Sujay Jakka

Svj0007@auburn.edu

COMP 5600

March 18, 2025

Assignment 2 Report

In this report, I will detail my algorithmic and design choices for Assignment 2. This report will consist of 5 sections, each dedicated to explaining my thought process for each question of the assignment. Furthermore, most of everything that is said in this report is also commented in my code to prevent any confusion.

Question 1

Question 1 asks us to create a new evaluation function for the Reflex Agent class. The goal is to come up with a "score" for a state-action pair, which is essentially a score for the successor state. We were given starter code which generates the successor state, the position of Pacman in the successor state, the food object for the successor state, and the ghost states in the successor state which contain data about the ghosts such as the scared time of a ghost. I wanted to create a function that utilizes as much information as possible. I created a function that utilizes the food locations, capsule locations, ghost locations, and ghost scared times in the successor state. My algorithm kept track of a "change in score variable". This is basically a number that will be added to the successor state's built in score. This sum will then be returned by the evaluation function. The functions first finds the the distance of the food that is closest to Pacman. I then multiply 10 with the reciprocal of this distance, and then add this to my change in score variable. The reason why I chose 10 is because that is how much that gets added to the score if Pacman eats a food pellet. The intuition here is that the closer you are to the food pellet,

the closer to the actual value of the food pellet will be added to the change in score variable. Next, I find the distance to the closest capsule from Pacman. I then multiply 30 with the reciprocal of this distance, and then add this to my change in score variable. I chose the value 30 arbitrarily. I wanted to choose a number that was decently higher than the score gained from eating a food pellet. Lastly, I looped through all my ghost positions in the successor state, and found the distance from Pacman to each ghost. Just to clarify the distance used throughout this assignment is Manhattan distance. I first check for a given ghost if it has a scared time that is greater than 0. This means that the ghost is currently scared because Pacman has ate a capsule(power pellet). I then add the product of 75 and the reciprocal of the distance from Pacman to the ghost to the change in score variable. I chose 75 to heavily reward successor states that are closer to ghosts when Pacman has ate a power pellet. This will incentivize Pacman to eat the ghosts. However, if the scared time is not greater than 0 this means that the ghost is not scared and Pacman cannot eat the ghost. My function heavily penalizes successor states where Pacman is within 2 Manhattan distance units away from a ghost. For each ghost, it will subtract 150 from the change in score variable. If the ghost is not scared and the ghost is not within 2 Manhattan Distance units, it will subtract the product of 5 times the reciprocal of the distance to that ghost from the change in score variable. The number 5 was chosen arbitrarily through trial and error to find a value that will allow the function to pass all the test cases. Lastly after the algorithm is done looping through all the ghosts positions, it will return the sum of the score of the successor state and the change in score variable as the result.

Question 2

Question 2 asks us to return the action that results in the best successor state using the minimax algorithm given the current state and max depth. I implemented this algorithm very

similarly to the pseudo-code provided in the lecture slides and I made sure to follow the hints provided in the assignment instructions such as implementing helper functions that are recursive. The way I implemented the getAction function for the MinimaxAgent was to create two helper functions titled “min_ghost” and “max_pacman”. The min_ghost function tries to find the successor states that minimize Pacman’s score while the max_pacman function tries to find the successor states that maximize Pacman's score. For the min_ghost function I have three parameters which are the current state, current depth, and the current ghost. The current ghost variable is basically a counter letting the min ghost function know that it is time to switch over to the max_pacman function. For example, if there are 2 ghosts and the current ghost variable is 2 this means that it is time to switch over to the max_pacman function. However, if the current ghost variable is 1 then the min_ghost function will call min_ghost function again but this time we will be evaluating the successor states created by the legal actions of the 2^{nd} ghost not the 1^{st} ghost. For the min_ghost function the algorithm starts out checking if the game is over, if it is then it will treat this node as a terminal node and return the score of the current state. If not, we initialize a variable called minimum score to positive infinity. Furthermore, we find all the legal actions of the current ghost and set it to a list called “legal actions”. We then loop through each legal action and find the successor state given the current state and the legal action. I then check if the current ghost is less than the number of ghosts, if it is then I will call min_ghost function again except I will set the current ghost to the next ghost(e.g. if the current ghost is one I now set it to two). However, if the current ghost is equal to the number of ghosts then this means that the algorithm needs to switch to a max node and it will call the max_pacman function which has parameters current depth and current state. We will pass the current depth plus 1 for the current depth and the successor state for the current state. The min_ghost function will be the function to

increase the current depth once it has finished calling min_ghost on all the ghosts in the game. I implemented it this way because for every max layer there is a min layer for each ghost. Lastly, for every function call to either min_ghost or max_pacman, we take the output from this function call and compare it to our minimum score found so far. The algorithm then updates the minimum score if this new score is lower. The min_ghost function then returns the minimum score found across all successor states.

The max_pacman function is extremely similar to the min_ghost function. Rather than having three parameters which are the current depth, current state, and current ghost, we only have the current depth and current state. The algorithm checks to see if the current state is a terminal node. It checks to see if the current state is the final state(either Pacman has won or lost) or if the current depth has exceeded the maximum depth, in either case it will return the score of the current state. The algorithm then initializes a list of legal actions for Pacman from the current state. The algorithm also initializes two more variables which are the maximum score found so far and the best action that leads to the successor state with the maximum score. The algorithm then loops through each of Pacman's legal actions and finds the score of the successor state by calling the min_ghost function. It calls the min ghost function as the next layers from the max layer(Pacman layer) is the min layers(Ghost layers). It then checks to see if the score of the successor state is the best found so far, if it is then it will update the maximum score variable and then set the best action to the current action. Lastly, the function will return the best action if the depth is 1. If the depth is not 1, it will return the maximum score. We return the best action if the depth is 1 because that implies the current node is the root node and the objective of the getAction function is to return the best action from the current state(root node). Lastly, we call

max_pacman function outside both helper functions and set the current depth to 1 and the current state to the gamestate that was given.

Question 3

Question 3 asks us to implement the minimax algorithm with alpha-beta pruning. I reused my implementation from question 2 but with some slight modifications. I made sure to implement this algorithm similar to the pseudo code provided on the slides. I first added an alpha parameter and a beta parameter to the min_ghost helper function and the max_pacman helper function. I kept track of the minimum score in the min_ghost helper function and I initialized this to positive infinity like in question 2. However, I also updated the beta parameter if the score from the successor state was less than the beta value. I also included a check after each successor state was evaluated to check if the minimum score(essentially beta) was less than alpha. If it is, then it will return the minimum score. If pruning never happens in the function, then the function will just return the minimum score. The max_pacman function is very similar to this except that we keep track of the maximum score and initialize it to negative infinity. The helper function also updates the alpha parameter if the score from the successor state is greater than alpha. Similar to the min_ghost function there is also a check after each successor state to check if the maximum score(essentially alpha) is greater than beta. If it is then it will return the maximum score. If pruning never happens in the helper function, then the function will return the maximum score found across successor states. Lastly outside both helper function and inside the getAction function, I call the max_pacman helper function with 1 as the current depth, gamestate as the current state, and negative infinity as my initial alpha and positive infinity as my initial beta. This call is what starts the minimax algorithm with alpha-beta pruning.

Question 4

Question 4 asks us to implement the expectimax algorithm in the getAction function of the ExpectimaxAgent class. This algorithm was very trivial to implement. I reused both helper functions(min_ghost and max_pacman) from question 2 except that I made a few minor changes. For the min_ghost function, I renamed it to “expecti_ghost”. Rather than keeping track of a minimum score which finds the minimum score across all successor states, it just calculates the average of the scores across successor states. It does this by adding to the “expected score” variable the score of the successor state divided by the number of legal actions from the current state. This expected score is what we return rather than the minimum score. The only change made to the max_pacman function is that it calls this expecti_ghost function rather than the min_ghost function. Outside both helper functions but inside the getAction function, we call the max_pacman function with a current depth of 1 and a current state equal to the gamestate given.

Question 5

Question 5 asks us to create a new evaluation function that evaluates a state rather than a state-action pair. I basically did exactly what I did in Question 1 where I used the nearest distance to food, nearest distance to capsule, ghost scared times, ghost distance, and the default score of the state. The only thing I changed is instead of evaluating a successor state I am evaluating the current state. My intuition behind reusing my previous code was that I wanted to run my previous implementation first to see what areas of the evaluation function I might need to change or improve. I wanted to use it as a baseline. However, it happened to pass all the auto grader’s tests which is why I decided to reuse my previous implementation of the evaluation function with some changes for this new function.