the tabu by checking our set. If the position already exists then we add to a consecutive counter variable. If not then we set our consecutive counter variable to 0. If the tabu list is already at its max size then we remove the element from the front of the list and remove that same element from the set and then append our new element.

To check for cycles we check if our consecutive counter variable is larger than the max size of the tabu.

For our use case we chose a max size of 4 as we observed that most of our cycles were of length 1, 2, or 4. Cycles larger than 4 were uncommon and trying to accommodate the tabu size to recognize those large cycles meant it would take us longer to actually detect smaller cycles (since we wait until our consecutive counter is equal to or larger than the max size of the tabu).

Observations

Overall, we were generally happy with the results of our agents. They solved most problems efficiently and it become quite difficult to play against the agents manually especially considering human error and the ability for the agents to make precise calculations about distance.

Some of the things we felt our agents excelled at include our offensive agent being able to find reachable pellets and to go around the enemy in search of them as well as moving towards areas of food that were most rewarding to eat. Also our defensive strategy of guarding positions was quite efficient at making sure enemies couldn't escape and when partnered with a secondary defensive ally became quite impregnable to gain points against.

There were not any large, noticeable errors but still a lot of niche places that could have been improved. For example, originally we had created our offensive agent to never die but soon realized this method only really allowed for draws rather than the wins needed to place highly in tournaments. To combat this we chose to go for pellets even when that meant not leaving exits open to escape. But once we began prioritizing eating pellets over reaching exits it became obvious that there were times when we could become trapped after the opponents' scared timers ran out. This happened mainly because we didn't check if upon reaching the pellet there would be enough time that the opponents were scared by which we could reach the exit. However, this

Observations continued on next page. . .

was also non-trivial to fix as we felt that we couldn't be sure that a path might not exist after eating the pellet such as in cases where we eat the opponent.

There were also multiple occasions when our offensive agent had trouble finding a correct exit to enter through. This was often a source for getting stuck in cycles and really seemed like it couldn't be avoided on certain maps given an optimal enemy that had started first. Additionally, sometimes we would move towards exits that provided additional leeway but that actually moved away from what would be our eventual desired position and created a disconnect between the actions of our offensive agent when on our side and when on the opponent's side.

Another problem we encountered were frequent cycles. We implemented the tabu data structure to counteract these but it couldn't fix all the cycles. One really interesting case that we observed was that our agent would detect a cycle, cool down by playing defensively, but then end up in a state that would lead us back to that same cycle creating an even larger cycle. To stop a cycle like this we could have increased the length of our "cool down" timer but then that would considerably restrict our flexibility by forcing us to make a predetermined action for longer and not take into account the most recent information available.

Conclusions & Recommendations

Our two agents were able to create novel and effective means of winning this game of Capture The Flag by striving for distinct goals in distinct situations, allowing them to be flexible and adaptable to multiple genres of opponents. Though we were satisfied with most of our design choices by the end there were still many areas of frustration.

One big issue was that when we were on offense, we would be wary of the enemy ghost that was actually trying to invade our side, and would thus not venture deep into enemy territory as we were afraid this ghost would eat us. In this same vein, it feels that our offense could have taken more risks when it came to eating food, as we always assumed that the enemy was making optimal decisions when it came to blocking our exits. However, sometimes an enemy was not even chasing us or there were other paths we could take that if the opponent chased us down we could eventually reach an exit before them.

Similarly, our defense was good but there was still plenty of domain knowledge at its disposable that wasn't being utilized. For example the defensive agent simply chose to defend against the first invader instead of trying to account for the amount of food each invader was carrying.[7] Also, when scared our defense did a good job of still trying to defend exits and pellets however it accomplished this by simply always maintaining a finite distance from the enemy. This meant that in this pursuit of a finite distance our defensive agent could become trapped while scared and thus eaten which was a state we really should have tried to avoid more.

I think as far as our current strategy goes we had pretty much saturated most of our ideas. We thought there may have been some way to incorporate adversarial search strategies into our offensive agent especially when it came to allowing for more risks to be taken while collecting food. Also, we had considered implementing a secondary offensive agent for our defensive agent for use cases where we might be losing and need to make some last ditch effort to regain victory.

If we could start the project over we would probably want to look into reinforcement learning techniques to tune features and weights rather than creating so many niche condition-action rules and creating such an

7. We did try to implement a heuristic for picking which invader to defend against that used the amount of food each invader was carrying however it seemed that our defensive agent would switch between invaders too often to be effective and so we stuck to defending against one defender effectively rather than defend against one ineffectively.

Conclusions & Recommendations continued on next page. . .

unwieldy codebase. However, given the fact that we were unfamiliar with reinforcement learning techniques the leap into the unknown seemed like too large of gap to overcome especially by the first tournament deadline and ultimately not worth risking.

References

Edelkamp, Stefan, and Stefan Schrödl. *Heuristic search*. 638. Morgan Kaufmann, 2012.

Russell, Stuart J, and Peter Norvig. *Artificial intelligence*. 3rd ed. 31–52. Prentice Hall, 2010.