PA4 Report

Chantelle Troutman and Piper Stickler

The Pacman Capture the Flag assignment was a multi-agent planning problem in an adversarial environment. Two cooperative agents must operate on a grid-based board while competing against two opposing agents, strategizing their moves together. The challenge was to design agents that can get the enemies' food and also keep them away from their own side. The game environment introduced uncertainty, maze-like constraints with walls, and the need to coordinate offensive and defensive strategies within a time limit. The test of how good the agents are is if they're capable of consistently outperforming baseline agents. We had to develop a strategy where one agent focuses on offensive objectives like finding efficient paths to food while avoiding ghosts, and another agent focuses on defense by tracking and intercepting enemy invaders without leaving its own safe side. We used built-in functions in the pacai API that allowed us to track enemy positions, see if ghosts are scared, and find the food pellets. Several constraints shaped the problem. First, agents must make decisions using only the information available in the current game state. Second, movement occurs within a discrete grid environment with walls, requiring pathfinding rather than simple greedy movement. Finally, the adversarial nature of the game means that optimal plans may change dynamically depending on opponent positions. This meant that the agents would have to rely on multiple heuristics to make the best possible move at a given particular time, and continue responding to changes on the board.

The problem was modeled as a grid-based search environment where each agent interacts with the game state provided by the Pacai framework. The core representation of the environment is the game state object, which includes information about agent positions, food locations, walls, and opponent positions. The state representation includes the agent's current grid position, the set of remaining food pellets in the opponent's territory, the positions of enemy agents, and the grid locations containing walls. These elements are used to determine the agent's available actions and the consequences of those actions. The action space consists of the four cardinal movement directions as well as the option to remain stationary. Actions move the agent to adjacent positions on the grid unless blocked by walls. Each action produces a deterministic state transition according to the maze layout. The objective differs slightly for each agent. The offensive agent seeks to maximize the number of food pellets collected from the opponent's territory while avoiding capture by enemy ghosts. The defensive agent seeks to minimize the opponent's score by intercepting invading agents and protecting food in its territory. Both try to maximize the overall team score. Some assumptions give a start to the solution. First, the agents assume that opponent behavior can be approximated through their current observed positions rather than attempting to predict long-term strategies. Second, Manhattan distance is used as a heuristic estimate of travel cost for pathfinding because it's computationally efficient. Another assumption concerns ghost avoidance. When an enemy ghost is within a certain Manhattan distance of the

offensive agent, the ghost's position and surrounding cells are treated as temporary walls in the search space. This allows the agent to avoid dangerous areas without explicitly modeling opponent behavior. Overall, these conditions and heuristics turn the agents' tasks into simple pathfinding problems that maximize game efficiency.

The computational strategy centered the A* search algorithm, to solve navigation problems within the maze. A* was chosen because it efficiently balances path optimality with computational performance by combining the cost of the current path with a heuristic estimate of the remaining distance to the goal, and we had plenty of heuristics to choose from in the API. The offensive agent uses A* search to compute paths from its current position to a nearby food target. The heuristic function used in A* is the Manhattan distance between the current position and the goal position. This heuristic is admissible in grid environments and helps guide the search efficiently toward promising states without doing anything detrimental. Target selection for the offensive agent is based on proximity to food while filtering out risky targets. The algorithm identifies food positions that lie in "cubbies," which are effectively dead-end locations with only one escape route. If a ghost could reach such a location before or shortly after the agent arrives, that food is considered dangerous and is avoided unless no other food remains. We added this because when the "cubby" food was the closest, the agent would go for it even if a ghost was on its tail, and the ghost would corner it every time. Ghost avoidance is implemented by dynamically adding walls if the offensive agent detects a ghost nearby. This prevents the A* search algorithm from planning paths that move too close to active ghosts, reducing the risk of capture. The defensive agent also uses A* search but with a different objective. Instead of targeting food, the defensive agent targets opponent positions. If an enemy agent has invaded the team's territory, the defensive agent computes the shortest path to intercept the invader. If no invader is present, the defensive agent patrols near the boundary between territories by positioning itself near the midline of the board. We chose to completely separate the tasks into offensive and defensive because there were clear heuristics to use for each and so it was the most straightforward to implement, and it is effective enough to consistently win. This wasn't the most strategic design possible, but it kept runtime down which was why it ultimately ended up being the best choice.

The implementation was structured around an offensive agent responsible for collecting food and a defensive agent responsible for intercepting invaders. They work similarly by retrieving relevant information from the game state, constructing a representation of the environment including walls and temporary obstacles, and then perform A* search to determine a path to the target. The agent begins with retrieving the current game state, including the agent's position, food locations, and opponent positions. From this information, the agent constructs a set of wall coordinates representing impassable cells. Additional cells may be temporarily added to this set to represent hazards such as nearby ghosts. To be clear, this set is dynamic and is redone for every evaluation of a gamestate made by agents. The A* search implementation uses a priority queue to manage the open set of candidate nodes. Each node tracks the cost from the start

position as well as the estimated total cost to the goal using the Manhattan distance heuristic. The algorithm expands nodes with the lowest estimated total cost until the goal is reached. One major obstacle involved handling situations where the chosen food target was unreachable due to obstacles or ghost avoidance restrictions. In these cases, the agent would fail to produce a valid path. This was resolved by introducing a fallback strategy that iterates through alternative food targets in order of increasing distance until a valid path is found. Another challenge involved preventing the offensive agent from entering dead-end regions where escape would be impossible if a ghost approached. The cubby detection logic was made to help the agent avoid risky areas unless necessary. Additional debugging was needed for edge cases where agents temporarily had no valid moves or when position data was unavailable due to the game engine's state transitions. We addressed these by adding safe fallback actions such as returning "STOP" when no valid path could be computed.

The evaluation of the agents was conducted through repeated gameplay against baseline opponents within the Pacman Capture the Flag environment. We evaluated performance through game outcomes and score differentials across multiple matches. Visually, the offensive agent demonstrated strong navigation performance due to the efficiency of the A* search algorithm. It was able to reach food targets quickly while avoiding direct encounters with enemy ghosts. The cubby avoidance mechanism also reduced the frequency of situations where the agent became trapped in dead-end areas. The defensive agent successfully intercepted invading opponents in many scenarios, particularly when opponents entered the defensive territory alone. By patrolling near the midline when no invaders were present, the agent maintained a strategic position that allowed it to respond quickly to threats. However, because the agents rely primarily on local heuristics rather than predictive modeling, they may occasionally make suboptimal decisions. For example, the offensive agent may avoid an area due to ghost proximity even when the ghost is moving away, leading to unnecessary detours. Another limitation is the lack of coordination between the offensive and defensive agents. While role separation simplifies the design, it does not allow agents to dynamically adjust their responsibilities in response to changing game conditions which might have better results. Future improvements could include incorporating probabilistic models of opponent movement, implementing reinforcement learning to adapt strategies over time, or introducing coordination mechanisms between agents. For example, agents could dynamically switch roles depending on the current game state or communicate information about opponent positions, possibly both attacking the other side at once. Or, we could have implemented Q learning instead of A* search.