Title of my document

Flavio R. de A. F. Mello

February 20, 2019

1 Domain and Task

This work aims to study the implementation of a Reinforcement Learning algorithm in a multi-agent scenario. For this study, a simple task was devised: agents are randomly placed in the playing field and their goal is to reach a particular desired arrangement. The environment is discrete, static, and fully observable by all agents. States are deterministic and episodic in nature.

1.1 Playing Field Arrangement

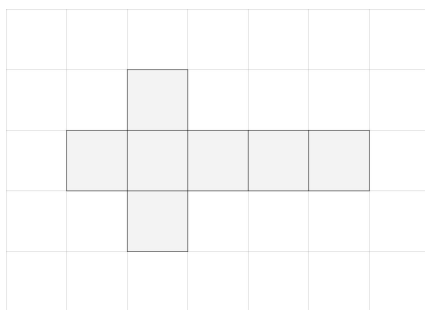The field consists of 7 locations arranged in a cross-shaped form.

Figure 1: Visual representation of the playing field

1.2 Playing Rules

Each action consists of a move to an adjacent location. Moves are only allowed in the cardinal directions, no diagonal moves are allowed. Multiple agents may not occupy a same location at the same time, therefore moves are only allowed if the destination is an empty location. Alternatively, agents may elect to stay in the same

place. Thus, there are 5 possible actions: move up, move right, move down, move left, and stay in the same place. It is important to note that the field topology plays a role in determining which actions are possible for a given agent in a given state. It is perfectly possible for agents' action options be reduced to staying in place for a specific state. Agents will take turns performing actions, starting with agent 1.

1.3 Task Goal

The goal is to have the agents reach the rightmost section of the field in ordered section. The agent with the smallest id number is to be at the rightmost square, followed by the other agents successively. The rightmost section is a narrow corridor, this setup allows for situations where a given agent may block the passage of the other. This characteristic of the playing field was intentionally designed to require for cooperation between agents. If they directly move to their final position, one agent may be blocking the another agent's path and thus fail to reach the final desired state.

2 State Transition and Reward Functions

For implementation purposes, the states are encoded into a single string of 7 characters. Agents are assigned ids in the form of positive numbers, which are used to indicate their position in the field. the value '0' is reserved for empty spaces. Each of the 7 locations is assigned an id from 0 through 6, which is used to map the location with the index in the encoded string. Figures 2 and 3 show the location indexes and an example of the mapping between a state string and its equivalent board arrangement.

The number of possible states is determined by the number of agents in the experiment and is equal to the number of unique permutations of agents and locations. It is calculated with the following formula:

$$N_{states} = \frac{m!}{(m-n)!} \tag{1}$$

For an experiment with 1, 2 or 3 agents there are 7, 42 and 210 possible states, respectively. Given the somewhat low number of possible states, this study opted to implement the transition matrix as a direct state to state mapping, resulting in a square NxN matrix, N being the number of possible states. This trades a higher memory footprint for ease of implementation and interpretation. This particular implementation has a memory footprint proportional to N^2, in problems with a larger state space, one might opt to implement the transition table based on possible
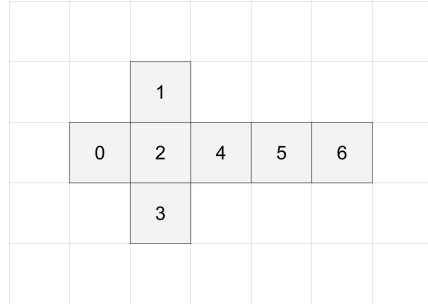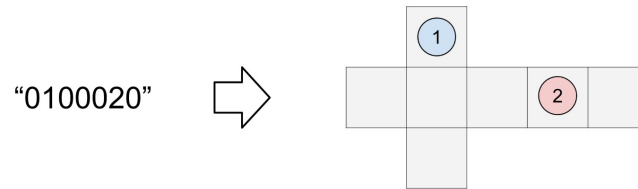
Figure 2: Playing field with location ids

Figure 3: State string with equivalent field representation

actions (Up, Right, Down, Left, and Stay) given that the matrix would increase to the order of N. Another, even more memory efficient implementation would be to store only the possible transitions in a list or dictionary. Given the sparse nature of the transition matrix, this option would greatly diminish the amount of memory necessary for running the learning process, at the expense of increased implementation complexity.

Considering each agents only moves itself, the state transitions will be different for each agent. In this particular implementation it was decided to keep a separate transition table for each agent. These tables are calculated at the beginning of the experiment. For each agent, the program iterates over all the possible states and checks which moves are possible for said agent.

The reward function grants agents 100 points for any action that transition them to the desired state, this includes the non-movement action of staying in the same place. Contrarily, agents are awarded negative 100 points for transitions that take them away from the desired state.

3   Learning Policy

The learning policy selected for this study is the ε-greedy policy. This policy relies on a dynamic value ε, such that $0 < \varepsilon < 1$, to determine an agent's strategy in a given state/time. The agent is to explore (i.e. take a random action) with ε probability, and exploit (i.e. take the action which has the highest expected return as per the Q-matrix) with (1-ε) probability. The learning process starts with a predetermined ε value, which is gradually decreased after each learning episode. This means that when there is more uncertainty (i.e. at the beginning of the learning process, when not much is known regarding the environment), the agent is more prone to take random actions, exploring the possible state space and gathering feedback and "understanding" of the surrounding environment. Gradually, the more the agent knows regarding the environment, less often it takes random actions. This effectively has the result of directing the random exploration to regions surrounding the currently learned action policy. The selection of initial value and decay function for ε plays an important role in the performance of the learning process. If the decay is too slow, or the initial value is too high, the agent will keep exploring for longer possibly over exploring the problem space before converging to a policy, leading to longer training times. Conversely, if the decay is too fast (or the initial value is too low), the agent will converge to a policy faster, but this policy may be suboptimal as the problem space was underexplored. Naturally the optimal decay rate is dependent on the particular task being learned and the size of the state space. Striking this optimal balance is, therefore, an important step in the tuning process.

4   Graphical Representation & R Matrix

consequently, the reward function will also be different.