# Worksheet – Reinforcement Learning

## Overview

### Objectives:

1. Add State Assignment Logic
2. Add Action Decision Making
3. Calculate Rewards
4. Update Learning Matrix
5. Extend Learning Matrix
6. Add new States & Actions
7. Save and Load Learning Matrix

## Preliminary

Primarily this tutorial work will require the use of a C++ IDE. Visual Studio is the recommended software to use although any C++ development environment should be sufficient.

The way this reinforcement learning works is through the use of an object called the learning matrix. This matrix stores the reward of a particular action against a state. The easiest way to imagine it is like a 2D grid.

|  |  |  |  |
| --- | --- | --- | --- |
|  | State 1 | State 2 | State 3 |
| Action 1 | 1 | 2 | 3 |
| Action 2 | 4 | 5 | 6 |
| Action 3 | 7 | 8 | 9 |

Each Actions and State is assigned a number, allowing us to use them like indices. For example with the grid above:

LM[State2, Action1] = 2

We can use the Learning Matrix to check which action has the highest reward for the state we are in a choose it. We can also update the Learning Matrix as we go with rewards. This way, the system learns which action to perform in certain states, without us manually designing it to do so.

## Getting Started

You can also download a zipped version of the project from MyLearningSpace called **Reinforcement Learning C++**.

If you chose to use a different IDE or compiler setup, then copy the below code into a file and work from that.

There are three items that form the core of this approach. A selection of states the AI can exist in, a selection of actions the AI can perform and a matrix that states how likely we are to perform said action, when in a certain state.

There are described by the following lines of code:



The game state and actions are described as ENUMs, just for easier reading. The Learning Matrix is managed as a 2D array, with all its starting values initialized to zero.

## Run the program

The program represents a closed water pipe system and the AI is in charge of managing the pressure of water in the system. Pressure in the system is lost over time so the AI has to learn to top up the system when it is considered low, remove water when its considered too high and maintain the pressure at a good equilibrium.

## Finish the TODO Sections

### Add Describe State Logic

To start with, we will need to have some way of identifying which state the system is currently in. There are three states as described above, Stable, Too low & too high.

Create the logic that will take the *system\_pressure* as an argument and return the appropriate state. The range for each state is up to you but for example, when *system\_pressure* is below 25, the state could be set to too low.

The best way to do this is with a function, returning a value of ENUM State. E.g.



Make sure to do this for ALL the available states.

### Add Action Choice Logic

Once we have the current state described, we will then need to choose the appropriate action. The best way to do this is to look through our Learning Matrix using our current state and return the action index with the highest value.



Whilst this is straightforward, it also has a problem. If we only look for actions that have the highest value, as soon as one actions is tried that has even a tiny positive impact, it will ignore the other actions. Therefore you may want to implement a random chance into the code. Say every time this logic is executed, we may want the system to have a 5-10% chance to try a different action.

You can use the rand() function to approach this. Create a random value between 0 and 99. If the value is greater than 10, select the action with the highest value. Otherwise, choose another action at random.



Note: There is an std:: random number generator that you may want to investigate.

### Make the AI perform the Action

Once an action has been chosen, we will want to actually perform the action. Use a switch statement to do this:



### Calculate Reward

Now that the system has performed an action, lets calculate the reward. Reward shaping is very important and can affect how the AI learns.

I will let you decide how you will calculate the reward value; generally you want to have a positive reward for behaviour you want to encourage, and a negative reward for behaviour you want to discourage.

I would suggest giving a positive reward if the pressure is within the ‘STABLE’ limits and negative if it is outside. You can even scale this by how far outside the limits the current pressure is.

### Update the Learning Matrix

Now we have a reward we need to update the Learning Matrix. Using the State you calculated in the first step and the chosen action, add the reward to the correct item in the Learning Matrix.

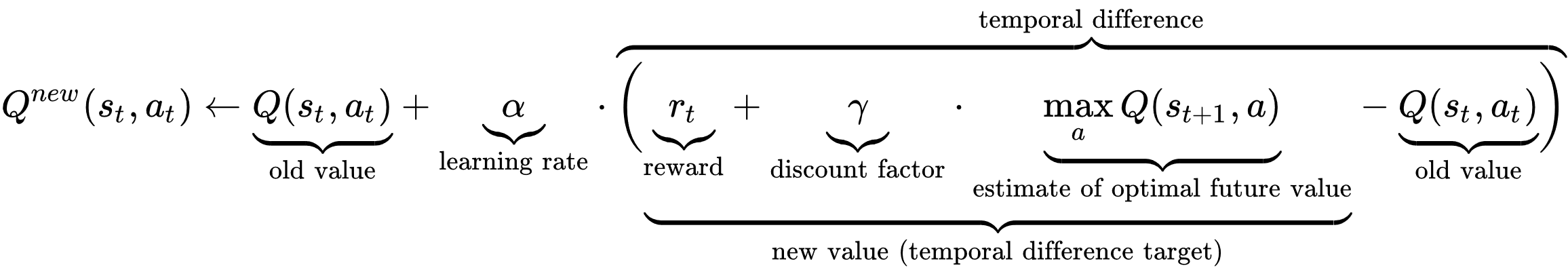


I would recommend at this point that you finish the last TODO in the code and have it output the chosen action.

Now run the program and see if it performs as you might expect. Does it keep the pressure within the ‘Stable’ range?

## Exercises

1. Edit the updating of the Learning Matrix to follow something along the lines of:



α – Represents your learning rate. I would set this to a low value such as 0.1

γ – Represents a concept called Discounting. The highest this number is, the more the AI will focus on potential future rewards. For example, if this value is set to 0, it will not think about future rewarding at all.

maxQ – This is the maximum value that this State/Action combination can reach. This prevents one action becoming too dominant and allows the AI to explore the different possibilities more. Alternatively you can have maxQ represent the current highest valued action for that state.

When using this this algorithm, called a Q-Learning algorithm, the Learning Matrix is often abbreviated to **Q-Matrix**.

1. Add a new two new states and actions to the system. One for very low pressure (lower than low pressure) and another for very high pressure (higher than high pressure) and appropriate actions to ‘stabilise’ the system.

Re-train the program and see if it learns how to react them it reaches these states. You may want to add keyboard input to force it into different states.

1. Get the program to write the Q-Matrix to a text file and either the option (or separate program that does not Q-Matrix updating) to load this file and run the system based off it.