05/16/2016

Report on:

Autonomous Treasure Hunt

Developed by Ramin Abbaszadeh Tavassoli

SE733 Spring 2016 Final Project

Professor, Christos Cassandras Ph.D.

Abstract:

Traversing across a terrain, object recognition among myriad of other fascinating systems problems can be modelled by considering such systems as a series of states comprising a grid. Doing so facilitates describing the behavior of such a system, whether discrete or non-discrete, through a continuum of actions taken by an agent from one square to the next. This methodology is most conducive to Markovian systems, where the probability of transitioning from one state to its neighbor only depends on the current and the subsequent states and therefore is independent of the course of actions undertaken in the past. The system modelled in this project is in fact Markovian and we will use the Markov Decision Process paradigm to solve it. Markov Decision Process or MDP uses dynamic programming to optimize parameters of a Markov model. This project is a simple illustration of the power of MDP framework to determine optimal decisions.

Introduction:

This project entertains the idea of a robot moving about the grid toward a treasure occupied square while avoiding the blazing squares torched by fire. However, the project is designed for the user to determine the grid size, rock and fire count and two other parameters titled the *living reward* and *discount rate*. The project imposed limits on these parameters for reasons discussed in the next section. The grid dimension must be in range of 5 to 10, the rock and fire sum may not exceed dimension, the living reward must be set to a negative value and the discount rate must be between 0 and 1. In case any of the limitations are violated, the user is asked to revise selection. After the parameters are legally assigned, the user clicks on the green button titled “Proceed to Placement!” to start populating the game board.

Initially a clear grid appears to the left of the control window. The user then proceeds to locate the three types of items there is namely the treasure, the rocks and the fires by clicking on the board. The first click on the board environment fills respective square green to mark the treasure. The user subsequently clicks away on new squares to position rocks distinguished by black color. When the number of rocks is exhausted, the next click to the last are associated with fire placement identified by red color. It is important that no grid is filled twice and if a mistake is made and the situation arises, the user must start over.

When fire count is exhausted, further clicking on the squares does not change the grid dynamics. There are two actions that may be taken at this stage. First is to press the close button in the top right hand corner to clear the board and start over with fresh input. Second is to press the space bar and determine the optimal policy for each viable state in the grid (non-viable states are squares occupied by rocks). Each state is comprised of a value and a direction indicating which direction is the optimal choice for our robot. The value depicted on the board is essential to determining the direction and its meaning and derivation is covered in the next section.

Finally the job is done. At this point, the user has access to the optimal policy for every state on the grid. At this stage, the user may clear the board by closing it and input another scenario, or he may exit the program by pressing the “Exit” button on the control panel.

Model:

First we give a cursory description of the model and then analyze its components. As we mentioned in the previous section, we have a grid with an autonomous vehicle traversing it. It is first essential to give meaning to the items in our agent's path: the treasure and the fire states are all exit states meaning that the only action available at those states is to leave the grid. This is an implicit modeling decision since the goal is to get to the treasure optimally. After the treasure is obtained, there is no reason to study the grid any further. Also, the fire states destroy the agent and remove it from the grid.

If the agent runs into a rock, it remains in the same square. Besides the rocks the user gets to place on the map, there is a wall (fictitious in the rendered game board but programmed in the code) surrounding the entire board. If the agent transitions into a wall square, it is forced to go back to the state it transitioned from. Therefore rocks and walls are interchangeable.

What makes this problem interested is the concept of stochastic movement. If the movement was deterministic, then the problem would be much easier and the solution be visible to the naked eye. However, this problem considers movement to be stochastic. By stochastic movement, we mean if the agent decides to go north, there is 80% chance of accomplishing that goal and a 10% a piece chance of veering to the east or the west.

The agent accumulates rewards at each step of the process. The reward for landing in the treasure square is 1, while the reward for ending up in the fire is -1 and fixed. To incentivize the agent to move toward the treasure, we need a living reward of negative value. This quantity encourages the agent to seek an exit state (preferably the treasure state) to stop accruing this negative living reward at every time step (the model assumes that the agent moves a square per time step). Why do we mean by "preferably the treasure state?" Imagine if the living reward is set to -2. In this case, the cost of living is so high that the robot is better off walking right into the fire and collecting the -1 reward than continuing to live. This is why we need to set the living reward to a small quantity in the range of -1 to 0. We also need a discount factor in the range of 0 to 1 to lower the value of future rewards and make future gains less attractive. The discount factor is denoted by γ and is used primary to avoid collecting infinite rewards in the long run.

Finally, this system is Markovian since the reward collected or state landed at the next time step is independent of the history of the agent and only dependent on its current state. Therefore, we can proceed to solve the model using Markov Decision Processing (MDP).

MDP Formulation:

We define the MDP by the following properties:

A) States: $s \in S$
B) Actions $a \in A$
C) Transition Function: $T(s, a, s')$
D) Reward Function: $R(s, a, s')$

States are all the squares on the grid excluding the squares occupied by rocks. Feasible actions are a subset of actions that the agent may undertake. In this problem, regardless of current state, all 4 actions of north, south, west and east are admissible. $T(s,a,s')$ is the probability of landing at s' after taking action a at state s. $R(s,a,s')$ is the reward collected at s' after taking action a at state s.

The MDP is constructed with on goal in mind, finding the optimal policy π^* from start to finish. A policy prescribes a set of actions for all states of the system and an optimal policy prescribes the best action to take at each state. So no matter where the robot is starting, it has a sequence of actions to follow to maximize its expected sum of rewards, given that he has the optimal policy.

To solve the MDP problem we need the following two quantities:

A) $V^*(s)$: expected utility of starting in s and acting optimally.
B) $Q^*(s,a)$: expected utility starting in s and having taken action a.
C) $\pi^*(s)$: optimal action at state s

$$V^*(s) = \max_a Q^*(s,a) \qquad (i)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V(s')] \qquad (ii)$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V(s')] \qquad (iii)$$

The equations above lay the groundwork for the bellman equation presented shortly. Let us first analyze these. By (i), the expected sum of rewards starting at s is one of four Q values each of which computed for one of four actions. By (ii), Q values are computed. For each action a, there exists 3 next states (s'). Let us consider a = North. If action 'a' has occurred, there are 3 places the agent may end up: 80% north, 10% east, and 10% west denoted by $T(s,a,s')$. $R(s,a,s')$ is the reward collected at s'. In our project, $R(s,a,s')$ = living reward. The action 'a' that yields the highest value $Q^*(\mathrm{s},\mathrm{a})$ is used to calculate $V^*(s)$.

We define $V_k(s)$ to be the vector representing the optimal value of s at every step, if the game ends in k more time steps. Initializing $V_0(s) = 0$, we have the Bellman equation:

$$V_{K+1}(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_K(s')] \qquad (iv)$$

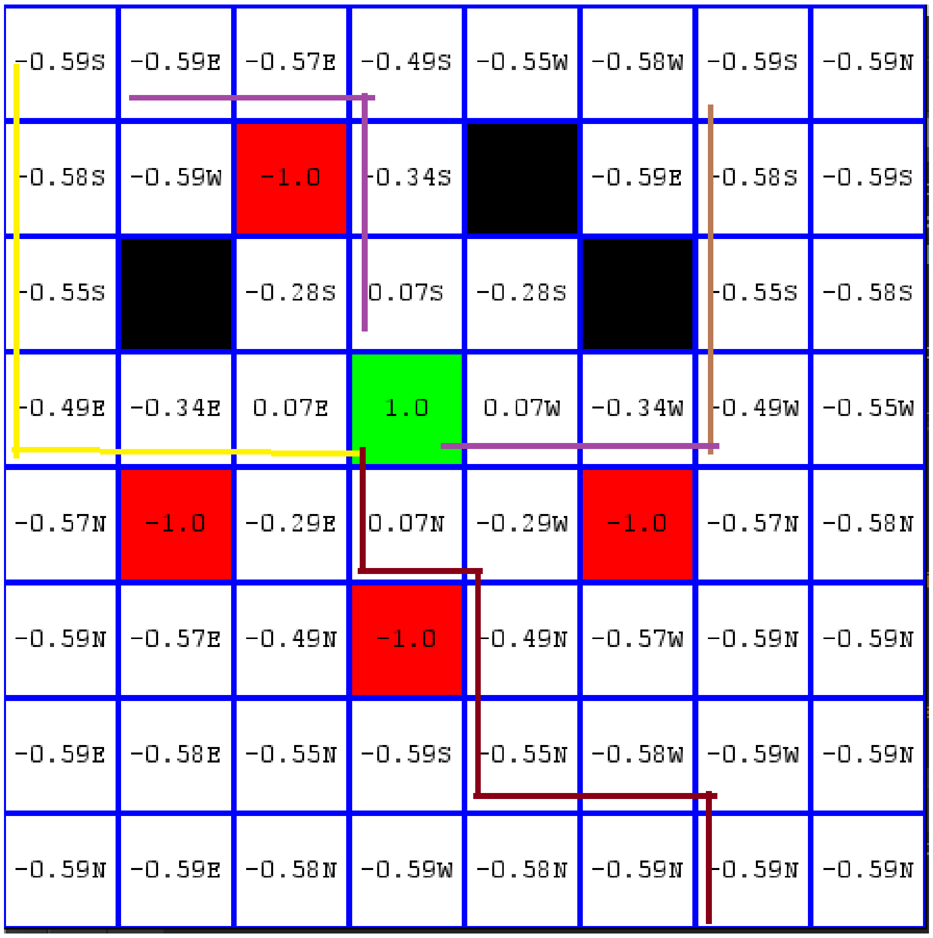
The program runs this code for k = 50 at which point the values converge for grids of relatively low dimensions such as ours. I have checked many examples of all dimensions and in all cases, state values converge at k = 50. This recursive relation is fully programmed in our project and all computations are checked multiple times to ensure accuracy.

This program is coded in python using its two major graphical interface modules titled "tkinter" and "pygame." The program codes the bellman equation on a board including the aforementioned fictitious walls on the perimeter of the board, thus increasing dimensions by 2 on both sides.

Results:

This is a sample constructed board where the black boxes are rocks, red boxes are fires and the sole green box is the treasure. Our goal throughout the project was to produce an output in shape of a board with directions printed on and we achieved that objective. The marks on the grid indicate sample paths, starting at initial states and optimizing the expected sum of utilities collected on their way toward the treasure. Each value is the state value, denoted by $V^*(s)$, which is the expected sum of rewards if the agent acts optimally from then on. The negative values arise from the relatively high |living reward| imposed (yes, |-0.3| is a high living reward, but since we are only interested in the optimal policy, it does not matter to us).

Figure 1. Sample Board 8x8, Living Reward = -0.3

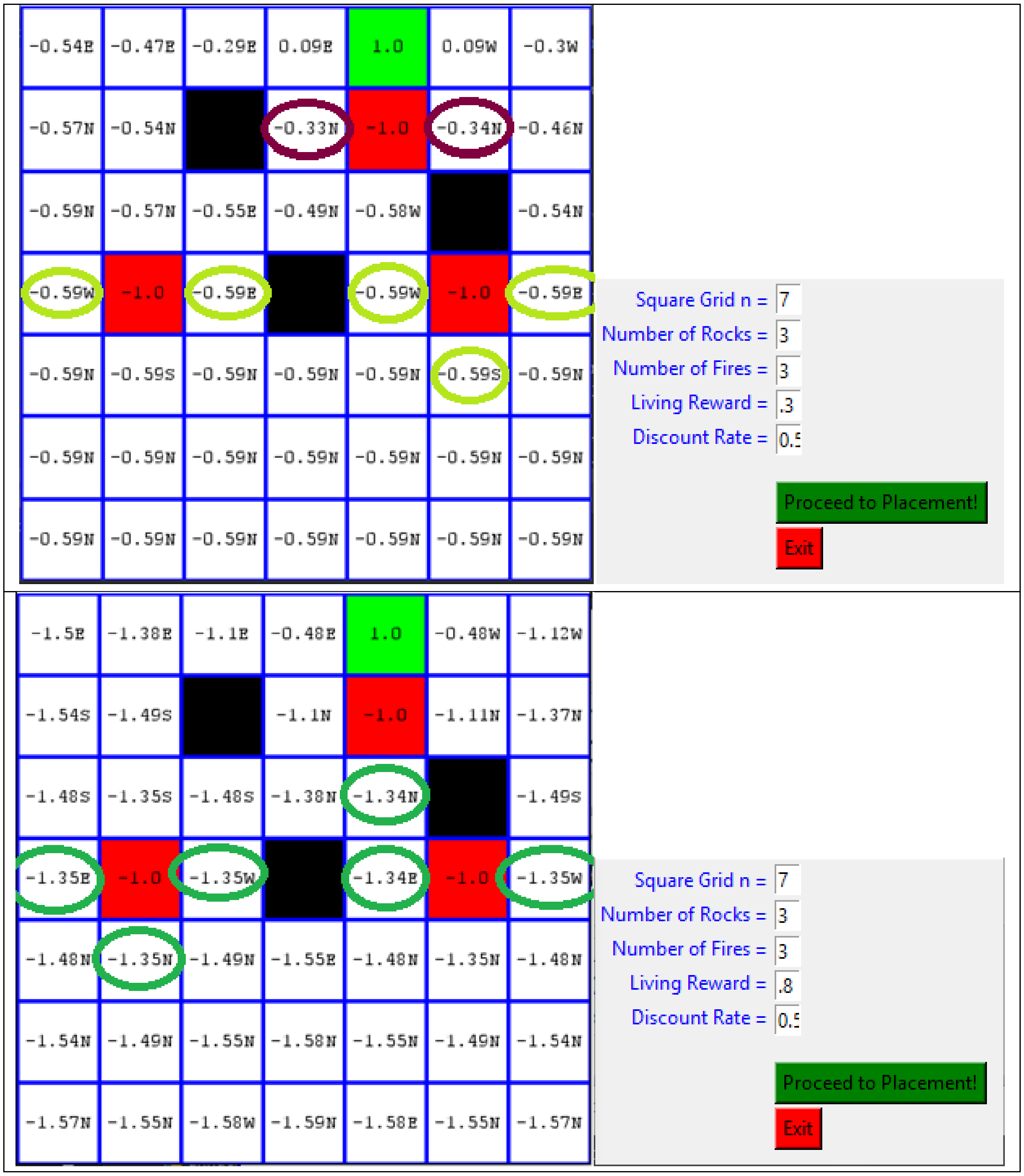


-0.59S	-0.59E	-0.57E	-0.49S	-0.55W	-0.58W	-0.59S	-0.59N
-0.58S	-0.59W	-1.0	-0.34S	⬛	-0.59E	-0.58S	-0.59S
-0.55S	⬛	-0.28S	0.07S	-0.28S	⬛	-0.55S	-0.58S
-0.49E	-0.34E	0.07E	1.0	0.07W	-0.34W	-0.49W	-0.55W
-0.57N	-1.0	-0.29E	0.07N	-0.29W	-1.0	-0.57N	-0.58N
-0.59N	-0.57E	-0.49N	-1.0	-0.49N	-0.57W	-0.59N	-0.59N
-0.59E	-0.58E	-0.55N	-0.59S	-0.55N	-0.58W	-0.59W	-0.59N
-0.59N	-0.59E	-0.58N	-0.59W	-0.58N	-0.59N	-0.59N	-0.59N

In figure.2, we observe the effect living reward and discounting. The two scenarios are identical in all parameters except their living rewards differ by 0.5. Let us consider the states circled in lime green in the top board and those circled in green in the bottom board. The top board has living reward -0.3 and under this condition, safety is emphasized over risk in the optimal policy. When there is a fire on left and a wall on the right of a state, the state chooses to run into the wall hoping to veer off to the sides with probability 10%. This is the least risky decision of all. Now let us consider the bottom board. The circled states on the bottom board of figure.1 are all suicidal. They are all jumping into the fire because the cost of living is too negative. Therefore it is very important for the living reward to be realistic and proportional to terminal states.

Also in the top figure in figure.2, two states are circled in purple and their optimal decision is to take a higher risk than those circled in lime green. This is an effect of discounting. As there are more steps to the treasure, the allure of the treasure drops as agents collect less utility on their way to the treasure than those who live closer to the treasure.

Figure 2. - 0.30 vs. - 0.80 Living Reward



-0.54E	-0.47E	-0.29E	0.09E	1.0	0.09W	-0.3W
-0.57N	-0.54N		-0.33N	-1.0	-0.34N	-0.46N
-0.59N	-0.57N	-0.55E	-0.49N	-0.58W		-0.54N
-0.59W	-1.0	-0.59E		-0.59W	-1.0	-0.59E
-0.59N	-0.59S	-0.59N	-0.59N	-0.59N	-0.59S	-0.59N
-0.59N	-0.59N	-0.59N	-0.59N	-0.59N	-0.59N	-0.59N
-0.59N	-0.59N	-0.59N	-0.59N	-0.59N	-0.59N	-0.59N

Square Grid n = 7
Number of Rocks = 3
Number of Fires = 3
Living Reward = .3
Discount Rate = 0.5

-1.5E	-1.38E	-1.1E	-0.48E	1.0	-0.48W	-1.12W
-1.54S	-1.49S		-1.1N	-1.0	-1.11N	-1.37N
-1.48S	-1.35S	-1.48S	-1.38N	-1.34N		-1.49S
-1.35E	-1.0	-1.35W		-1.34E	-1.0	-1.35W
-1.48N	-1.35N	-1.49N	-1.55E	-1.48N	-1.35N	-1.48N
-1.54N	-1.49N	-1.55N	-1.58N	-1.55N	-1.49N	-1.54N
-1.57N	-1.55N	-1.58W	-1.59N	-1.58E	-1.55N	-1.57N

Square Grid n = 7
Number of Rocks = 3
Number of Fires = 3
Living Reward = .8
Discount Rate = 0.5

Conclusion:

This project was absolutely fascinating and even though failures were not few, the code was fully debugged and the output massaged into its sharp form. One of the goals of the project for me was to create an environment that could be altered to serve different needs. There are many systems that operate in the neutral, punishment, reward dynamic of the treasure hunt game we entertained here. Also the grid size adjustment is a perk that allows users to model more concretely.