ME 133b Final Report

Big picture

Our project was recreating the classic arcade game Pac-Man. We were inspired by the unique personalities of the ghosts: Bliny, Pinky, Incky, and Clyde. Their "personalities" are attributed to their different styles in pursuing Pac-man. In other words, they have different path finding algorithms. Our goal was to solve how to catch a moving target, with multiple pursuers. This required determining how to coordinate pursuers to corner the target. We explored using different grids, goal nodes, and parameters to optimize (shortest path, repulsion of ghosts, etc.). The challenge was determining what behavior we wanted the ghosts to have. We had to keep in mind that we didn't want the ghost to follow the same path, that each ghost found an optimal path (based on the paths already computed, but not restricted by these paths). We used our understanding of path finding algorithms like A*, to base our planner.

Approach

Our overall goal was to coordinate four ghosts to pursue Pac-Man. There were two main approaches we took: sequentially planning for each ghost and simultaneously planning for all the ghosts. Both methods plan within the same time step, and therefore the difference in order could be overlooked (Figure 1).

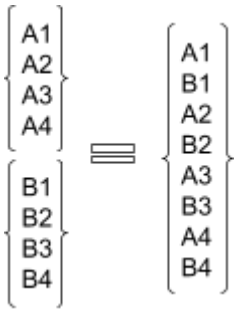

Figure 1: Sequential versus simultaneous.

We used A* to compute our paths. Since the ghosts' objective is to capture Pac-Man, they could take an aggressive approach of each following the shortest path. However, if the ghosts eventually merge to the same tile, then all the ghosts would follow the same path (left panel of Figure 2). For this reason, we must guarantee that each ghost can compute unique paths. Another behavior we wanted to avoid was trailing. Thus, we wanted the ghosts to avoid following the same path. This was done through either blocking or penalizing.

Sequential Planner:

In the sequential code, we solve for each ghost path separately. We first compute and sort the Euclidean distance from each ghost to Pac-Man. We then determine the path for the closest ghost to Pac-Man using A* and block the resulting path for the next closest ghost repeating A*. If no path is found for the ghost due to nodes from prior ghosts blocking any paths towards Pac-Man's position, we reinitialize the grid and compute the optimal path from that ghost to PacMan. This may cause overlap in the paths planned towards Pac-Man but since the closest ghosts already reach Pac-Man, the farther ghosts will never overlap in position with the closer ghosts. The result of this approach is shown in the right panel of Figure 2. This approach ensures that all Ghosts are moving towards Pac-Man in various directions.

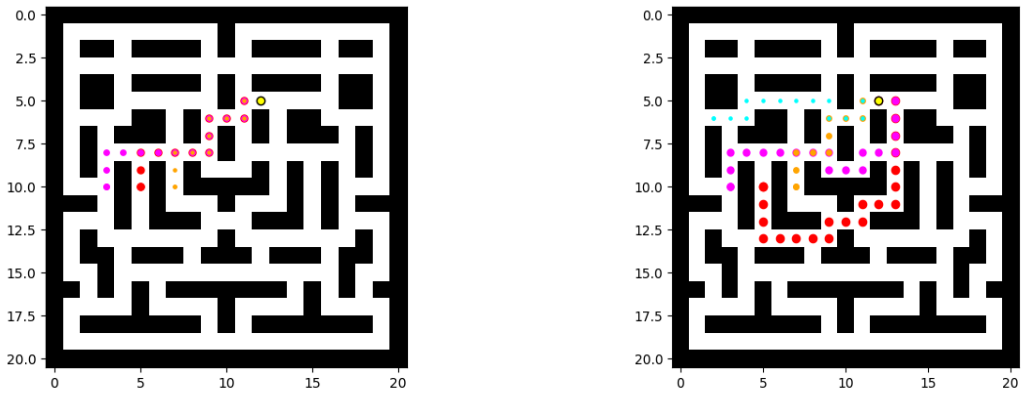

Figure 2: Left: Trailing Ghosts example where each . Right: Sequential A* planner after blocking nodes. The yellow circle indicates Pac-Man's position.

The complexity increases when we add two more ghosts. If Pac-Man is in a hallway, then after the first two paths have been solved for, no solution will be found afterwards for the remaining two ghosts. Similarly, if Pac-Man is at a 3-way intersection, then the last ghost will be unable to find a path. To account for these scenarios, if the planner returns an empty list, then we ask the planner to solve for the shortest path with the original grid. Without this additional step, the ghosts would remain still since they have no path to follow. Refer to the video to see this implementation with our GUI.

Region Planner.

Since Pac-Man always has at least two directions that a ghost could come from, we had the first two closest ghosts still be a sequential planner. However, since the first two ghosts are

sequential; having the other two ghosts also being sequential means that planning is not possible and thus the optimal path is taken. Since the ghosts are further away, it might be better for the remaining two ghosts to path to where Pac-Man might approach. To do so, we have the ghost target a point within the region around Pac-Man. This region would be found by going n steps deep in a breadfirst exploration around Pac-Man and taking all of those points within that region. This represented all of the possible locations that Pac-Man could be at after doing n moves. From there, the ghost would choose a location randomly within this region and path towards using A*. Since it was possible for two ghosts to target the same point,we had the ghost target 2 different points so that the ghost could cut Pac-Man off at different locations. Refer to the video to see this implementation with our GUI.

Multi-dimensional Planner:

To simultaneously coordinate the path of all ghosts, we implemented a multi-dimensional path planner using A* algorithm. For this, we consider one node of our graph to contain the coordinates (row, col) of each ghost chasing pac-man. We only use two ghosts with this planner to reduce the number of nodes generated and searched using our planner to 21^2 nodes. In Figure 3, we show the resulting paths of ghost 1 and 2 with no separation between ghosts in the estimated cost-to-go function (left panel) and separation enforced between the paths the ghosts take (right panel). Refer to the video to see this implementation with our GUI.

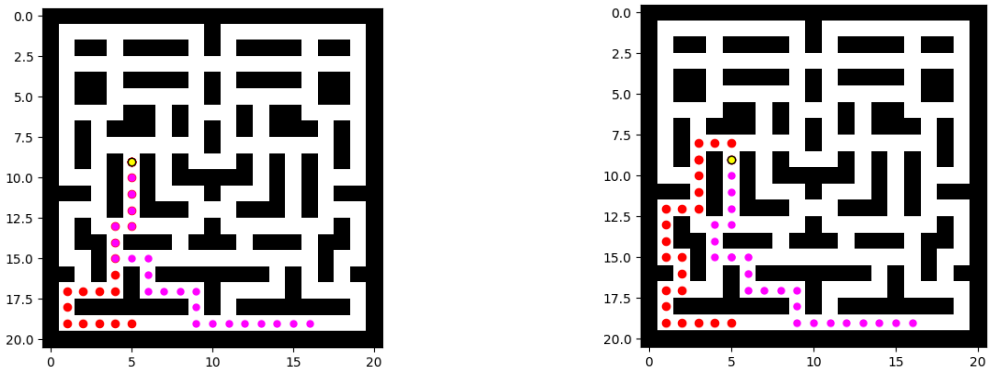

Figure 3: Left: Multi-dimensional planner with no separation between ghosts in A* estimated cost-to-go ($\beta = 0$). Right: Multi-dimensional planner with separation between ghosts in A* estimated cost-to-go ($\beta = 0.5$). The yellow circle indicates Pac-Man's position.

Technical Details

Sequential Planner

Our 2D graph is a maze of continuous flow along a singular pathway. The ghosts and Pac-Man are often limited to two directions to move in. At most, Pac-Man has four possible directions to move in, if he is at a 4 way intersection. Furthermore, the grid is 21x21 and symmetric about the vertical axis. We used '#" to represent walls.

The base cost function used was total_cost = cost to go + cost to to reach, where the cost to go was calculated using Manhattan distance and cost to reach was 1 per step.

Region Planner

To find the region around Pacman, we did a DFS search around pacman goin n-steps deep to get the locations that Pacmam can be at. The target nodes for the two furthest ghosts from Pac-Man's current location were assigned by randomly selecting two nodes from the DFS. The two nearest ghosts traveled towards Pac-Man's location similar to the sequential planner.

Multi-dimensional Planner:

We used the A* algorithm as the basis for our path planner. However, since each node is four-dimensional, we assigned a different set of actions from the conventional 2D grid-based planners. The actions are pulled from $\{\leftarrow,\ \rightarrow,\ \uparrow,\ \downarrow,\ 0\}$ where each arrow corresponds to one of the ghosts moving over to the adjacent grid cell from its current position in the prescribed direction. The last element of this set 0 defines the ghost as stationary. We define all combinations of two elements from the given set as our actions for each node except the case where both ghosts are stationary (0,0). Figure 4 highlights a few example actions both ghosts could take. We iterate through these actions using our A* algorithm to identify neighboring nodes to add to our onDeck queue. To ensure adequate separation between ghosts during path planning, we enforce the following cost-to-go, $\hat{C}_{togo} = d(n_{current}, n_{goal}) - \beta \cdot s$ where d is the Manhattan distance between the current and goal node and s is the Manhattan distance between ghost 1 and ghost 2 position. See Figure 3 for examples of how adjusting β alters the resulting path jointly planned for ghost 1 and 2.

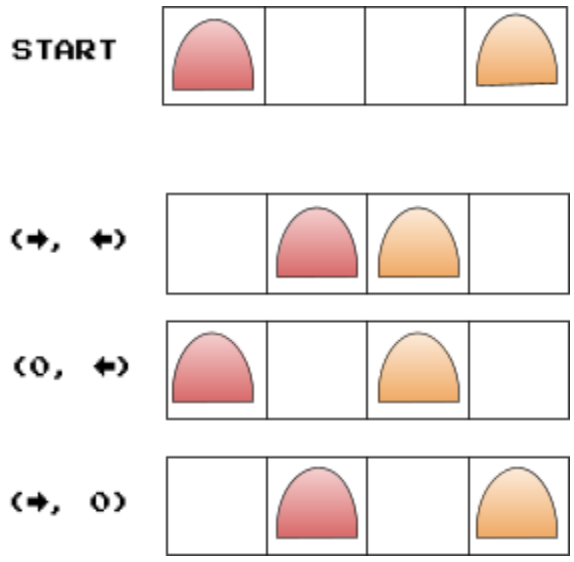

Figure 4: Example actions the set of ghosts takes in our multi-dimensional planner.

Method	Implementation	Pros	Cons
Sequential	In order of closeness to Pac-Man, this algorithm solves for one ghost path at a time, each time blocking out the path found for the next solution. If no solution is found, it solves for the shortest path on a cleared grid.	The ghosts are aggressive and trailing is uncommon.	Computationally expensive
Sequential + Region	Since Pac-Man can always be approached from two directions we had our two closest ghost target pacman in this way. The two further pacman then instead went towards a random point chosen n away from pac man.	Less computationally expensive than sequential. Possible to intercept the targets.	Not as aggressive as sequential.

Multi-dimensional	This approach simultaneously solves for the path of multiple ghosts. Each node contains the positions of all ghosts and the actions performed on each node define joint movements for all ghosts.	Jointly plans a path for all ghosts towards Pac-Man each iteration in the game with a cost function that penalizes the proximity of ghosts.	Computationally expensive due to the number of nodes ($21^2 = 2008$) to generate and search through and the number of actions ($5^2 - 1 = 24$) to iterate through during A*.

Conclusions

We found that there were many trade-offs to consider when implementing the various planners with the goal of all ghosts reaching and cornering Pac-Man. In the sequential planner, we solved for the path of one ghost at a time in order of distance from Pac-Man's location and blocked off nodes from the previous ghosts' paths for each subsequent ghost. While this approach created unique paths for each ghost to corner Pac-Man, the main issue is that it was computationally expensive. The region+sequential planner performed a DFS around Pac-Man to identify places n-steps deep from Pac-Man's location for the farthest two ghosts from Pac-Man to target while the closest two ghosts path plan towards Pac-Man's location. This approach was less computationally expensive than the sequential planner and planned to locations where Pac-Man could move to. However, this planner was not as aggressive in reaching Pac-Man than the sequential planner. The multi-dimensional planner simultaneously planned the ghosts' paths each iteration of the game. With this approach, we could actively control the degree of separation between ghosts as they approach Pac-Man. However, due to the number of nodes searched for and generated along with the number of actions to iterate through in A*, this approach was computationally expensive so we limited it to only two ghosts. One area we hope to explore in the future is the use of the region+sequential planner as we are able to target locations where Pac-Man could go to with this approach.