## Introduction

Machine learning is a term that dates back to the 1950’s, said to be coined by an American computer scientist called Arthur Samuel. In machine learning, mathematical algorithms are used to analyse data in order to make predictions about the future. There are several types of machine learning, one being reinforcement learning.

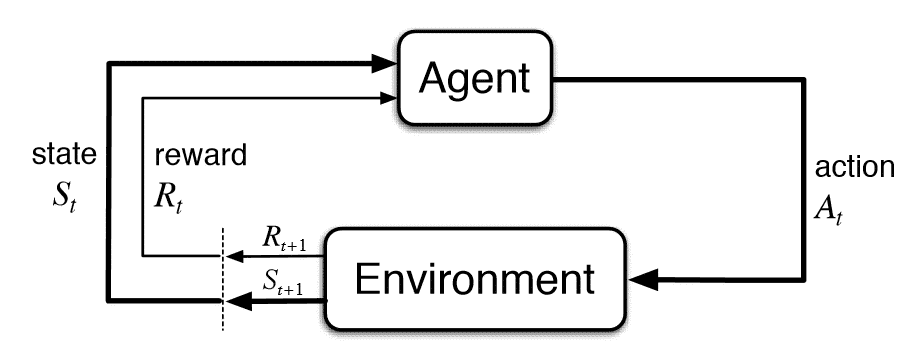
Reinforcement learning is a form of machine learning that is based on the idea of rewarding good behaviour and penalize bad behaviour. It is a form of machine learning that lends itself well to solving problems in environments were all possible states and actions can be specified, making it ideal in gaming applications and less so in the area of self-driving cars. The challenge when implementing reinforcement learning to solve a problem can be to define what good and bad behaviour is. A reinforcement system optimizes for the highest reward, it is the user’s responsibility to make sure that the highest reward implies that the wanted solution is found.

In this project, reinforcement learning was implemented to optimize the movements of virtual robots within a warehouse environment. The optimization problem consisted of enabling multiple robots to find the quickest path from a pick-up point to drop point, without colliding with each other or blocks in the environment.

The purpose of placing automated robots in a warehouse environment is to relieve humans of tasks that are monotonous and/or potentially physically straining. These types of robots are becoming more and more common in warehouses around the world, but they are still very expensive relative to their performance. To really offset the investment cost, these robots need to be able to perform their tasks at a greater speed and accuracy, why further research in machine learning in necessary.

## Q-learning

Reinforcement learning can visually be represented as follows:



A learning agent is at any given time in a state and when the agent takes an action it will reach a new state and collect a reward, the reward is a value that is specified by the program designer. This process is repeated over and over until the agent reaches a terminal state. When the agent reaches a terminal state, the simulation will be terminated, and a new simulation will be initiated.

One way of implementing reinforcement learning is to use a method called Q-learning. In Q-learning, the agent chooses the next action by referring to a state-action table, called a Q-table, which indicates what the optimal move is when in a particular state. The Q-table is updated according to a chosen policy after every move and eventually the agent can rely on it to find the optimal path to the goal state.

During training, the Q-table is updated according to the Bellman equation given below.

Legend:

|  |  |
| --- | --- |
|  | **State at time t.** The agent will move within a set environment consisting of discrete states. Each state will have a given reward value associated with it. |
|  | **Action at time t.** The agent has a limited set of possible moves available to it. Each action will take the agent into a new state. |
|  | **Learning rate.** The learning rate determines how much the next Q-value for a given state should depend on previous values. A learning rate of 1 would imply that the agent is not taking previous experience into account at all. |
|  | **Reward function.** This function maps every state in the environment to a value. These values must be chosen in a way that enables the robots to find an optimal path according to the … that has been set. |
|  | **Argument that give maximum Q-value.** This function returns the Q-value at , given that the optimal action was taken. This means that a state might be more valuable if it is in proximity to a high valued state. In other words, we might accept short term losses in exchange for gains later on. |
| **γ** | **Discount factor.** The discount factor weighs the importance of future rewards, a low-valued γ means that we care more about short term rewards than long term rewards. The discount factor is used because we generally don’t expect our model to be ideal. If we would set γ equal to 1, we might end up with a long and complex solution and if we can’t trust that our model is ideal, we can’t trust that the solution is either. |

## Optimization for 1-2 robots

When optimizing for 1-2 robots in a 10 x 10 grid, traditional Q-learning is effective enough to obtain an acceptable computation speed.

## Deep Q-learning

In this project, the goal was to optimize the movements of up to 4 robots in a 10x10-squared environment. Implementing Q-learning in simulations dealing with more than 2 robots proved to be problematic since the simulations would require gigantic Q-tables. When running simulations in instances with 1-2 robots, the combined positions of the robots would be considered a state and the same principle was applied to actions. This meant that a Q-table for *n* robots taking *a* possible actions, moving in an *m* x *m* environment would have entries (assuming the environment is shelf-free). In the instance of the end goal of the project, the Q-table would have 62.5 billion states. DUBBELKOLLA. Optimizing the movements of 4 robots this way would require a lot more computer power than the average computer could handle. To get around this problem, the simulations were run with the aid of deep neural networks instead of updating a Q-table after every simulation.