Assignment 4: Discrete Markov Decision Process

GROUP 43: MAKSYM LYTOVKA (S3705609), IVAN BANNY (S3647951)

Q 3-b:

After adding multiple goals agent was planning towards the goal with a higher reward. If two goals with equal rewards were placed, the agent was moving to the nearest one. On the other hand, if one goal weighed more and the travelling expenses were lower than the difference, the agent selected that goal as a target and moved to it.

Interesting to note, that our approach to the environment description has one problem: if one goal with a lower priority is on the path to the other goal, the agent will terminate at the 1st one (unintendedly). Consider the following example:

```
# # # # # #
# *   A 1 #
# a   #   #
# # # #   #
#   # #   #
#   # #   #
#       #
#   # # B #
# b # # 3 #
# # # # # #
```

In that case, the agent will end up at terminal 1, even though it would be better to go on and reach 3. It's not a problem of the MDP approach, but rather the way of defining the environment in the code.

Q 4-a:

As we have 15 keys and doors, there are 15 keys. We may possess or not the certain key. Therefore, the number of unique states is:

$100*100*2^{15} = 3.2768 * 10^8$ (a lot)

Q 4-b:

We would need

$3.2768 * 10^8 * 32 = 1048576 * 10^4$ bits, which is 1.31 Gigabytes (GTA Vice City weighs 1.5 GB)

Q 4-c:

If we are using value iteration, then we store a value for every state. We update every value in the table until we reach the convergence. That means, that the time complexity of the algorithm is the size of the table times number of iterations we need. Empirically, around 10-15 iterations are needed to reach the convergence with the theta equal to 0.001. Hereby, the number of iterations is $3.2768 * 10^8 * 15 = 4.9 * 10^9$. If we approximate, an average computer executes around 10^9 actions per second. Therefore, the program would be solved in 5 seconds. However, considering the amount of virtual memory this program needs, execution may take up to 15-20 seconds.

Q 4-d:

The curse of dimensionality is a fact, that the number of states grows exponentially. In our problem definition, the main cause of the exponential growth is the number of keys. That's because as we add one key, the number of states doubles (each state may have that key or not). Therefore, the number of states overall is the number of free cells, multiplied by the two to the power of the number of keys. Hereby, exponential growth. If we were to add another property to each state (e.g. whether we have been to this place before or not) or another item to the inventory (e.g. the bonus coin), we would need to multiply the number of states by two.

Q 5-a:

The depth of the shortest path would be 21 (as we need to take 20 moves, which leads to 20 states + the initial state). Knowing, that the branching factor is 4, we may compute, that such a tree has $4^{21} - 1$ states. Moreover, as we use iterative deepening, we consider the same nodes several times. Hereby, the formula for the number of states visited (and, therefore, for the time complexity) is

$$n\_states = \Sigma^{depth}_{n=1} \ (branch\_factor^n - 1)$$

Where *depth* is the minimum depth, at which the solution may be found (21, in our case), and *branch_factor* is the number of actions you may take at each step (4, in our case).

If we substitute the values, we get more than $5.864*10^{12}$ (again, a lot). To get the operation time, we need to multiply this number by the constant time of one operation of the computer you are running the algorithm on.

Q 5-b:

If we take the constant of the operation time as 10^{-9} for our computer, we need 5864 seconds (1.63 hours) to execute the iterative deepening algorithm. Meanwhile, the DP algorithm was computed in a matter of seconds. Hereby, the DP algorithm is empirically much faster.

Q 5-c:

DP algorithm stores the solution as a table of values and the tree/graph search as a vector of states. The benefit of the DP representation is that you see the picture overall and you may compute the cost and profit of any trace you like. On the other hand, the benefit of the vector is that you don't need to perform any additional computations to follow the answer. Moreover, you can straightaway get the number of steps you need to make until you reach the goal (the length of the vector), meanwhile, in the table, you need to go through the states (which takes additional computational resources) and count the number of actions.