CS7.301 **7^{th} March, 2020**

MDL Assignment-2

Vikrant Dewangan, Roll No.- 2018111024
Mohammad Nomaan Qureshi, Roll No.- 2018111027

Contents

1 File Structure

```
team_29
├── outputs
│   ├── task_1_trace.txt
│   ├── task_2_part_1_trace.txt
│   ├── task_2_part_2_trace.txt
│   └── task_2_part_3_trace.txt
├── task_1.py
└── team_29.pdf
```

2 Task 1

2.1 Problem Statement

To finish Lero's quest, we have to implement a value iteration algorithm which maximizes the probability of Lero getting the final reward at the end to kill the Mighty Dragon(MD). Lero can perform 3 actions -

1. **Shoot**

   Under this action, Lero reduces his stamina by 50 points and loses the arrow. He has 0.5 chance of hitting the MD(Mighty Dragon), upon hitting the dragon, it's health reduces by 25 points. If Lero's stamina is 0 or has 0 arrows, he can't shoot.

2. **Dodge**

   If he dodges with stamina 100 points, his stamina reduces to 50 with probablity 0.8 and to 0 with probability 0.2. If his stamina equals 50 points, this his stamina drops down to 0 always. Lero is smart and picks up the dodged arrow with probability 0.8. Else, if Lero's stamina is below 50 points, he won't be able to dodge.

3. **Recharge**

   Lero can recharge and increase his stamina by 50 points. This only happens although 80% of the time. Else, it stays the same.

Each action taken by Lero has penalty of (-5) points.

2.2 Implementation

The following was considered as a state -

$$S = (\text{enemy\_health, number\_of\_arrows, stamina})$$

where

$$\begin{aligned} enemy\_health &\in [0, 4] \\ number\_of\_arrows &\in [0, 3] \\ stamina &\in [0, 2] \end{aligned}$$

The following formulae has been used -

$$V_{i+1}(s) = max\Big[\sum_{s'} P(s, s')\big[R(s, a, s') + \gamma V_i(s')\big]\Big]$$

until

$$V_{i+1}(s) - V_i(s) < \delta$$

Note that the reward function $R(s, a, s')$ is the following -

$$R(s, a, s') = \begin{cases} -5 & s' \notin T \\ 0 & s' \in T \end{cases}$$

where T is the set of terminal states.
The *stam*, *enemy*, *anna* keep track of the number of the state. The *final* stores the utilities upon reaching taking the corresponding action

```
for i in range(60):
    stam = i%3
    enemy = i//12
    anna = (i % 12)//3
    arrows = (i%12)//3

    if enemy is not 0:
        final = [-1e11 for i in range(3)]

    if stam > 0 and anna > 0:
    # SHOOT
        final[0] = 0.5*(util[enemy-1][anna-1][stam-1]) +
            0.5*(util[enemy][anna-1][stam-1])
        final[0] = final[0]*gamma + 0.5*(reward + penalty) +
            0.5*(penalty)
    if stam > 0:
    # DODGE
        if stam is 2:
```

```
        # 100 points
            final[1] = 0.8*(0.8*util[enemy][(anna+1)%4][stam-1] +
                0.2*util[enemy][anna][stam-1]) +
                0.2*(0.8*util[enemy][(anna+1)%4][stam-2] +
                0.2*util[enemy][anna][stam-2])
            final[1] = final[1]*gamma + penalty
        else:
            final[1] = 0.8*util[enemy][(anna+1)%4][stam-1] +
                0.2*util[enemy][anna][stam-1]
            final[1] = final[1]*gamma + penalty

    # RECHARGE
    final[2] = 0.8*util[enemy][anna][(stam+1)%3] +
        0.2*util[enemy][anna][stam]
    final[2] = final[2]*gamma + penalty
```

The SHOOT, DODGE and RECHARGE calculate the value for the respective actions and finally the max value is taken to be the next utility.

```
    #CALCULATE MAX UTILITY
    policy = max(final[0],final[1],final[2])
    ind = 0
    if policy is final[1]:
        ind = 1
    elif policy is final[2]:
        ind = 2

    line += actions[ind]
    line += "\n"

    # UPDATE EQUATION
    if abs(policy - old_util[enemy][anna][stam]) >= delta:
        key = True
    old_util[enemy][anna][stam] = policy
```

Note that the *util* array saves the utilites.

2.3 Inferences

We output some sections of the trace -

```
iteration=111
(0,0,0):-1
(0,0,1):-1
(0,0,2):-1
```

```
.........
(0,3,0):-1
(0,3,1):-1
(0,3,2):-1
.........
(4,1,1):SHOOT=[-116.566]
(4,1,2):SHOOT=[-111.788]
(4,2,0):RECHARGE=[-110.671]
(4,2,1):SHOOT=[-105.756]
(4,2,2):SHOOT=[-100.842]
(4,3,0):RECHARGE=[-99.829]
(4,3,1):SHOOT=[-94.777]
(4,3,2):SHOOT=[-89.724]
```

Here is the last piece of the file, suggesting that the algorithm has run for **111** iterations.

Also, doing a word count on the actions -

$$\begin{aligned} \text{SHOOT} &-> 2688 \\ \text{DODGE} &-> 896 \\ \text{RECHARGE} &-> 1792 \end{aligned}$$

suggests that the action SHOOT is taken more commonly followed by RECHARGE and then DODGE. (3:1:2) ratio.

3 Task 2

3.1 Part-1

Upon decreasing the step cost exclusively for SHOOT, we see that the number of iterations decreases. This is due to the fact that as SHOOT is more probable to steer the game towards completion, we say that having less step cost of SHOOT finishes the enemy earlier.

3.2 Part-2

We see that the game lasts only 5 iterations. This can be attributed to the fact that since discount is lesser, the effect of each further action is decreased on utility. As γ becomes closer to zero, γ^2,γ^3,γ^4· and so on have their less contribution. Thus it is safe to conclude that in long term, the agent becomes myopic only learn about actions with an immediate reward.

3.3 Part-3

Upon further decreasing δ, we see that the agent takes once again more iterations to reach at the reward. This can be explained as we are increasing the level of accuracy level we want in order to combat the looseness due to decrease in γ. It continues until the reward is close, thus countering the effect of γ decrease.