Part 3 Report

Team Name: Dunder Mifflin (Abhijeeth Singam, Saravanan Senthil)

Team Number: 8

Making the A matrix:

The A matrix describes the probabilities that a state-action pair leads out of a state and into a state. The value of probability will be negative for the states it leads into and positive for the state it leads out of.

The dimensions of A are (number of states, number of state-action pairs), which ends up being (600,1936). To create A, we first create a list of all state-action pairs, and then iterate through them and evaluate them one at a time. For each state-action pair, we used the "get_prob(state, action)" function (copied over from our part 2 code) to get all the possible outcomes and their probabilities. Using this, we were able to easily create A by simply adding the probability to the start state and subtract the same from the final state. This takes care of self-loops as the probability that it remains in the start state is both added to and subtracted from the value corresponding to the start state. Thus, self-loops are not counted.

For those states with no actions, or rather the "NONE" action (states where the monster has 0 health), A is created differently. The value of the start state for that state-action is set to 1, and all other element are left as 0.

Code to create A:

```
def get_A():
    A = np.zeros((len(idx_to_state),len(idx_to_state_action)))
    A_list = []
    for idx, st_act in enumerate(idx_to_state_action):
        if st_act[1] == 'NONE':
            A[int(state_to_idx[st_act[0]])][idx] = 1
        else:
            probs, states = get_prob(list(st_act[0]), st_act[1])
            for i, prob in enumerate(probs):
                A[int(state_to_idx[st_act[0]])][idx] += prob
                A[int(state_to_idx[tuple(states[i])])][idx] -= prob
    for i in A:
        A_list.append(list(i))
    return A, A_list
```

Finding Policy:

After using `cvxpy` to solve the set of constraints and optimise the objective, we store the final `x.value` . Using this, we are able to evaluate how beneficial a specific state-action pair is for IJ. Thus, to get the policy for a state, we go through all state-action pairs corresponding to that state and pick the one action of the state-action pair with the highest valuation in `x.value`.

Understanding the Policy:

The policy for part 3 remains somewhat similar to part 2 except that IJ chooses his actions in a slightly more aggressive and risky manner. Firstly, regarding the trends being seen in the policy:

- The overall strategy is simply to reach EAST while incurring as little negative reward (including step cost) as possible, and then continuously hit the monster until the monster dies. In some cases (when IJ has arrows at EAST), IJ will choose to SHOOT MM instead of HIT-ting. This is due to the higher accuracy of the arrows and the fact that IJ will lose them after being hit once.
- One interesting thing we noticed was IJ's tendency to try to avoid MM's attack and the manner in which IJ did it. IJ often tries to avoid MM's attacks by moving to/staying in WEST, NORTH, or SOUTH. This was seen in part 2 as well. This trait of IJ is probably due to out relatively low step cost (-5) which makes the step cost of avoiding MM's attack muck lower than the reward of getting attacked by MM. One difference we noticed here was that in some places where IJ chose to GATHER in SOUTH in part 2, IJ now chose to STAY instead. We attributed this change in decision to the 15% probability that STAY teleports IJ to EAST which is his prime attacking position. Thus the aggressive action of STAY over the safe action of GATHER enables IJ to have a chance of saving one step to reach EAST. Also, given the 15% chance that he teleports and the 50% chance that MM remains in the ready state when he teleports, this aggressive action has a very low chance of backfiring.
- We attribute a lot of these changes to the absence of the reward for killing MM and the discount factor.

Multiple Policies?

There are two cases within the idea of having multiple policies, (i) multiple policies without changing any of the parameters (A, R, alpha), and (ii) multiple policies being created with slight changes in "interpretation". Let's address both of these:

1. The only way to have multiple policies for the same parameters is if the policy is not absolute, i.e. multiple state-action pairs are equally beneficial (nearly impossible unless intentional). In this case, the chosen policy would depend on how the code to extract the policy is written and how the code accounts for this case. Thus, for such a situation multiple policies would be possible.

2. When creating the constraints for our MDP, alpha represents the probability that a certain state is the starting state. For a "general" case, alpha is initialised with every state having the same probability of 1/600, but in a "specific" case alpha would instead have a value of 1 for the starting state and 0 for all other states. This change in alpha can lead to a different policy.