Bryan Xu
bryanxu@ucsb.edu
6849707

My code consists of the evaluation function and the expectimax function. My evaluation function only considers three things: current game score, food score and ghost score. The current game score is the predefined game score, which means that my pacman will prioritize increasing the score.

My food score is calculated based on the following. First, it finds the distance to the closest food item. Then, it multiplies this by negative 2. This is to discourage the pacman from going long distances. After that, I factor in the amount of food remaining. This is done by adding (number of food left * -15). I implemented this because my pacman was often avoiding food if there were few quantities left. For example, if we had two pieces of food left, but they were far apart, my pacman wouldn't take the second to last piece in the past, because it would lead to a lower score. This change should fix that.

My ghost score only considers ghosts within 4 spaces of pacman. Then, we change the score based on whether or not the ghost is scared or hostile. If the ghost is scared, we add the distance calculation to the score, but if the ghost is hostile, we subtract from the score. This means that our pacman approaches scared ghosts, but runs away from hostile ones. By adding all of these sums, I could get the score for my evaluation function.

For my expectimax function, I implemented three functions in total. These were the expectimax function, the max function, and the expected function. The expectimax function evaluates the current gamestate when we reach the maximum depth, and if not, it determines if the current agent index is a pacman or ghost. If it is a pacman, it runs the max function; otherwise it runs the expected function. The max function simply tries all of the next available moves, and finds the maximum value of all of them. This maximum value is then returned to the recursive call of expectimax. In my expected function, I first found the distribution of the ghost moves by copying and pasting in the getDistribution function found inside ghostAgents.py, and changing it to work with my program. I then used this to calculate the probabilities of each move to find the expected value. This was also returned to expectiValue to be used. I originally encountered a bug with my expectimax function because I was going out of bounds for the agent indexes when the depth > 1, but it was resolved by taking the modulo of agent index with the number of agents. Other than that, most of the work came from understanding the pacman game interface, and defining a working evaluation function.

I tried many different variations of the program, from valuing the ghost distance a lot, to having a wide search range for ghosts, to accounting for the capsule distance, but these all had their own problems. My first implementation made it so that the distance of the ghost was raised to the power of 10, which made pacman get stuck often trying to avoid the ghosts. Since it was so focused on avoiding the ghosts, the pacman would never try to pursue food. When the range where we accounted for the ghosts was too high (7 or 8), it also led to the pacman failing because pacman would run away before the ghost was even a threat. When I tried to account for the capsule distance in my evaluation function, there were cases where the pacman would get stuck between deciding if it wanted to go towards a capsule or towards food. Ultimately, eliminating that choice allowed the pacman to make its choice easier.

Overall, this was a fun project to learn about how expectimax functions work, and to learn about how important evaluation functions are for AI. My score dramatically improved since the beginning, and most of it was due to slight tweaks in the evaluation!