Deep Self-optimizing Artificial Intelligence for Tactical Analysis, Training and Optimization

Michael Rüegsegger, Matthias Sommer
armasuisse Science and Technology
Feuerwerkerstr. 39, 3602 Thun
SWITZERLAND
michael.rueegsegger@armasuisse.ch, matthias.sommer@armasuisse.ch

Giacomo Del Rio, Oleg Szehr
Dalle Molle Institute for Artificial Intelligence (IDSIA) - SUPSI/USI
Via la Santa 1, 6962 Viganello
SWITZERLAND
giacomo.delrio@idsia.ch, oleg.szehr@idsia.ch,

ABSTRACT

The increasing complexity of modern multi-domain conflicts has made their tactical and strategic understanding and the identification of appropriate courses of action challenging endeavours. Modelling and simulation as part of concept development and experimentation (CD&E) provide new insight at higher speed and lower cost than what physical manoeuvres can achieve. Amongst other, human-machine teaming through computer games is a powerful means of simulating defence scenarios at various abstraction levels. However, conventional human-machine interaction is time-consuming and restricted to pre-designed scenarios, e.g., in terms of pre-programmed conditional computer actions. If one side of the game could be taken by Artificial Intelligence (AI), this would increase the diversity of explored courses of actions and lead to a more robust and comprehensive analysis. If the AI plays both sides, this allows employing the Data Farming methodology and creating and analysing a database of a large number of investigated scenarios. To achieve a high degree of variability and generalization capability of the investigated scenarios, we employ combined Reinforcement Learning and search algorithms, which have demonstrated super-human performance in various complex planning problems. Such AI systems avoid the reliance on training data, human experience and predictions by learning tactics and strategy through exploration and self-optimization. In this contribution, we present the benefits and challenges of applying a Neural-Network-based Monte Carlo Tree Search algorithm for strategic planning and training in air-defence scenarios and virtual war-gaming with systems that are available currently or potentially in the future to the Swiss Armed Forces.

1.0 INTRODUCTION

Today, wargaming is a widely applied and accepted tool within the military domain for planning, training, and decision-making. However, traditional wargaming, where military operators and decision makers play against Red Team operators, faces limitations [1]. Wargames are driven by player decisions and are therefore often hard to reproduce. Objective tracking of cause and effect is difficult, and outcomes often depend on the subjective assessments of human players. Usually, Red Team operators rely upon long established techniques, tactics and procedures leading to over-constrained Courses of Action (CoA) {REFEREE:please add REFERENCE}. Finally, the number of CoAs that can be explored by human operators is limited, by the availability of both financial and human resources. This is even more true given the complexity of continuous-time decision making and the large numbers of available options (high branching factors) that characterize real-world scenarios. In practice this results in a relatively small number of explored scenarios and a lack of strategic diversity in CoA exploration, leading to biased and potentially

vulnerable decision-making. With the approach presented in this paper, we hope to address and overcome these challenges.

Recent developments in the Artificial Intelligence (AI) area and in particular Reinforcement Learning (RL), have demonstrated the ability of artificial agents to master commercial real-time strategy (RTS) video games of increasing complexity, reaching super-human performance in games such as StarCraft [2]. Artificial agents have also proven super-human tactical and strategic capabilities in complex games such as Chess and Go [3]. Self-playing systems have been applied for various purposes, amongst other, assessment of quality of experience in video gaming industry [9] as well as decision support in defence [10]. Typically, the involved RL systems are capable of learning 'tabula rasa', i.e., without relying on provided training-data or human experience. The training samples used for the optimization of the RL agent are produced in the course of the training process by exploring complex decision-spaces in a large number of scenarios and subsequent self-optimization. This avoids the bias towards a specific data-farming methodology, or human experience and perception and leads to more robust, stronger, and non-emotional decisions in execution. While the learning process of AI systems often involves significant computational effort, their speed in execution is fast, often outperforming the 'discussion-based' human decision process.

In this work, we investigate AI systems, in particular Neural-Network-based Monte Carlo Tree Search algorithms (neural MCTS), to support planning, training and decision-making in the field of Ground-Based Air Defence (GBAD). We apply the AI to the commercial of the shelf (COTS) wargame "Command: Modern Operations" (CMO) to explore complex decision-spaces, and to generate surprising new Red Force CoAs. The chosen algorithm combines two key features. First, being an RL algorithm, neural MCTS is capable of learning tabula rasa, i.e., data samples that are used for learning are generated during the training process. In CMO, where only little CoA data is publicly available for training, this is an important advantage as compared to supervised learning techniques. Second, neural MCTS combines the RL setup with directed search towards promising CoAs. This feature is important for the capability of a trained agent to operate in previously unseen scenarios. While the Neural-Network components reflect abstract qualitative features for the evaluation of scenarios, the search component matches the planned actions to the acquired knowledge, providing important generalization capabilities. In the future this will challenge the preconception of Blue Force operators and stimulate the development of new techniques, tactics, and concepts.

2.0 METHODS

Two main software components are involved in the application described above. First, there is the model with rules and physical constraints of the scenario to be simulated (the so-called simulator), and second, an AI algorithm that controls one or both players in the conflict represented by the model. In the current scenarios, one side of the game is played by an AI agent, while the other is controlled by the game engine itself, through pre-scripted, conditional actions. Specifically, the AI controls the attacking red fighter jet while the game engine controls the blue air defence batteries.

2.1 Model

The use of commercial off-the-shelf (COTS) games as simulation engines has a number of advantages: To name a few, COTS are available at low cost and often possess over a large user-base, high fidelity and come with a good support. Also, the gaming industry has been a driver for the development of realistic warfare scenarios, respective visualizations, and tailor-made hardware. Two central requirements for a model to be controlled by an external RL agent are:

1. To produce the training data, the RL training process executes a large number of simulations. The possibility of faster-than-real-time simulation is therefore important. This requires to run the simulation in a fast 'console environment', without graphics, internet connection and other miscellaneous features.
2. interface to control and assess the state of the game externally.

The game "Command: Modern Operations" (CMO) fulfils these requirements in its professional version (Command Professional Edition) and covers the air defence domain [11]. It is a real-time strategy game mainly for air and sea combat that features a comprehensive and realistic database of past, current and future weapon systems.

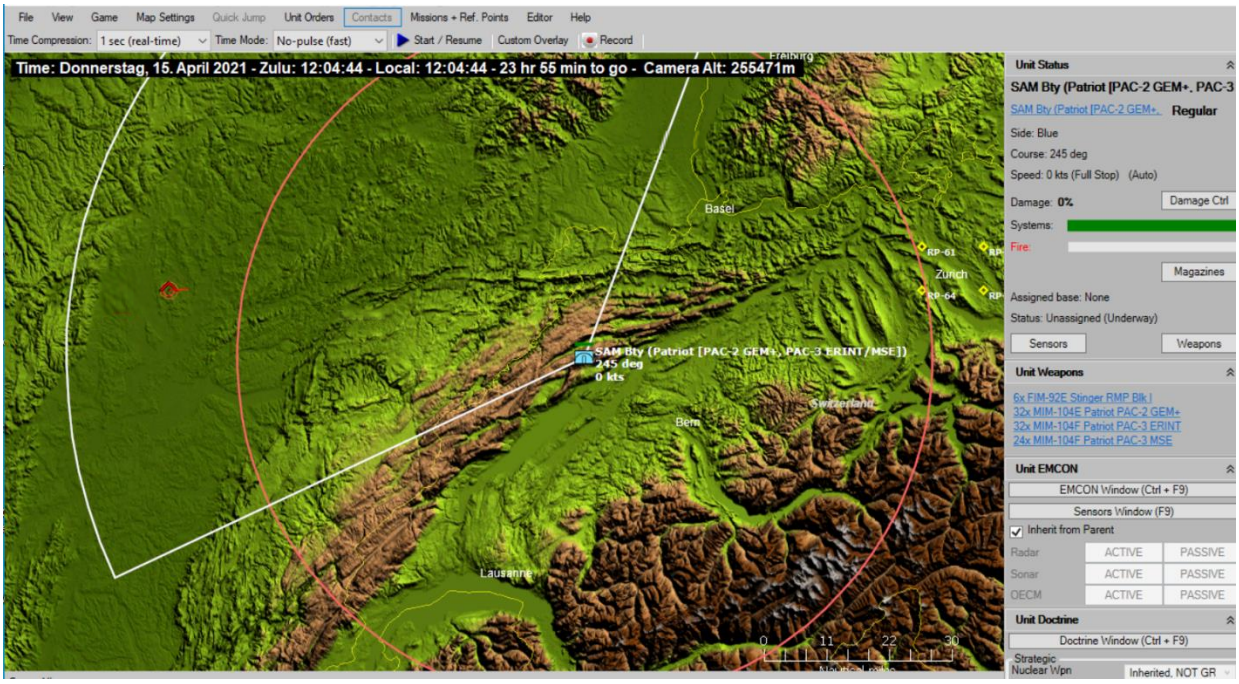

Figure 2-1: User interface of Command: Modern Operations

CMO provides an Application Programming Interface (API) that allows interaction with the running simulation in batch mode, i.e., without graphics. In order to make the game more amendable to the AI, we introduced a turn-based mode for the real-time game. At each time step the game is paused and the AI can assess the scenario and select an action. Subsequently, the action is executed, and the simulation proceeds for a given time window, here 10 seconds simulation time.

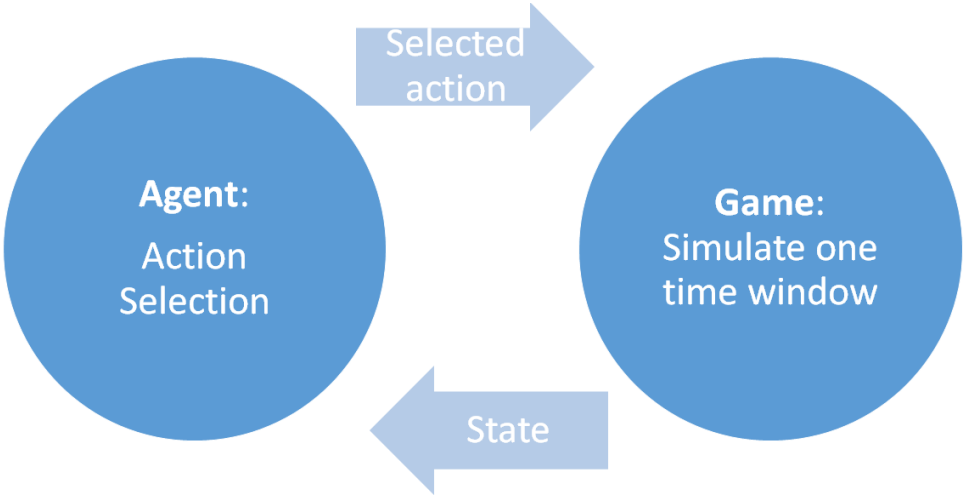

Figure 2-2: Turn-based game mode of Command: Modern Operations

In the design mode, one has the possibility to create a scenario with pre-defined or custom assets. In game mode, the simulation is then executed with the actions from either the players or the external AI.

2.1 AI

For complex game search problems, such as the CMO scenarios, the direct computation of optimal policies is intractable. The development of suitable approximation techniques has been an active area of research for many years leading to striking AI systems that outperform humans and classical tree search algorithms in strategic games such as Chess and Go [3]. At the core of the AI lies a sophisticated neural network-based Monte Carlo Tree Search algorithm, whose performance is the result of a combination of techniques from diverse areas including Reinforcement Learning, Monte Carlo simulations and Deep Neural Networks. Traditionally search algorithms (such as alpha-beta search) are employed to study game play, but the search tree grows exponentially with its depth leading to large computational effort even for simple games. The tree's terminal nodes must be evaluated by a tailor-made valuation function, whose design involves significant human expertise. On the other hand, the choice of actions in a (Markovian) game can be viewed as a planning problem, with an agent whose goal it is to win, or, in other words, to maximize the total reward over the planning horizon. Such planning problems are based on the theory of Markov decision processes [4]. The research area concerned with their solution has been commonly called Reinforcement Learning (RL) in recent years. In RL the Value/Policy iteration algorithms proceed by performing repeated Bellman updates over the state space and are guaranteed to yield the optimal policy if sufficient resources are granted. However, even for planning problems of moderate complexity the required resources tend to be immense because each Bellman update involves the entire state space. An important idea to improve the performance is to rely on Monte Carlo simulations to estimate state values [5]. In brief, each state is evaluated by trying every possible action a certain number of times, and recursively, from each generated state every possible action the same number of times, too, until the depth of the planning horizon is reached [6]. Again, this yields the optimal

course of actions in theory but in practice the width and depth of the Monte Carlo simulations are such that a near optimal solution becomes elusive.

The key idea behind MCTS is to combine the iterative algorithms from RL with (the more traditional) tree search techniques [7]. Thus, instead of relying on 'brute force Monte Carlo' for state valuation, a problem-specific, restricted and asymmetric decision tree is built. The growth of the decision tree is controlled by a dedicated tree policy, whose purpose it is to trade-off the creation of new branches versus the execution of existing promising lines. The missing piece to engineer a world-champion AI are deep neural networks. The MCTS design is enhanced by deep neural networks [8] to guide the construction of the decision tree and to provide accurate evaluation of leaf-nodes. At the core of the neural MCTS algorithm lies an 'expert iteration' scheme, where the MCTS search and the neural networks mutually improve each other in an iterative process. In each iteration an instance of MCTS (the expert) delivers a list of strong actions and a neural network (the apprentice) is trained via supervised learning to imitate the MCTS tree policy. In the next iteration, the trained neural network, in turn, is used to bias the MCTS' tree construction towards stronger actions. The purpose of the expert is to accurately determine good actions. The purpose of the apprentice is to generalize those actions across states and to provide faster access to expert advice. Specifically, the apprentice policy π is trained on tree-policy targets by minimizing cross-entropy loss at state s

$$Loss_{TreePolicy} = -\sum_{a \in actions} \frac{n(a)}{n} log\big(\pi^A(a|s)\big).$$

Here, $n(a)$ is the number of times action a has been chosen so far and n is the total number of executed actions. The value network reduces search depth and avoids inaccurate rollout-based value estimation. It is trained to minimize the mean-squared error to the MCTS valuation z of the state

$$Loss_{Value} = -\big(z - V^A(s)\big)^2.$$

In turn, the apprentice improves the expert by guiding tree search towards stronger actions. For this the standard MCTS tree policy (UCB1) is enhanced with an extra term

$$NUCB1(a) = UCB1(a) + w\frac{\pi^A(a|s)}{n(a)+1},$$

where a hyper-parameter w weights the contributions of standard MCTS tree policy (UCB1) and the neural network. Neural MCTS branch selection proceeds by choosing the action a that maximizes *NUCB1*. To regularize value prediction and accelerate training, the tree policy and valuation function are simultaneously covered by a multitask network with separate outputs. The loss for this network is the sum of losses from value and tree policy networks.

2.1.1 Training process

Being an RL algorithm, neural MCTS is capable of learning without any provided data. Data samples of increasing quality are produced by the algorithm through self-play during the training process. They are used to optimize the agent (and the quality of samples) for subsequent training iterations. This addresses an important bottleneck of supervised learning methods, which require a large number of high-quality samples for training. Ultimately, the out-of-sample performance of classical operation research and supervised learning methods is restricted by the quality of the training data. If training samples would be produced following a 'data-farming'/ Monte Carlo approach, the resulting samples would be biased towards the specific

methodology that produced them. RL naturally addresses the data-farming problem, but this comes at the price of a significantly larger computational burden, particularly in the initialization phase. To achieve faster learning and higher stability of the training process, our RL agent is pre-trained before operating on CMO directly. For this a reduced surrogate model and a dedicated API are implemented in the same programming language (Python) as the AI algorithm. This surrogate model captures some of the important features of the combat scenario, such as the location of assets at inception, the location of targets, the basics of flight dynamics etc., but the level of implementation detail is limited. The purpose of the surrogate is to pre-train the agents from scratch with significantly lower (approx. 1000 times) latency as compared to CMO. The pretrained agent is subsequently refined by training on the full game engine. Thus, the process involves the training on a reduced but faster simulator and the transfer of the pretrained agent to the CMO simulator for fine-tuning. This way the data-intensive (and therefore slow) initialization of the RL process can be handled efficiently, while there is no reduction in the agent's performance due to training in a restricted setting.

3.0 SCENARIO

Armasuisse Science and Technology is investigating this technology for application to air defence training, concept development and experimentation (CD&E) as well as procurement projects. The main assets employed in the warfare scenarios are fighter jets, radars, and air defence missile systems. Ideally, an intelligent agent would be capable of identifying and executing a close-to-optimal strategy for all assets simultaneously. Due to the complexity of realistic scenarios, developing and training of such an agent is beyond the goals of the current project stage. Instead, the aim is a feasibility study with reduced scenarios, where the RL agent controls a single asset. This setup will be extended in a later stage of the project. The simple test scenarios are characterized in the above figure 2-1 and the following table 3-1:

Table 3-1: Scenario description

	Red Side	Blue Side
Assets	1 Fighter Jet	1 SAM Battery (Radar, missile system)
Mission	Penetrate blue Zurich airspace (yellow)	Defend blue Zurich airspace (yellow)
Red Scoring	+1 for attacker mission success -1 for loss of aircraft	-1 for defender mission success +1 for attacker mission success

4.0 RESULTS

The AI controlling red aircraft learns quickly that flying the direct path from its take-off to the target will make it visible to the blue radar and therefore vulnerable to blue air defence. It thus chooses a more northern route to avoid the radar cone. In the following, we will work out what went well and where we see the challenges and bottlenecks for the transition to more interesting and complex scenarios.

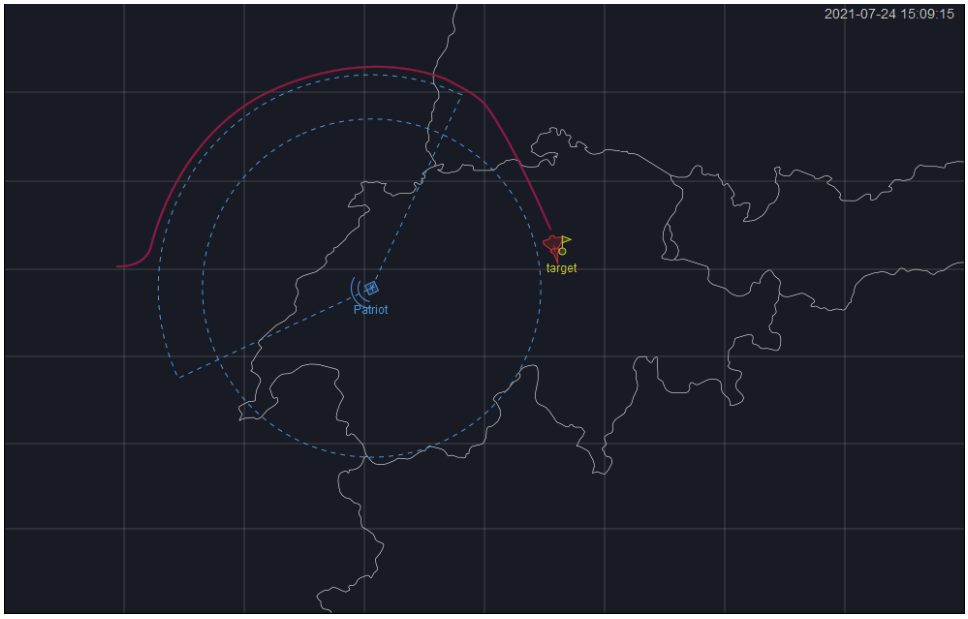

Figure 4-1: Example trajectory of trained red agent around the range of the blue air defense system

4.1 COTS Game as simulation engine and its interface

The COTS industry has been a driver of research and development into various high-end technologies of considerable significance for the armed forces. This includes detailed terrain visualizations, realistic flight simulators, and tailor-made hardware. COTS games are available at low cost and often possess over a large user-base, high fidelity and come with a good support infrastructure. The separation of AI, simulation engine and analytics tools make the application design more flexible and re-use of components possible. In particular, this approach avoids a large black-box simulation engine, where insight and extensibility are limited.

PLEASE ADD SOME MORE STUFF ON COTS ABOVE, ILL THEN TEXT EDIT IT.

4.2 Application of neural MCTS to CMO scenarios

In all investigated scenarios the outlined training procedure led to neural MCTS finding the maximal theoretical reward after only a short number of learning cycles (5 neural training cycles, each based on 500 episodes and 100 MCTS samples per search step for all investigated scenarios). While this is somewhat expected given the relatively low complexity of the combat scenario and the proven ability of neural MCTS to solve sophisticated strategic problems, the presented work provides proof of concept that the practical implementation in the CMO context is successful. Connecting neural MCTS to CMO requires a dedicated API with fast access. In our experiments we have observed that access via the existing Lua API is too slow for MCTS to guide the behaviour of CMO. To overcome this issue a dedicated surrogate simulator has been developed, which provides fast access in a simplified setting. The pretrained MCTS agents are subsequently handed to the full CMO setting for fine-tuning. As a result, we observe solidly increasing rewards on the simulator and the trained agent can be transferred to CMO successfully.

4.3 Generalization to unseen scenarios

Neural MCTS combines RL with deep neural networks and search. The role of the neural networks is to provide accurate state evaluation and to guide search towards promising CoAs. In this sense the neural network identifies abstract features of the underlying state that are relevant for the state evaluation and search. When trained on a sufficient number of different scenarios, those features allow for an assessment of the quality of the underlying state irrespectively of the specific scenario at hand. Typical features that are not specific to a given scenario include the number and combat power of present assets, map information, the relation of combat power and terrain, the relative position of units, etc. Those features provide a backbone capability to allow for a successful application of the trained agent in unseen scenarios. The search component identifies courses of action and scenario-states, where, based on those features, the neural-network is confident of a favourable outcome. Taken together these two components thus provide a natural means to generalize the agent beyond specific training scenarios.

In testing we have presented an agent, which was trained on a specific set of scenarios with an unseen situation. Overall, we observed only little degradation of chosen paths to the target, with respect to the training scenarios.

5.0 CONCLUSIONS AND OUTLOOK

Our results provide proof of concept of the application of RL and search techniques in the context of COTS wargames. In forthcoming research, the complexity of war game scenarios will be increased successively. The more complex setting is reflected by adaptations in the neural MCTS architecture. While simple linear, feed-forward networks are sufficient for the basic scenarios, we plan a deep convolutional architecture for the more complex setting. The additional computational burden of the complex scenarios is covered by code parallelization and cluster computing. We expect that MCTS will also be successful in complex scenarios as neural MCTS algorithms proved their superior performance in strategy games like Chess and Go. Overall, the value of an independent and innovative simulation application for training as well as CD&E seems larger than the prize to overcome the challenges in developing it.

[1] Wargaming Handbook, UK Ministry of Defense (2017)

[2] Vinyals, Oriol, et al. "Grandmaster Level in StarCraft II Using Multi-Agent Reinforcement Learning." *Nature News*, Nature Publishing Group, 30 Oct. 2019, www.nature.com/articles/s41586-019-1724-z#citeas.

[3] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. A general reinforcement learning algorithm that masters Chess, Shogi, and Go through self-play. Science, 362(6419):1140–1144, 2018.

[4] M.L. Puterman. Markov decision processes: discrete stochastic dynamic programming. Wiley Interscience, New York, 1994

[5] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A Sparse Sampling Algorithm for Near-OptimalPlanning in Large Markov Decision Processes. Machine Learning, 49(2/3):193–208, 2002

[6] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A Sparse Sampling Algorithm for Near-OptimalPlanning in Large Markov Decision Processes. Machine Learning, 49(2/3):193–208, 2002

[7] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A Survey of Monte Carlo Tree Search Methods. IEEE Transactions on Computational Intelligence and AI in Games, 4(1):1–43, 2012

[8] J. Schmidhuber. Deep Learning in Neural Networks: An Overview. Neural Networks, Volume 61, January 2015, Pages 85-117 (DOI: 10.1016/j.neunet.2014.09.003)

[9] Sviridov, G., Beliard, C., Bianco, A., Giaccone, P., & Rossi, D. (2020, July). Removing human players from the loop: AI-assisted assessment of Gaming QoE. In IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) (pp. 1160-1165). IEEE.))

[10] Pottrill et al 2020. Hierarchical Reinforcement Learning: A Novel Approach to Decision Support Analysis, Proceedings 14th NATO ORA conference

[11] Command Modern Operations Manual: https://www.matrixgames.com/amazon/PDF/CMO/CMO_manual_EBOOK.pdf and description of professional edition https://command.matrixgames.com/?page_id=3822