Using Reinforcement Learning to deliver safer, more efficient and predictable air traffic control

Oskar Aaron Oramus[*]

Abstract
Air traffic control systems are a vital part of the aviation industry. They provide flight patterns to aircraft and control towers which contain crucial information to ensure safe and efficient air traffic. These systems are highly complex and require a high level of expertise to operate and maintain. Safety is a top priority, as any error can have serious consequences. The scalable nature of the air transport industry creates a need for a solution that can match the long term growth. As the number of passengers and aircraft increases rapidly, an automated system is needed to handle the dense air traffic. This paper investigates a possible solution for aircraft control using a multi-agent reinforcement learning system that is capable of safely managing air traffic in a stochastic environment. The proposed solution is decentralised, i.e. each unit acts on its own, which has the advantage of greater scalability.

[*] oao206@exeter.ac.uk

1. Introduction

When we think of learning, we naturally envision acquiring new knowledge, learning skills through study, experience or instruction. We collect data and make decisions based on the available information. When we take an action, we observe the environment and see how it changes. These observations and stimuli are encoded and memorised for future use. This mechanism is called the retrospective memory and it is used to make decisions based on past experience. In addition, there is prospective memory, which is responsible for assessing the quality of the current decision and its impact on the future [Zentall 2010]. This is exactly how reinforcement learning works: we figure out how to link states and actions in such a way that our reward signal is maximised [Sutton and Barto 2018, 1]. Reward-seeking behaviour is a natural way in which animals behave and anticipate rewards [Arias-Carrión and Pöppel 2007]. It states that the higher the expected reward for an action, the higher the desire or motivation for the reward. This can be modelled mathematically with a Markov decision process, and the desire for an action is represented by a probability function or matrix $\mathbb{P}$. The process of adjusting probabilities is the learning process.

An agent is a learner in our simulation, that is tasked with finding the optimal behaviour for a given environment. The learning process for an agent is a simple loop as shown in Fig.1, it takes an action and observes the change in environment, an agent then receives a reward based on its actions and adjust its behaviour.

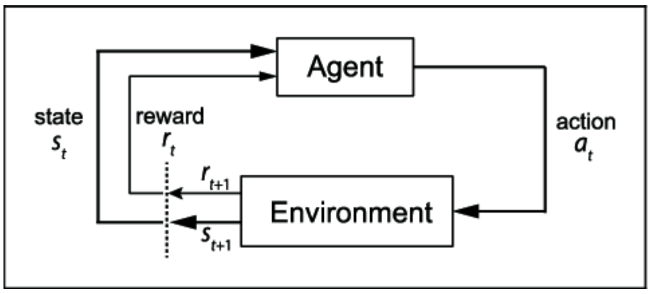

Fig. 1. Learning cycle of an agent $[Galatzer - Levyet\ al.\ 2018]$

2. Motivation

Reinforcement learning has gained popularity after the success of AlphaGo showed that it is possible to train a computer to play a game at a professional level [DeepMind 2022]. In 2016, AlphaGo beat the world champion in the game of Go, an extremely complex game that requires processing a lot of information that cannot be computationally handled by a min-max algorithm. Today, reinforcement learning is changing the world by powering recommendation systems [Zou et al. 2019], controlling smart power grids and training real-life robots to automate tedious tasks [Vilches 2016]. The aviation industry is constantly striving to become more efficient and accessible, but the ever-increasing demand for pilots makes this a challenge [Lutte 2018]. However, this problem is closely linked to flight planning in an ever-changing environment where reinforcement learning excels at.Goal is for our aircraft to land safely and efficiently at their destination [Bonsai 2017, Vilches 2016]. Reinforcement learning is a type of machine learning that fits naturally in simulated environments . Other types of machine learning require data that has already been created. They simply find patterns in that data and use them to predict data values based on past trends.

Another good candidate for this task is an Evolutionary Algorithm. The main difference is that the evolutionary algorithm is an inter-life algorithm, i.e. it has to die in order to evolve and adapt its behaviour, whereas reinforcement learning is a constantly evolving learning algorithm [Neelarghya 2021]. It is possible to combine both algorithms by having many agents learn and then cross over and mutate their solutions. A study in Philadelphia has found that a genetic algorithm finds a nearly optimal solution in around 20 seconds, and after that it would slowly converge [Alliot et al. 1993, 341-342]. However, the study used an approximated A* solution, i.e. the results were not compared with the optimal solution. The algorithm also did not contain enough iteration steps because it was computationally infeasible, so in some cases the separation was not triggered, which meant that a conflict went unnoticed by the system.

Another study at the University of Singapore found that using a Deep Reinforcement Learning solution with Deep Q Networks resolved conflicts 100% of the time and only took about 1000 episodes to quickly converge to a solution. It uses a discrete action space composed of different steering angles for left/right $\langle 12, 10, 8, 6, 0 \rangle$ to account for passenger comfort and the physical limitations of aircraft control. The study has shown that the use of reinforcement learning can provide promising results and is able to maintain the minimum distance between aircraft at all times. However, performance decreases when the distance between aircraft falls below a certain limit. Due to the limitations of the action space, it is sometimes impossible to avoid a collision because of the control angle limitation and speed of the aircraft facing a particular situation [Mukherjee et al. 2022, 5-7].

In line with the results of another study conducted by Beihang University, it turns out that the use of a "MADDPG" algorithm developed by OpenAI [Lowe et al. 2017] is particularly suitable for situations with high traffic volumes [Lai et al. 2021]. The algorithm is designed to work in environments where multiple agents interact. The Multi-Agent Deep Deterministic Policy Gradient uses a centralised critic based on observations of all agents and actions. They found that in simulations where agents interact in some way, the average reward is much higher than other popular methods that use the same learning parameters. However, the main drawback was that the input space of Q grows linearly with the number of agents. The proposed solution uses centralised learning with decentralised execution. Using numerous different scenario configurations, the study is able to show that the experimental results are scalable even in dense traffic situations.

3. Markov Chains

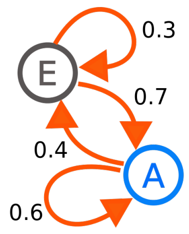

Fig. 2. Basic Markov Chain System
[mar 2022]

A Markov chain is a stochastic model that describes a sequence of possible events, where the probability of each transition depends on the current state. For example, in a blackjack game, our state space would be the cards we currently have and the dealer's hand. Our action space is hit, double down, stay or split, and the probabilities of the action transition would be different depending on what our current state is [mar 2022, Powell and Lehe 2022].

For a Markov state s and successor state s', state transition probability is defined by [Silver 2015a]

$$\rho_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s] \tag{1}$$

4. Markov Reward Process

The return G_t is the total discounted reward from time-step t [Silver 2015a].

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2}$$

We want to find a balance for an algorithm that is able to find a solution quickly, but is also able to explore to avoid possible local maxima. This is done with the help of a discount factor $\gamma \in [0, 1]$.

It is important for the learning process because we are not sure what the future will look like. Our environment could be constantly evolving so we cannot have a perfect plan for the next t steps, but we can make a rough plan and adjust it as needed. It is also used to avoid infinite returns in a cyclical Markov process. We do not want our model to keep repeating an action that earns it a quick reward. When set to 0, the agent acts greedily, ignoring all future states and focusing only on the present situation. This is very similar to the way animals instinctively make decisions, acting to obtain immediate rewards rather than focusing on long-term rewards [Silver 2015a]. With a value of 1, on the other hand, they would seek to maximise all future rewards. The usual starting value for γ is 0.1 or 0.01 [Bengio 2012].

4.1. Policy Function

A policy function $\pi(s)$ is a probabilistic mapping from state space S to action space A. It fully defines the behaviour of an agent. There are many ways to translate a policy into actions, for example we can always choose the most likely one, choose based on probability or ϵ-greedy, i.e. we choose the best action with probability $1-\epsilon$, but completely random with probability ϵ.

The goal of an MDP is to find a strategy that maximises the decision maker's reward in state s. This is not easy because the more parameters and actions we have, the more computationally intensive the problem becomes. This is called the curse of dimensionality, and it is well known that the search for an optimal control policy increases exponentially with the number of states and actions [Powell 2011, Vilches 2016].

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s] \tag{3}$$

4.2. Deterministic Policy

The policy will take exactly the same action given the same condition. In a simulation where all information can be determined (i.e. the environment is an MDP), there is always an optimal policy that is deterministic.

4.3. Stochastic Policy

Stochastic policy is helpful when we do not know everything in our simulation. In poker, for example, more and more information becomes known as the game progresses, so there is no guarantee of winning every time. In environments that change (non-Markovian environments), we need to use stochastic strategies to get better results. However, a stochastic strategy can become deterministic if we choose an action with the same formula every time. For example, we could always choose the highest probability [Silver et al. 2014]. In our context, this could mean that two drones with exactly the same starting position and the same target would take two different actions to reach the target.

5. Markov Decision Process

Markov decision processes (MDPs) are an extension of Markov chains. They provide a framework for modelling a decision process where outcomes may be partly random and partly chosen by the decision maker. This is very important for reinforcement learning as we want the agent to explore new potential solutions but at the same time be in control of their decisions. The key idea in MDP is that the future is independent of the past. In a chess game, for example, it does not matter how we got into that position, what matters is the current best move.

Markov Decision Process is a 5-tuple [mdp 2022].

$$(S, A, P_a, R_a, \gamma) \tag{4}$$

where: $P_a(s, s') = Pr(s_{t+1} = s'|s_t = s, a_t = a) =$ the probability that action a in state s at time t will lead to state s' at time $t+1$

5.1. State-Value Function

We need a way of our agent knowing how good it is it be in a current state. We can calculate this using a value function $v_\pi(s)$. The state-value function is our immediate reward plus a discounted value of successor state [Dittert 2020, Lee 2005].

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \tag{5}$$

We simply calculate expected G_t return from a given state at time t. Idea being that we want to maximize this over time.

5.2. Action-Value Function

Following from State-Value Function, we can derive the expected return if agent chooses action a using policy π [Dittert 2020].

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{6}$$

We want our agent to maximize both of the function values. Optimal value for state-value and action-value for all policies are expressed by:

$$v_*(s) = \max_\pi v_\pi(s) \tag{7}$$

$$q_*(s, a) = \max_\pi(s, a) \tag{8}$$

The final objective is to compute optimal q_* for our MDP, once that is reached we have reached the optimal policy for our environment and the agent will behave optimally with the given information.

5.3. Optimal Policy

Define partial ordering over policies

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s \tag{9}$$

For any MDP there exists an optimal policy π_*, that is better or equal to all other policies [Silver 2015a],

$$\pi_* \geq \pi, \forall \pi \tag{10}$$

It is possible to have multiple policies that all are optimal, but with MDP we are guaranteed to have at least one.

If our improvements stop (i.e. our values have converged)

$$q_\pi(s, \pi'(s)) = \max_{a\in A} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s) \tag{11}$$

then the Bellman optimality equation has been satisfied

$$v_\pi(s) = \max_{a\in A} q_\pi(s, a) \tag{12}$$

5.4. On Policy Learning

On-policies improve the same policy that chooses the agent's actions. They use a small probability of choosing a non-greedy action when making a decision to allow exploration. This is necessary because otherwise the agent may miss out on better outcomes. For example, if we have a game where we choose between two paths, one of which gives us +1 and the other has a 10% chance of giving us +25, the agent might not initially realise that the second path gives a better expected return (1 versus 2.5). This is modelled by introducing ”noise” into our decision process that randomly changes the action decision. We have a ϵ probability of choosing a random action and a $1 - \epsilon$ probability of choosing an optimal action. An example of an on-policy learning algorithm is SARSA [Hristov 2022].

5.5. Off Policy Learning

Off policies use a different policy to select actions than those they seek to improve. We have a behavioural policy b_π and a goal policy t_π. b_π is used to explore and create episodes. While t_π is used for function estimation and general improvement. Essentially, t_π uses the mistakes of b_π to learn, keeping track of the good actions. An example of an off-policy learning algorithm is Q-learning [Hristov 2022].

5.6. Bellman Optimality Equation

If the entire environment is known, we can solve the optimal action-value and state-value functions simply by dynamic programming. However, for most problems, we cannot explicitly represent the transition probability distributions. Therefore, a common solution is to create an environment and reward the agent for actions that bring it closer to the solution [Ng 2021].

6. Solving Bellman Optimality Equation

A common solution to solve the Bellman optimality equations is to use iterative processes that converge to an optimal strategy π_*. Training takes place within an episode that simply refers to a sequence of states, actions and rewards that end in a final state. This could be a game that is played until someone wins, the agent dies or the time limit runs out. Epoch is a certain number of episodes that we go through until the learning simulation is completed. After a certain number of epochs, we save the current strategy, plot the value function over time to assess the agent's performance and see if any adjustments are needed.

6.1. Policy Iteration

We start by initializing policy to some arbitrary policy π. Since our value function can give us a rating of the policy, all we need to do is make sure the value function increases.

First we evaluate current policy

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + ...|S_t = s] \tag{13}$$

We then calculate improved policy with one-step look-ahead to replace initial policy $\pi(s)$

$$\pi_{new}(s) = \max_{a \in A} \mathbb{E}\big[R(s,a,s') + \gamma V(s')\big] \tag{14}$$

We repeat these two steps until convergence to optimal value function.

6.2. Value Iteration

Value iteration is a way of solving for $\pi \approx \pi_*$. It starts at the "end" and works backwards [Poole and Mackworth 2018]. If we can improve a policy π using v_π to yield a better policy, π', we can then compute $v_{\pi'}$ to get π'' [Silver 2015b, Sutton and Barto 2018, 82-85]. If we repeat this process to ∞ we will monotonically be improving policies and value functions.

$$V_{k+1}(s) = \max_{a} \mathbb{E}\big[R_{t+1} + \gamma v_k(S_{t+1})|S_t = s, A_t = a\big] \tag{15}$$

Main differences between the two methods is that Policy Iteration is more complex to implement, but is cheaper to compute and requires fewer iterations to converge [Tokuc 2022].

6.3. Q-Learning

Q-Learning is an off policy method of solving the Bellman Optimality Equation that uses a Q-Table to maximize reward output [Banoula 2022, Violante 2019]. It uses a greedy policy to determine the next action, but will update Q values based on exploration. It is preferable when we do not care about agent's performance during the training process [Soemers 2018].

Algorithm 1 Q-Learning [of Computer Science and Sydney]

```
1: Create a table of size $S \times A$ with all values initialized to 0
2: for Each Episode do
3:     Initialize $S$
4:     Choose $A$ from $S$ using policy derived from $Q$
5:     Take action $A$, observe $R, S'$
6:     Update Q table using bellman equation (figure 16)
7:     $S \leftarrow s'$
8: end for
```

$$Q_{new}(s,a) = Q(s,a) + \alpha \Big[R(s,a) + \gamma \max_{a'} Q'(s',a') - Q(s,a) \Big] \tag{16}$$

where: $\alpha \in [0,1] =$ is learning rate of the agent

Main disadvantage of Q-Learning is that it only works in discrete and finite action spaces [Heberer 2019]. Another problem is that due to the table size it is only feasible in smaller problems, when our problem becomes much bigger and more complex the table will converge a lot slower and our memory usage will grow more rapidly.

Deep Q Network (DQN) is an attempt at fixing issues in Q-Learning, instead of using a Q-Table it uses a neural network that takes a state and approximates the Q-values for each action in that state [Loeber 2022].

6.4. SARSA

SARSA also known as $Q(s,a,r,s',a')$, is an on-policy algorithm very similar to Q-Learning. Main difference is that Q-Learning uses an optimal greedy policy when evaluating behaviour, whereas SARSA will use same policy that was used to generate the environment experience [Soemers 2018]. If we have an environment where failing is expensive (for example a robot falls off a cliff and is destroyed), in this scenario SARSA would be preferable as it would quickly learn that being close to a cliff is dangerous.

Algorithm 2 SARSA [of Computer Science and Sydney]

```
1: Initialize $Q(s,a)$ arbitrarily
2: for Each Episode do
3:     Initialize $S$
4:     Choose $A$ from $S$ using policy derived from $Q$
5:     for Each step of episode do
6:         Take action $A$, observe $R$, $S'$
7:         Choose $A'$ from $S'$ using policy derived from $Q$
8:         Update $Q(s,a)$ using equation from figure 17
9:         $S \leftarrow S'$
10:        $A \leftarrow A'$
11:    end for
12: end for
```

$$Q_{new}(s,a) = Q(s,a) + \alpha \Big[R + \gamma Q(s',a') - Q(s,a) \Big] \tag{17}$$

7. Project Aims & Objectives

7.1. Objective & Requirements

The objective of this project is to create a system that can learn to control the movement of aircraft in an airspace.

7.2. Evaluation Strategy & Experimental Design

The reinforcement learning solution is tested using a simple scenario to ensure that it works. One of the simpler tasks would be balancing a pole on a cart. An input for the horizontal movement of the card $x \in [-1, 1]$. We reward +1 for each time step that the pole stays upright (the dot product with the horizontal vector is > 0.95). The output is a signed angle of the pole to the horizontal vector. If the agent is able to balance the pole for 30 consecutive seconds, we can consider the problem solved.

Then a simple model with drones is used to explore solutions in a 2-dimensional environment (y-axis constant). In each episode, the drone is moved to a random location and is tasked with reaching point B, which is at least a few units away and chosen at random. Once the drone agent finds its way around the environment, more drones are added with additional inputs such as jam distance and command cost. Collision detection will use a simple circular collision and congestion will be triggered when drones are in close proximity to each other. The drones will be able to accelerate very quickly in their desired direction.

When a simple drone simulation succeed, experiment will be moved onto three dimensions. In the case of airspace simulation using aircraft some extra rewards and movement limitations will need to be put into place. Main ideas are:

- The plane can only fly at a certain speed
- A plane has a more limited movement flexibility compared to a drone
- Instructions can only be sent at a certain rate
- Instructions are sent at a delay of a few seconds

For our functional requirement, we can create a simple naive model to test the efficiency of the reinforcement learning solution. The drone simply flies in a straight line from A to B, ignoring traffic congestion. If a single drone can achieve a similar average speed as the naive model whilst taking congestion into account, we can call this a better solution. For the sake of simplicity, the airspace environment is treated as Markovian, meaning the current optimal action can be resolved using only the current state.

7.3. Project Timeline

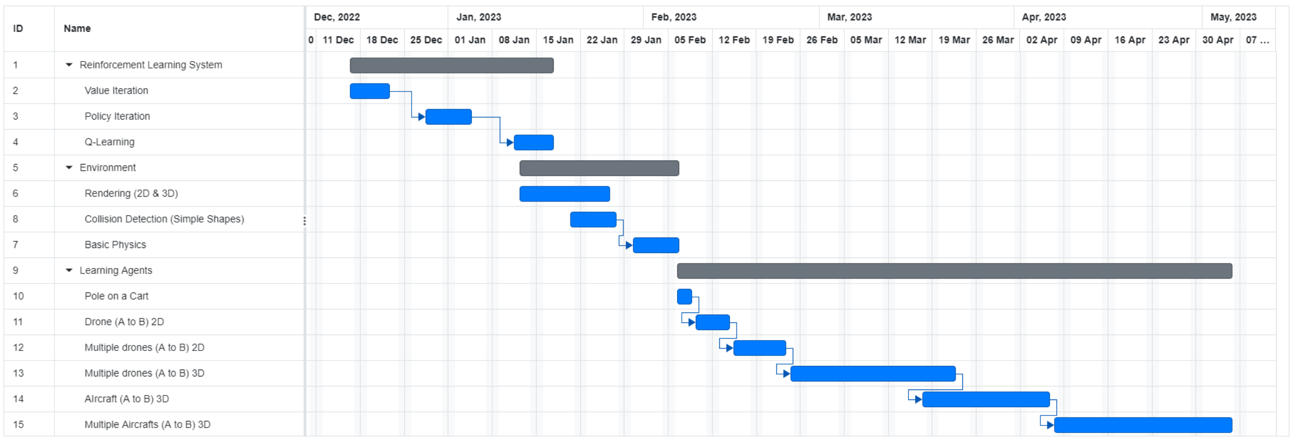

Fig. 3. Timeline for the project. Arrows show dependencies.

Project will start by implementing a basic reinforcement learning library capable of different policy learning algorithms. The library will be tested on variety of problem types, slowly leading to the final air traffic control problem. Solution will be written in Python. OpenAI Gym [Brockman et al. 2016] offers a way of comparing reinforcement learning algorithms with a standardised API and will be used initially in the project to ensure library outputs the correct results. It will also greatly simplify complexity of experiments as the focus will be strictly on the reinforcement learning library rather than essentials like physics or rendering.

Due to many hyper-parameters that need to be tuned, it is very hard to predict how long it will take to train the system. Which is why the reinforcement learning system needs to work early on, so that we can work on many simulations with different parameters, and see which one works best. The more complex the experiments become, the more time is needed to find the optimal policy for each scenario. For this reason, the time span in Fig.3 increases drastically at the end.

7.4. Reward System for air traffic control

A study at IBM Research AI [Ghosh et al. 2021] has found that using the following reward/penalty systems have improved the reinforcement learning system in an air traffic control system:

- Conflict cost - To encourage separation distance, each agent will receive a high penalty if the distance between them is closer than a threshold.
- Congestion cost - If number of agents in a given radius exceeds a threshold, a penalty is imposed.
- Path length cost - To minimise distance travelled, reward when plane arrives will be inversely proportional to distance travelled.
- Fuel cost - We want to minimize usage of fuel (which also will minimize the distance travelled), so our agent will receive penalty based on fuel used.

Additionally we can impose an instruction cost which penalises every time agent sends an instruction, this could be used to minimise instructions sent thus only sending the most important information across. (In a real life scenario we wouldn't want to flood the radio channel with repeating information).

To combine all of the rewards together we could naively sum them together, but this could cause issues as one rewards might out-weight other ones. This is something that would need to be adjusted during simulation testing.

7.5. Evaluation & Success Criteria

The project will be successful if we can find out whether reinforcement learning is a good candidate for the task, and if so, we can start working on a more complex simulation. Before the main project can be tackled, we will conduct a series of smaller experiments to test the system. The results will be analysed to see if the system is able to learn from experience and adapt to new situations. The system should be able to learn in a small environment and adapt this relatively quickly to a larger environment. If we add more agents, they should be able to solve the problem together without major disruptions.

7.6. Ethical Issues

There are no ethical issues associated with this project, as the system will not be used in a real life scenario, and will not be used to control real aircraft. This project aims to test the feasibility of using reinforcement learning to control aircraft in an airspace.

7.7. Conclusion

This paper proposes an efficient real-time solution for the air traffic control system. Further research will be conducted to determine whether the dimensional complexity of the problem allows to solve it in a feasible time. If it is, the most appropriate strategies for the task will be categorised and broken down, along with the hyper-parameters used and their impact on the solution.

REFERENCES

Markov chain. 2022. URL https://en.wikipedia.org/wiki/Markov_chain. Visited on 19 Oct 2022.

Markov decision process. 2022. URL https://en.wikipedia.org/wiki/Markov_decision_process. Visited on 19 Oct 2022.

J.-M. Alliot, H. Gruber, G. Joly, and M. Schoenauer. Genetic algorithms for solving air traffic control conflicts. In *Proceedings of 9th IEEE Conference on Artificial Intelligence for Applications*, pages 338–344, 1993. doi: 10.1109/CAIA.1993.366591.

Ó. Arias-Carrión and E. Pöppel. Dopamine, learning, and reward-seeking behavior. *Acta neurobiologiae experimentalis*, 2007. Visited on 23 Nov 2022.

M. Banoula. What is q-learning: The best guide to understand q-learning. 2022. URL https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-is-q-learning. Visited on 5 Nov 2022.

Y. Bengio. Practical recommendations for gradient-based training of deep architectures. *Lecture Notes in Computer Science*, page 9, 2012. doi:10.1007/978-3-642-35289-8_26. Visited on 24 Oct 2022.

Bonsai. Why reinforcement learning might be the best ai technique for complex industrial systems. 2017. URL https://medium.com/@BonsaiAI/why-reinforcement-learning-might-be-the-best-ai-technique-for-complex-industrial-systems-fde8b0ebd5fb. Visited on 26 Oct 2022.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016. Visited on 20 Nov 2022.

DeepMind. Alphago. 2022. URL https://www.deepmind.com/research/highlighted-research/alphago. Visited on 24 Oct 2022.

S. Dittert. Reinforcement learning: Value function and policy. 2020. URL https://medium.com/analytics-vidhya/reinforcement-learning-value-function-and-policy-c22f5bd1d1b0. Visited on 29 Oct 2022.

I. R. Galatzer-Levy, K. V. Ruggles, and Z. Chen. Reinforcement learning schematic. 2018. URL https://www.researchgate.net/figure/Reinforcement-learning-schematic-Reinforcement-learning-RL-can-be-formulated-as-a_fig4_322424392. Visited on 19 Oct 2022.

S. Ghosh, S. Laguna, S. H. Lim, L. Wynter, and H. Poonawala. A deep ensemble method for multi-agent reinforcement learning:a case study on air traffic control. 2021. URL https://ojs.aaai.org/index.php/ICAPS/article/view/15993/15804. Visited on 19 Oct 2022.

R. Heberer. Why going from implementing q-learning to deep q-learning can be difficult. 2019. URL https://towardsdatascience.com/why-going-from-implementing-q-learning-to-deep-q-learning-can-be-difficult-36e7ea1648af. Visited on 8 Nov 2022.

H. Hristov. Off-policy vs. on-policy reinforcement learning. 2022. URL https://www.baeldung.com/cs/off-policy-vs-on-policy. Visited on 29 Oct 2022.

J. Lai, K. Cai, Z. Liu, and Y. Yang. A multi-agent reinforcement learning approach for conflict resolution in dense traffic scenarios. *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC), Digital Avionics Systems Conference (DASC), 2021 IEEE/AIAA 40th*, pages 1 – 9, 2021. ISSN 978-1-6654-3420-1. URL https://uoelibrary.idm.oclc.org/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.9594437&site=eds-live&scope=site.

M. Lee. Value functions. 2005. URL http://www.incompleteideas.net/book/ebook/node34.html. Visited on 15 Nov 2022.

P. Loeber. Reinforcement learning with (deep) q-learning explained. 2022. URL https://www.assemblyai.com/blog/reinforcement-learning-with-deep-q-learning-explained/. Visited on 8 Nov 2022.

R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments, 2017. URL https://arxiv.org/abs/1706.02275.

B. Lutte. Pilot supply at the regional airlines: Airline response to the changing environment and the impact on pilot hiring. 2018. URL https://commons.erau.edu/jaaer/vol27/iss1/1/. Visited on 26 Oct 2022.

A. Mukherjee, Y. Guleria, and S. Alam. Deep reinforcement learning for air traffic conflict resolution under traffic uncertainties. *2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC), Digital Avionics Systems Conference (DASC), 2022 IEEE/AIAA 41st*, pages 1 – 8, 2022. ISSN 978-1-6654-8607-1. URL https://uoelibrary.idm.oclc.org/login?url=https://search.ebscohost.com/

login.aspx?direct=true&db=edseee&AN=edseee.9925772&site=eds-live&scope=site.

Neelarghya. Reinforcement learning vs genetic algorithm — ai for simulations. 2021. URL https://medium.com/xrpractices/reinforcement-learning-vs-genetic-algorithm-ai-for-simulations-f1f484969c56. Visited on 23 Nov 2022.

R. Ng. Markov decision processes (mdp) and bellman equations. 2021. URL https://www.deeplearningwizard.com/deep_learning/deep_reinforcement_learning_pytorch/bellman_mdp/. Visited on 21 Oct 2022.

S. of Computer Science and E. U. Sydney. Reinforcement learning. URL https://www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html. Visited on 6 Nov 2022.

D. L. Poole and A. K. Mackworth. Value iteration. 2018. URL https://artint.info/2e/html/ArtInt2e.Ch9.S5.SS2.html. Visited on 8 Nov 2022.

V. Powell and L. Lehe. Markov chains explained visually. 2022. URL https://setosa.io/ev/markov-chains/. Visited on 19 Oct 2022.

W. B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality, 2nd Edition*. Wiley, 2 edition, 2011. Visited on 26 Oct 2022.

D. Silver. Rl course by david silver - lecture 2: Markov decision process. 2015a. URL https://www.youtube.com/watch?v=lfHX2hHRMVQ. Visited on 21 Oct 2022.

D. Silver. Rl course by david silver - lecture 3: Planning by dynamic programming. 2015b. URL https://www.youtube.com/watch?v=Nd1-UUMVfz4. Visited on 5 Nov 2022.

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

D. Soemers. What is the difference between q-learning and sarsa? 2018. URL https://stackoverflow.com/questions/6848828/what-is-the-difference-between-q-learning-and-sarsa. Visited on 8 Nov 2022.

R. S. Sutton and A. G. Barto. *Reinforcement Learning, An Introduction, Second Edition*. MIT Press, Cambridge, MA, 2018, 2 edition, 2018.

A. A. Tokuc. Value iteration vs. policy iteration in reinforcement learning. 2022. URL https://www.baeldung.com/cs/ml-value-iteration-vs-policy-iteration. Visited on 15 Nov 2022.

V. M. Vilches. Reinforcement learning in robotics. 2016. URL https://medium.com/@vmayoral/reinforcement-learning-in-robotics-d2609702f71b. Visited on 24 Oct 2022.

A. Violante. Simple reinforcement learning: Q-learning. 2019. URL https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56. Visited on 5 Nov 2022.

T. R. Zentall. Coding of stimuli by animals: Retrospection, prospection, episodic memory and future planning. *Learning and Motivation*, 41(4):225–240, 2010. Visited on 20 Nov 2022.

L. Zou, L. Xia, Z. Ding, J. Song, W. Liu, and D. Yin. Reinforcement learning to optimize long-term user engagement in recommender systems. 2019. URL https://ieeexplore.ieee.org/abstract/document/9514509. Visited on 24 Oct 2022.