SINGLE & MULTI AGENT REINFORCEMENT LEARNING FOR STRATEGY GAMES

A Practice School Report submitted to
Manipal Academy of Higher Education
in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

in

Computer Science & Engineering

Submitted by
Rahat Santosh

Under the guidance of

Mr. Ashish Srivastava
Scientist 'D'
Centre for Artificial Intelligence and Robotics(CAIR), Defence Research and Development Organization

Prof. Dr. Ashalatha Nayak
Professor and Head of the Department

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

July 2022

MANIPAL INSTITUTE OF TECHNOLOGY

MANIPAL

(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Manipal

15 / 07 / 2022

CERTIFICATE

This is to certify that the project titled **SINGLE & MULTI AGENT REINFORCEMENT LEARNING FOR STRATEGY GAMES** is a record of the bonafide work done by **RAHAT SANTOSH** *(Reg. No. 180905194)* submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech.) in **COMPUTER SCIENCE & ENGINEERING** of Manipal Institute of Technology, Manipal, Karnataka, (A Constituent Institute of Manipal Academy of Higher Education), during the academic year 2021-2022.

Prof. Dr. Ashalatha Nayak

Professor & Head of Department

Prof. Dr. Ashalatha Nayak

HOD, CSE Dept.

M.I.T Manipal

कृत्रिम ज्ञान तथा रोबोटिकी केन्द्र
रक्षा अनुसंधान तथा विकास संगठन
भारत सरकार – रक्षा मंत्रालय
डी.आर.डी.ओ कॉम्प्लेक्स, सी वी रामन नगर, बेंगलूर- 560093
फोन: 91-80-25244288, फैक्स: 91-80-25244298,2534 2644
ईमेल: director.cair@gov.in,.cair@gov.in
आई. एस. ओ. 9001:2015 प्रमाणित केंद्र
सी.एम.एम.आई. डेव 1.3, परिपक्वता लेवल 2 मूल्यांकित केंद्र

सभी पत्रादि निदेशक के पते से भेजे जाने चाहिए।
All correspondence to be addressed to the Director

CENTRE FOR ARTIFICIAL INTELLIGENCE AND ROBOTICS
Defence Research & Development Organisation
Government of India– Ministry of Defence
DRDO Complex, CV Raman Nagar, Bangalore- 560093
Tel: 91-80-25244288, Fax: 91-80-25244298, 25342644
E-mail: director.cair@gov.in,.cair@gov.in
ISO 9001:2015 Certified Establishment
CMMI Dev 1.3, Maturity Level 2 Appraised Establishment

आज़ादी का अमृत महोत्सव

Certificate

This is to certify that the Student Trainee Mr. Rahat Santosh pursuing B. Tech (CSE) at Manipal Institute of Technology, Manipal has been given the opportunity to work on the topic **"Single and Multi Agent Reinforcement Learning for Strategy Games"** at Centre for Artificial Intelligence and Robotics (CAIR) during 03rd Jan 2022 to 15th Jul 2022.

Signature of Guide
Name & Designation:
(Ashish Srivastava, Scientist 'D')

प्रभारी अधिकारी (शैक्षणिक)/OiC (Academics)
कृते निदेशक सी.ए.आई.आर./For DIRECTOR, CAIR.

Place: Bengaluru – 93
Date: 15/7 Jul 2022

Director CAIR should not be approached for any references or recommendations

ACKNOWLEDGEMENTS

I am grateful to have received an opportunity to pursue my final semester project at an organization of such repute as the Center for Artificial Intelligence and Robotics (CAIR), Defence Research and Development Organization, and to be able to work on such an exciting and novel topic, that has assisted me in developing both personally and professionally and has helped me progress towards my goals. I am extremely grateful to CAIR for having afforded me this internship and for giving me access to the resources, computational and otherwise, without which I would not have been able to work in such a field.

I would like to express my sincere gratitude to my supervisor at CAIR, Mr. Ashish Srivastava, for taking time out of his busy schedule to provide guidance, suggestions, and motivations throughout the duration of the project and for allowing me to work on such an interesting and novel field. I would also like to thank my department HOD and internal guide, Dr. Ashalatha Nayak, for assisting me in getting this opportunity and the project supervisor, Dr. Radhika Kamath, for further guiding me through the course of this project and the other related formalities. I also want to take this opportunity to thank the Director, Manipal Institute of Technology.

I would like to acknowledge my gratitude towards my family and friends for providing moral support and encouragement.

I perceive this opportunity as a major milestone toward my professional and academic goals, and I will strive to use gained experience and knowledge to my best abilities and will continue to strive towards their improvement. I hope to have continued cooperation with all I have had the opportunity to interact with during this project.

Sincerely,
Rahat Santosh

Place: Bangalore
Date: 15/07/2021

ABSTRACT

Strategy games are most relevant to real-world applications due to their interactions between the agent(s) and the environment. This nature makes them ideal candidates to benchmark and tests the relevance of reinforcement learning algorithms in different practical scenarios. One can notice this in the observation that most algorithms in artificial intelligence, especially reinforcement learning, have been benchmarked against various strategy games. Further, the nature of real-time strategy games makes them an ideal testbed and launch platform for using deep reinforcement learning tasks in command and control tasks, as opposed to the standard combat simulators. The project also attempts to draw parallels to effectively translate from a real-time strategy game environment into a more realistic combat simulation environment. The complexity of the StarCraftII game, as compared to other strategy and real-time strategy games, makes it ideal for the above targets.

During the course of this project, we explore a variety of algorithms benchmarked against important strategic games of both perfect and imperfect information, attempt to improve upon the existing algorithms. The AlphaZero algorithm is implemented, and further, the mini-AlphaStar algorithm is implemented in place of the AlphaStar algorithm due to the computational constraints. Further, a hierarchical multi-agent reinforcement learning network has been implemented on the SMAC environment, a pure combative multi-agent environment. Here, a hierarchical approach is taken; hence there is a centralized and decentralized aspect.

The mini-AlphaStar algorithm can achieve the target win rate of 50%, which is at a saturated learning point and hence cannot be improved without altering the model's capacity, hence increasing the computational costs. The hierarchical MARL network has also been implemented, which can further be used to command and control tasks due to the shared inherent structure.

In conclusion, the mini-AlphaStar algorithm has achieved a 50% win rate but with a decreasing win rate. This can be attributed to the increasing divergence between the supervised and the reinforcement policies, which can be bridged by using the KL divergence loss, as in the original algorithm. The hierarchical network has better computational efficiency, but the performance is stuck in a cyclical loop, which can be improved using a centralized critic for the lower policies.

Contents

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

The scope of this chapter is to give a high level introduction to Reinforcement Learning(RL), including both single and multi-agent RL. It further discusses the practical significance of the topic of work and the underlying motivation for the same. It also discusses the intended objectives and target specifications of the project. Finally, the chapter lists the project schedule and report organization.

Reinforcement learning (RL) is the branch of machine learning concerned with learning sequential decision-making tasks from interactions with an environment, guided via a series of corresponding positive or negative rewards. One can argue that RL as a field of study is the closest to mimicking the nature of the human brain at a high level. It naturally follows that this field extends into Neuroscience as a study of Reward systems or as Operant Conditioning in Psychology.

Reinforcement learning transcends many fields under different names- Operations Research, Bounded Rationality, Optimal Control, Operant Conditioning, etc.[1], and has a wide range of applications, extending from Economics to Engineering to Mathematics. The most significant applications of RL are in problems where one cannot apply direct supervised learning due to a lack of tangible, simple enough data or where simple reinforcement learning cannot ensure sufficient exploration. RL algorithms can be single or multi-agent, pertaining to the number of learnable agents deployed in the environment, and model-based or model-free.

Single-agent games consist of a single agent interacting with the environment. For example, an agent in the 'Cart-Pole' environment consists of a single agent attempting to balance a pole, and there is no possibility of adding a new agent. As opposed to this, Multi-agent games consist of two or more agents interacting in an environment; this interaction can be cooperative or competitive. In most cases, the agents have independent control; the interaction is automatically learned depending on implicit or explicit rewards. Game-theoretic concepts can be used to understand these interactions better. In the case of 2 player games, the interaction is simple enough to be modeled by simple self-play or versions of fictitious self-play. Whereas in

multi-agent games with more than two players, more complex methodologies are used to enable the agents to reach an optimum strategy, keeping in mind the responses of other significant agents, for example, the league training algorithm, as used in AlphaStar.

Here, the specific focus is on reinforcement learning for strategy games. A strategy game is a system of interaction wherein each player chooses an action with no information about the other player's actions. In such cases, the aim of the algorithm is, theoretically, to achieve a Nash equilibrium - an optimum solution for non-cooperative games, where the player is unable to make any gains by deviating from their strategy, given that no other player deviates from their strategy. Strategy games can have perfect or imperfect information. Perfect information means that all aspects of the game are visible to the players; for example, in chess where both players can view the entirety of the board and the opponent's moves. Alternatively, imperfect information means that certain aspects of the game are hidden, such as poker, where only the player's cards are visible. Generally, reinforcement learning algorithms for perfect information games tend to be model-based and model-free for imperfect information games since it is easier to approximate the environment model when all aspects of the environment and its responses are accessible to the algorithm.

Real-time strategy(RTS) games are a subsection of strategy games, where the progress is not incremental in steps but simultaneously; all players can make moves in real-time. For example, a game such as Chess or Go would be a strategy game with incremental moves, and StarCraft II or DOTA 2 would be an RTS game with all players making simultaneous moves.

1.1. Organization

The Centre for Artificial Intelligence and Robotics is a laboratory established in 1986 under the Defence Research and Development Organization, established for the research and development of Artificial Intelligence, Robotics, Command and Control, and Information and Communication Security.[2]

1.2. Areas of Computer Science

- Reinforcement Learning
- Machine Learning
- Deep Learning
- Computer Vision

- Artificial Intelligence
- Game Theory
- Probability Theory

1.3. Hardware and Software Requirements

Software Requirements:

- Python
- PyTorch
- OpenAI Gym
- Box-2D
- PettingZoo
- PyGame
- Anaconda
- PySC2
- SMAC
- PyMARL
- Stable-baselines3
- Ray RLLib
- S2Protocol

Specific Hardware Requirements: NVIDIA Graphical Processing Units (GPUs).

1.4. Present Day Scenario in the Work Area

In the current scenario, RL has gained traction, especially since the adaption of deep learning function approximators to traditional RL algorithms. This has demonstrably improved the performance of these algorithms, as first proven in the Deep Q Networks (DQN)[3] paper. RL is beneficial in fields requiring a sequential interaction with an environment as opposed to an independent and identically distributed dataset (iid). In the case of MARL, the applications are extended to fields where the interaction is not just with the environment but a variety of other agents, or in other words, in a non-stationary environment. The applications of MARL

would be any distributed system with multiple agents, such as 5G networks, vehicular networks, Internet of Things, or unmanned vehicular clusters.[4]

Further, related to the scope of this project, with improving performance of MARL algorithms, particularly in strategy games, it has an increasing role in AI Command and Control, which has been the purview of more traditional and heuristic AI methodologies. The applications of this extend beyond the obvious battlefield simulation and planning to similar wargame tasks such as epidemic prevention and pest control.

1.5. Motivations of the Project Work

A strategic game is a model of interaction in which each player chooses an action not having been informed of the other players' actions[5]. Reinforcement learning algorithms on strategy games can learn strategies that can be translated to more practical applications. It naturally follows that a strategic game is an ideal environment to test reinforcement learning algorithms to have further relevance to real-world applications; that is, a strategic game most closely mimics the nature of a real-world interaction between an agent and the environment for all practical purposes. This application can be attested to by the fact that all previous milestones in artificial intelligence have also been, in most cases, measured against strategic games, from IBM Deep Blue in Chess[6] to AlphaGo[7] in Go and AlphaStar[8] in StarCraft II.

Real-time strategy(RTS) games can mimic combat simulations to train reinforcement learning algorithms for command and control tasks. Commercial RTS games such as StarCraft and DOTA series, coupled with deep reinforcement learning methods to find optimal strategies, are improving the possibility of applying deep RL to complex battlefield simulations[9]. A proven optimal policy from an RL algorithm can be of high value for military planning. The suitability of RTS games as an environment for combat simulations has been demonstrated in the paper "On Games and Simulators as a Platform for AI Command and Control" [10], where the authors have demonstrated the use of StarCraft II with a custom map, made using the StarCraft Galaxy Map Editor, to implement "TigerClaw", an offensive operation scenario. In such cases, multi-agent reinforcement learning can be used to simulate the lower level and higher level decision-making. Hence, multi-agent reinforcement learning algorithms trained on real-time strategy games can be further fine-tuned to assist in command and control situations. Further, a network modeled on the command and control hierarchical structure can also improve performance in a multi-agent environment.

Applying reinforcement learning algorithms to strategy games, be it perfect or imperfect information games or RTS games, can help benchmark and further translate these algorithms into corresponding practical situations. And hence the application of reinforcement learning

algorithms to various strategic game environments can validate algorithms and open up numerous fields, depending on their nature, for the application of reinforcement learning.

1.6. Objectives

1. Implement model-based single-agent reinforcement learning algorithms on perfect information strategy games, such as Monte Carlo Tree Search[11] in the AlphaGo series.
2. Implement model-free multi-agent reinforcement learning algorithms, such as actor-critic, on imperfect information real-time strategy games, such as in the AlphaStar[8].
3. Explore and build on top of the existing single and multi-agent reinforcement learning algorithms applied to perfect and imperfect information strategic/real-time strategic games.
4. Explore AI command and control tasks using multi-agent frameworks and environments.

1.7. Target Specifications

The expected result is the implementation of the AlphaZero algorithm and, more importantly, the implementation of AlphaStar or a related algorithm. Further improvement of the algorithm for real-time strategy games in terms of increased performance or computational efficiency. Explore the use of real-time strategy games as a testbed environment for command and control algorithms.

1.8. Project Work Schedule

- January 2022 *[Preparatory Phase]*
  - Machine Learning Course – Andrew Ng [12]
  - Reinforcement Learning Course – David Silver [1]
  - Probability Course (Markov Chains) - John Tsitsiklis [13]
- February 2022 *[Paper Reading and Implementations]*
  - AlphaGo
  - AlphaGo Zero [14]
- March 2022 *[Paper Reading and Implementations]*
  - AlphaZero [15]

  - Literature Review - Multi-Agent Deep Reinforcement Learning for Combat Simulation

- April 2022 *[Paper Reading and Implementations]*
  - AlphaStar[8] - mini-AlphaStar[16] implementation, which is a lighter version of the AlphaStar algorithm, which can be run on a single commercial server.[17]
  - Game Theory preparatory course specific to multi agent reinforcement learning.
- May 2022 *[Literature Review and Modifications]*
  - Continuation of mini-AlphaStar implementation.
  - Literature review of environments suited to multi agent combat tasks.
  - Literature review of standard multi agent reinforcement learning specific techniques.
- June 2022 *[Literature Review and Modifications]*
  - Result analysis for mini-AlphaStar.
  - Work on implementation of hierarchical network with a single upper and multiple lower policies, on the StarCraft Multi Agent Challenge(SMAC) environment.
  - Project Report Writing
- July 2022 *[Project Completion]*
  - Continuing work on hierarchical multi agent network and result analysis of the same.
  - Project Report Writing

1.9. Organization of the Project Report

- **Chapter 1 - Introduction:** General introduction to the area of work and motivation for the project work. Also discusses the project work schedule and objectives of the project.
- **Chapter 2 - Background Theory/Literature Review:** Describes the result of the literature survey and the background theory related to the area of the project work.
- **Chapter 3 - Methodology:** Implementation details, algorithms, and flowcharts of all the algorithms implemented in the project.
- **Chapter 4 - Results and Discussions:** Analysis of the result of each experiment and its significance. It also discusses any deviations in the performance and possible reasoning behind it.

- **Chapter 5 - Conclusion and Scope of Future Work:** Provides a summary of the work and concludes in the backdrop of the results obtained. It also discusses the future scope and possible improvements in the work, as understood from the result analysis.

LITERATURE REVIEW

Introduction: This chapter introduces the background theory of the project, including the fields of Reinforcement Learning, Multi-Agent Reinforcement Learning, and the relevance of strategy games and real-time strategy games. Further, it explains the knowledge utilised from the literature surveys in combat simulation and multi-agent reinforcement learning, which were utilised in the project.

2.1. Reinforcement Learning

Reinforcement learning(RL) concerns maximising rewards for an agent acting in an environment. The environment here can almost always be either formally defined or can be modified to be defined as a Markov Decision Process.

A Markov Decision Process(MDP)(Figure 2.1) is a Markov chain, a memoryless random process with markov states, with rewards and decisions included. An MDP can be defined as a tuple $< S, A, P, R, \gamma >$, where,

- S is a finite set of states
- A is a finite set of actions
- P is a state transition probability matrix, $P^a_{ss'} = PS_{t1} = s' | S_t = s, A_t = a$
- R is a reward function, $R^a_s = ER_{t1} | S_t = s, A_t = a$
- γ is a discount factor $\gamma \in [0, 1]$.

History is a sequence of observations, actions and rewards.

$$Ht = O_0, A_0, R_1, O_1, A_1, R_2, ..., O_t, A_t \tag{2.1}$$

i.e. all observable variables up to time t.
State is a function of the history, i.e. a function encoding the entire trajectory history.

$$S_t = fH_t \tag{2.2}$$

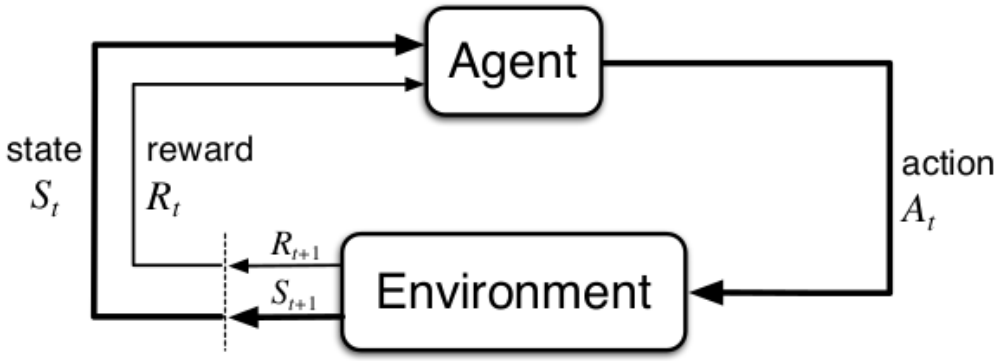

Figure 2.1: Markov Decision Process transition[18]

More specifically, a markov state is a state representation, where the present captures all relevant information from the history.

$$PS_{t1}|S_t = PS_{t1}|S_1, ..., S_t \tag{2.3}$$

In other words, when the state is given, the agent needs to focus only on the present state, in order to get information about the past states. Hence, the MDP formulation is an important aspect of any RL task, since it controls the effectiveness of the states in encoding the relevant features from the present and historical observations and actions.

The policy and the value function are essential aspects of the MDP decision-making process. The policy function π is the decision-making aspect of the agent; it selects which action or outcome is to be selected by the agent. The value function V is the expected return at a particular state; this can be used to either guide the policy or independently select actions in case of a greedy policy.

2.1.1. Model Free RL

Model-free RL is used when the MDP model is unknown or when it is known but too large to be effectively used. Hence, the value function or the policy function of an unknown MDP is optimised using model-free RL.

In model-free RL, there are two different types of learning - on-policy and off-policy. On-policy learning is when the policy is learned from experience sampled using the same policy. Off-policy learning is when the policy is learned from experience sampled from another policy. It can be compared intuitively to "looking over someone's shoulder," that is, policy learns from the experiences of another policy algorithm.

2.1.2. Model Based RL (Integrated Learning and Planning)

In Model-Based RL, a model of the environment is first approximated using samples from the environment, and then the value function or the policy function is constructed based on the type

of algorithm used.

Sample Based Planning

Sample-based planning uses the experience sampled from the environment to learn a model and samples experience from the model, which is then used to train a model-free algorithm.

Integrated Architectures

In Integrated Architecture, both simulated and real-world experience is utilised. Hence, there are two sources of experience, real experience - sampled from the environment and simulated experience - sampled from the model. The model is learned from real experience sampled from the environment, and then the value and/or the policy function is then learned from both the real and simulated experiences.

Simulation-Based Search

An extension of model-based control, where a model of the environment simulates episodes of experience, which can further be used for a look-ahead search. Since simulations would start from the present state and continue forward; hence simulation-based search focuses more on the current state, as opposed to giving equal importance to all states. This follows intuitively since, in any game, all states carry a certain weightage in determining the outcome, but at any certain timestep, the current state and nearby states have a greater influence on the outcome.

Monte Carlo Tree Search(MCTS)

MCTS[19] is a decision-time planning, rollout algorithm. Since it is a simulation-based search method, more weightage is given to the current state, in addition, to only focusing on the current and look-ahead states and ignoring all previous states. The look-ahead is performed only on certain states depending on the tree policy, hence is not as computationally expensive as other traditional tree search algorithms.

The MCTS algorithm consists of two policies - tree policy and rollout policy. The tree policy is followed for action selection when the simulation is within the tree; when the simulation encounters a leaf node, it runs the rollout policy until the end of the game. Due to the number of simulations needed, MCTS is generally effective computationally for single agent sequential decision problems if the environment model is simple enough for fast multi-step simulation.

The MCTS iteration consists of 4 steps(Figure 2.2) - Selection, Expansion, Simulation, and Backup. Selection selects the action for the current step of the iteration. The expansion phase

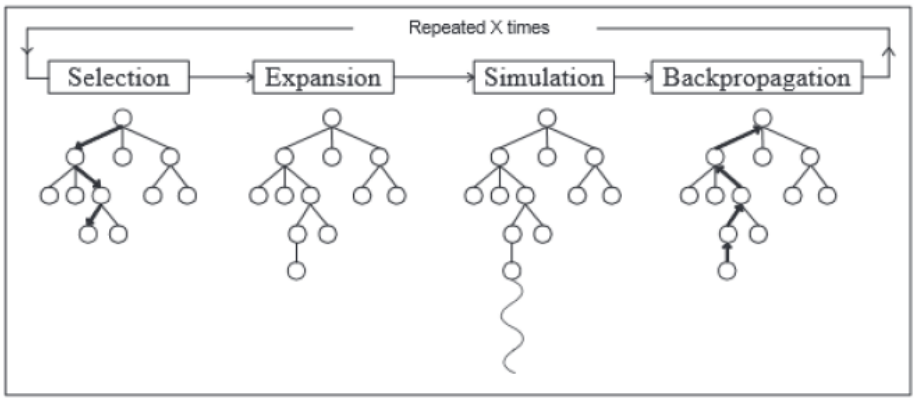

Figure 2.2: Steps of MCTS.[20]

Figure 2.3: Left: Workers mining for minerals. The minerals are resources that are useful in construction of buildings or units. Without minerals the agent cannot expand its fighting force.[21]

gets executed only when a leaf node is encountered and adds child nodes via unexplored state-action pairs, hence expanding the tree. The simulation phase simulates a step of gameplay by selecting opponent action, i.e. , simulates the 'environment.' The simulation uses the tree policy or rollout policy depending on whether the simulation is inside or outside the tree. Furthermore, in the backup phase, the return generated at the end of a set of simulations is backed up from the terminal state to the root node, and this results in an updation of the action values attached to the edges of the tree traversed by the tree policy in that particular iteration.

2.2. StarCraft-II Game

StarCraftII is a real-time strategy game, which is a proprietary of Blizzard Entertainment^{TM}. It is a partially observable real-time strategy game, where only a portion of the map, observed with scouting units, is available to the user, and one needs to preempt the opponent and make moves, as opposed to other classical strategy games such as Chess or Go. In addition, the game is real-time, where the players can continuously take decisions, where the only speed limitations are the decision-making times and the physical click rate restrictions, as opposed to the turn-by-turn alternating classic strategy games(Table 2.1).

Figure 2.4: Player constructing buildings on the map. The buildings can be storage, troop training or building centres, or simply for strategically cordoning off regions.[21]

It is considered as the most challenging strategy game since players need to perform both high-level strategy and long-term thinking as well as low-level control and planning. The game requires not only coordination, multi-tasking, map awareness, and general mechanics but also heavy memorisation, quick thinking, and split-second decision making. Because of the additional economic(Figure 2.3) and governance(Figure 2.4) related tasks, StarCraft II is more complex than most wargaming scenarios. Hence, it can be safely said that no wargaming task can be as complex as StarCraft-II.

The player takes high level economic and strategic decisions, including:

- Mining material
- Constructing units and buildings
- Scouting
- Building defences
- Attacking the opponents

They have to strike a balance between high-level economic decisions and low-level individual control of hundreds of units. Grand-master level players have around 300-600 APM (Actions Per Minute) in a professional game. For these reasons, it can be argued that StarCraft II is the ideal testbed for complex, competitive, decision-making RL tasks(Table 2.2).

StarCraft II	Chess	Go
Imperfect Information	Perfect Information	Perfect Information
$\approx 10^{26}$ action space	≈ 35 action space	≈ 270 action space
Real-Time	Alternate Turns	Alternate Turns
Economic and High Level Strategic Decisions	Low Level Combative Decisions	Low Level Combative Decisions
Hierarchical Control Structure	Direct Control	Direct Control

Table 2.1: Comparison of StarCraft II with other traditional strategy games. Economic decisions are constructing units/buildings and mining.

StarCraft II	Other RTS Games
300-600 APM at Expert Level	Speed not a Requirement (Max 80 APM)
1v1; Purely Skill Based	High Dependence on Team; Hero Selection Gamble
High Paced	Slow Paced
Extensive Multitasking	Low Multitasking
Micro and Macro Management	Low Strategic Depth
Multiple Maps with Specific Strategies and Builds	One Map with Same Strategy and Timing

Table 2.2: Comparison of StarCraft II and other RTS (Real-Time Strategy) Games like DOTA2, League of Legends, World of WarCraft etc.

StarCraftII Challenges

- Imperfect Information: The complete information is not available to the player and needs to be discovered by "scouting."
- Long Term Planning: Cause and effect is not simultaneous, as in many real-world problems.
- Real Time: The players must perform actions continually throughout the game.
- Large Action Space: StarCraft II has a combinatorial action space, where hundreds of units and buildings must be controlled at once. Actions are hierarchical and can be modified and augmented. There are approximately 1026 legal moves at every time step.
- No Single Best Strategy: Hence, an AI agent would find it extremely difficult to manage the high-level decision while micro managing each unit.

Due to these aspects of the game, it poses a new Grand Challenge for Reinforcement Learning, specifically multi-agent reinforcement learning, representing a more challenging class of problems than considered in most prior work.

StarCraftII Environments

Google DeepMind and Blizzard Entertainment have created the StarCraft II Learning Environment (SC2LE)[22], an RL environment based on the StarCraft II API. Based on the SC2LE, two major environments are present, both catering to different paradigms:

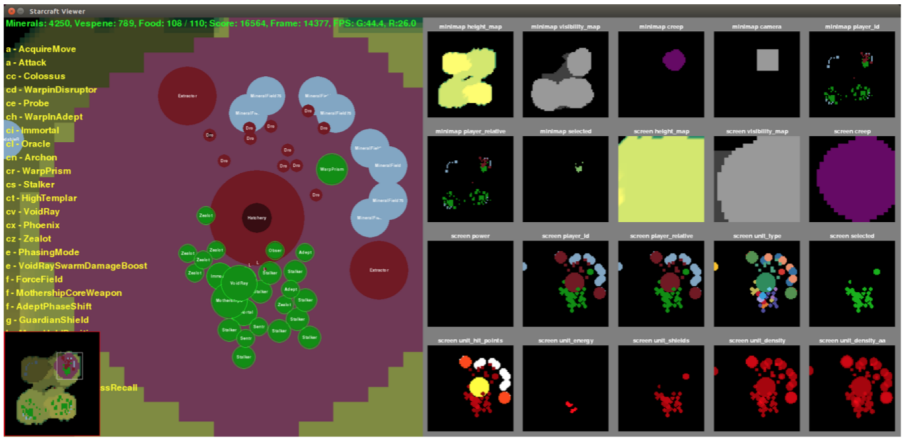

Figure 2.5: Top Left(Green): Resources present with the agent; Left (in Yellow): Queued actions; Bottom Left Box: Minimap view of the entire map; Left Half: Agent camera view; Right: Feature maps that can be used as input to agent.[22]

- PySC2[22]: Created by Google DeepMind (Python Interface). It emulates the environment of the multiplayer games in StarCraft II. It is an exact implementation of the original StarCraftII game, with two to four players depending on the map and each player controlling multiple units. It is more suited for MARL tasks utilising a super agent method, wherein one super agent controls the actions of multiple entities. In addition, it includes some simpler maps with rudimentary tasks, excluding the high-level economic decisions. The environment is a bare-bones version of StarCraftII without any of the textures and animations used to augment the human player experience, instead giving multiple channels, each focusing on specific aspects of the input in a more neural network compatible manner.

- SMAC[23]: The StarCraft Multi-Agent Challenge, which was explicitly for multi-agent research, by Whiteson Research Lab. It is built on top of the PySC2 framework and is specifically for game theoretic multi-agent reinforcement learning, i.e. , multiple separately controlled agents. Includes purely combative tasks and excludes the high-level economic tasks; hence the entities can only attack and move but not build new entities. Each entity gets an encoded input of its observation area, and there is a limited area of action, i.e. , the entity can take action on an enemy or allied entity only if it is within this area.

2.3. Combat Simulation Literature Survey

The literature survey focuses on the environments and required specifications to apply multi-agent reinforcement learning algorithms to combat simulation tasks. Further, the focus is also on the type of algorithms that have a better proven performance on command and control tasks.

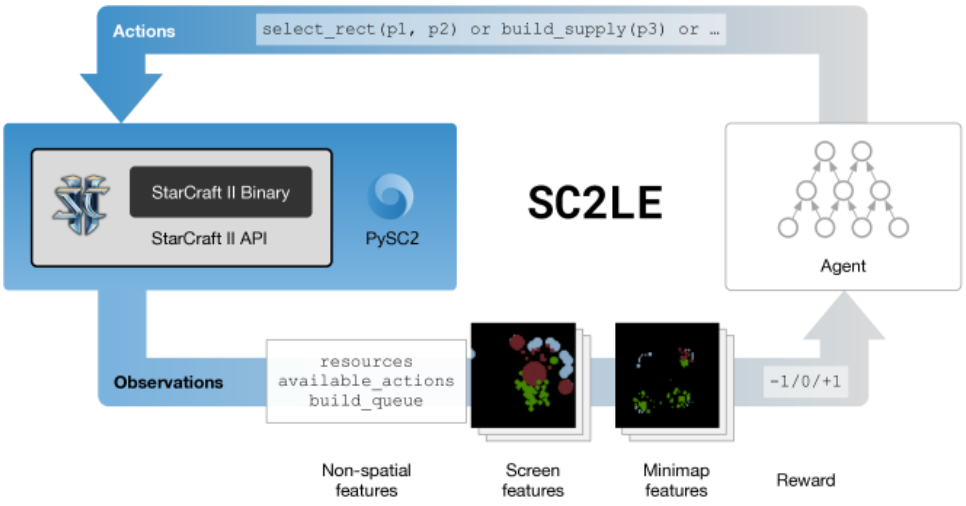

Figure 2.6: PySC2 environment[22]

Figure 2.7: SMAC: Pure combat between 2 teams of varied units. Hence, both cooperative and competitive MARL.[23]

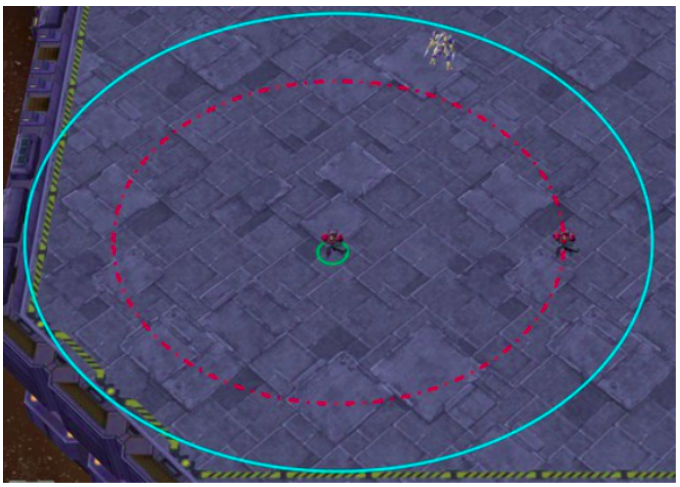

Figure 2.8: The cyan and red circles respectively border the sight and shooting range of the agent.[23]

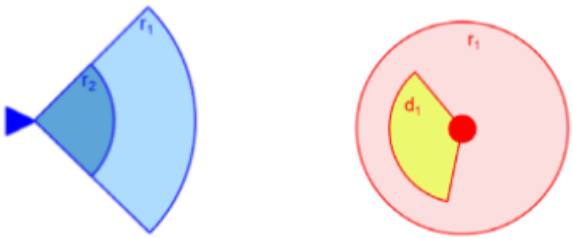

Figure 2.9: Left: Discrimination(r_1) and Decision(r_2) area for blue agent.
Right: r_1 is area where red agent is in blue agents discrimination area and d_1 is decision area for red agent.[9]

2.3.1. Battlefield Strategy from Deep Reinforcement Learning

The paper[9] discusses the benchmarks for a multi-agent environment to be able to recreate a combat environment realistically. The correct environment will be able to simulate complex battlefield scenarios and war games at a fraction of the costs of real-life experimentation. This can be effectively used to study the endgame dynamics, i.e. , the Nash equilibrium state, of any particular conflict to determine its value or the optimal policy to be utilised. Further, modifying the optimal policy can also facilitate an in-depth study into the effect of different parameters in the endgame; in other words, it can facilitate an effective "what-if" study.

The MDP is formulated as a mix of active RL and passive manual policies. The passive policy is modelled as a part of the environment, with heuristic decision-making based on more classic command and control paradigms. The active RL policy is the learning actor in this scenario.
The actor, both active and passive, needs to have certain restrictions in order to mimic a real-life situation effectively(Figure 2.9):

- A restrictive field of view (FOV), as opposed to a 360 degree one.
- Implementation of restrictive discrimination and decision areas. In effect, ensuring a partially observable and partially actionable MDP.
- Discrimination Areas - region for ascertaining whether the incoming agent is from the same or opponent team.
- Decision areas - the region around the agent, from where it may decide whether to engage the opponent or not.

For a proof of concept, the environment was tested using a policy gradient method, with a densely connected five layer neural network as the policy function approximator. This was tested on a simple scenario to move towards the specified target area with passive enemy agents

Figure 2.10: Creating StarCraft II custom map to implement the "Tiger Claw" Brigade-scale offensive operation scenario using MILSTD2525[25] standard symbols[10]

on the map. The algorithm obtained a 100% win rate after 20,000 episodes of training.

The paper effectively establishes a baseline framework to translate any multi-agent real-time strategy game environment into a testbed for command and control algorithms.

2.3.2. On games and simulators as a platform for development of artificial intelligence for command and control

The paper[10] is a case study comparing the use of the StarCraft-II game and a proprietary combat simulator OpSim[24], for command and control tasks.

Command and control is the exercise of authority and direction by a properly designated commander over assigned and attached forces to accomplish the mission.[1] Through command and control, commanders provide purpose and direction to integrate all military activities towards a common goal/mission accomplishment.

StarCraftII Case Study

The challenges posed by StarCraft II for AI algorithms make it a suitable simulation environment for wargaming, command and control, and other military applications. Especially, due to the presence of complex state and action spaces, with thousands of possible actions lasting over tens of thousands of time steps. In addition to this, the game can capture uncertainty due to partial observability. Further, the game has heterogeneous assets, which would help support agile and adaptive command and control of multi-domain forces and inherent control architecture that resembles military command and control. The StarCraft II Galaxy Map Editor can be used to make the game easily customisable to other maps and customisable symbols(Table 2.3),

[1]Army U. Army doctrine publication no. 3-0: Operations. Washington, DC: Headquarters, Department of the Army 2019;

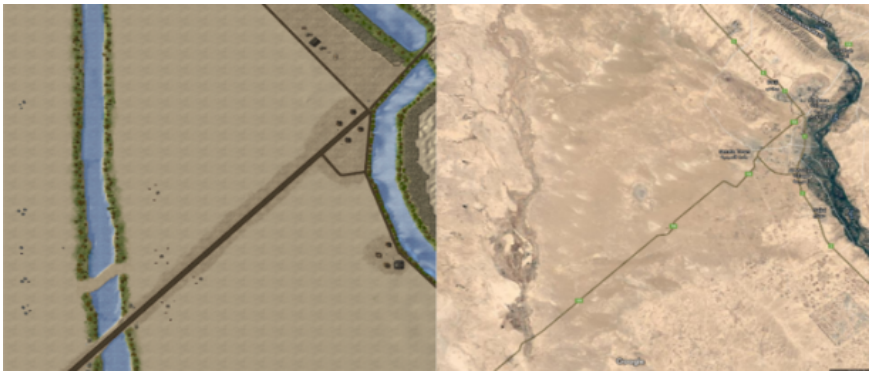

Figure 2.11: Comparison b/w StarCraft II custom created map(Left) and real world satellite imagery(Right).[10]

which has been used in this case to replicate the *"Tiger Claw"* Brigade-scale offensive operation scenario using *MILSTD2525* standard symbols(Figure 2.10).[25]

***TigerClaw* Unit**	***StarCraft II* Unit**
Armour	Siege Tank(Tank Mode)
Mechanized Infantry	Hellion
Mortar	Marauder
Aviation	Banshee
Artillery	Siege Tank(seige mode)
Anti-armour	Reaper
Infantry	Marine

Table 2.3: Mapping of "TigerClaw" units to StarCraft II units.[10]

OpSim: Reinforcement Learning using a Military Simulator

OpSim[24] is a proprietary decision support tool developed by Cole Engineering Services Inc. (CESI)^{TM} that provides planning support, mission execution monitoring, mission rehearsal, embedded training, and re-planning. OpSim also integrates with the SitaWare C4I command and control suite, a critical component in the Command Post Computing Environment; hence in this case, the practical integration and realism are more accurate. While the system is also designed to run much faster, it was not originally designed for AI applications. In this case, an OpenAI Gym interface was developed to expose the simulation state and offer simulation control to an external agent. OpSim also has an inbuilt rule based system that can be used instead of a learning system. The major disadvantage of the system is its incompatibility with learning methods and proprietary nature.

Figure 2.12: Implementation of "TigerClaw" in OpSim. Learned strategy by the Blue Force to engage only with the most lethal units while protecting vulnerable assets[10]

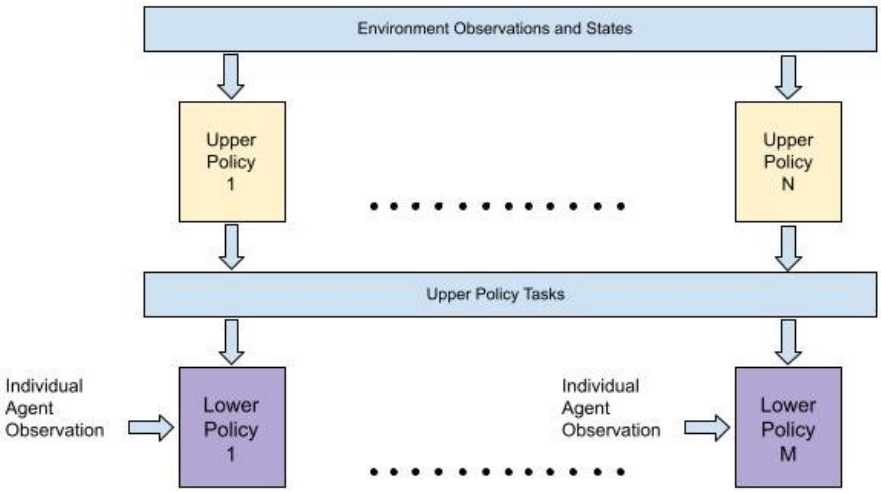

Figure 2.13: Graph abstraction and attention to ascertain connections between agents, which can further influence actions.

2.3.3. Large Scale Deep Reinforcement Learning in War Games

In this paper[26] a hierarchical policy(Figure 2.13) modeled on the hierarchy of military command and control has been implemented; further, the measurement of the performance has been via a custom made wargaming environment. The custom wargame environment has certain challenges, (similar to the StarCraftII environment, and in most combat simulations):

- Large State Space: 5000 grids, with high dimensional state space for each grid.
- Intricate Action Space: Multiple possible actions for each entity, with sub-parameters and varying sizes depending on the entity and its state.
- Partially-observed state: Partial observation for each agent.
- Long-time horizon: The effects of the decision can take multiple timesteps to show results.
- State transition instability: Multiple agents, in a non super agent framework, hence a non stationary environment.

Networks Architecture

A hierarchical network is used, with separate policies to handle different levels of control in a hierarchical manner.

- Upper Policy: Performs higher level tasks such as selecting whether to attack or build defences etc. Assigns specific types of units for the task and assigns target location.
- Lower Policy: Performs low-level, more independent tasks based on the tasks as given by the upper policy and coordinating with other friendly lower-level agents. Such as route planning to reach a certain attack location, or which specific units are to be deployed, etc.

2.4. MARL Literature Survey

Literature survey focusing on game theoretic multi-agent reinforcement learning methodologies. In the context of this project, this is applied in the case of real-time strategy games, more specifically, the SMAC environment.

2.4.1. Deep Nash Q

The paper[27] introduces a multi-agent specific reinforcement learning algorithm. This is in contrast to most other methods, where an algorithm specific to single-agent reinforcement learning is simply modified to work in a multi-agent setting. Specifically, the Nash Q algorithm, using Q learning for solving Nash equilibrium, is adapted to the deep learning paradigm. This is done keeping in mind the same advantages brought about with a shift from Q learning to Deep Q learning, i.e. state space generalisation and function approximation, which is useful particularly in high dimensional state spaces or in case of a large number of players.

In the original tabular Nash Q[28] learning paper, the major bottleneck was the repeated computation of Nash equilibrium, which is NP-hard. To tackle this, the action value (or Q value) is decomposed into the advantage function and the value function as:

$$As, a = Qs, a - Vs \tag{2.4}$$

which is the definition of the Advantage function. here,
A, is the advantage function
Q, is the action value function
V, is the state value function.

The optimality condition has been established by extending the Bellman Equation for Nash Equilibrium,

$$R_i x; \pi_i^*, \pi_{\neg i}^* = max_{u \in U_i} \{ r_i x, u, \pi_{\neg i}^* x \ \gamma_i \mathbb{E}_{x' \sim p.|x,u} R_i x'; \pi_{i,1}^*, \pi_{\neg i}^* \} \tag{2.5}$$

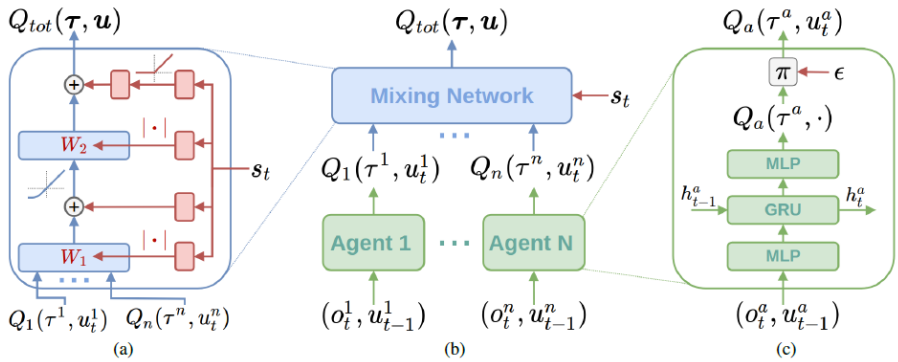

Figure 2.14: (a) Mixing network structure. In red are the hypernetworks that produce the weights and biases for mixing network layers shown in blue. (b) The overall QMIX architecture. (c) Agent network structure.[29]

where,
R_i is the expected reward per agent i
π_i is the fixed policy of agent i
$\pi_{\neg i}$ is the fixed policy off all other agents
r_i is the reward from the environment
x is the game state

Further, the bellman equation[18] is modified using the Nash operator[27],

$$\hat{V}^{\theta} x = \mathbb{N}_{u \in U} \hat{Q}^{\theta} x; u \tag{2.6}$$

here, θ is the parameters of the respective function approximators.

The game structures can be simplified based on identical preferences and label invariance assumptions. Label invariance implies that the agents do not differentiate from one another; hence the reward function must be invariant of the ordering of the agents' states or actions. Identical preferences mean that the objectives of all the agents are the same, i.e. the same performance metrics, hyperparameters and discount factors can be used. These assumptions can reduce the parameterisation by utilising single shared weights for all the agents or subgroups of agents where these assumptions hold.

This approach effectively utilises function approximators for multi-agent stochastic games with a more computationally effective Nash equilibrium calculation.

2.4.2. QMIX

The paper[29] explores a novel method for estimating the joint action value functions for all the decentralised agents as a non-linear combination. Following the central training and

decentralised execution paradigm, a centralised critic and decentralised policy are used(Figure 2.14). This has been evaluated on the StarCraftII SMAC environment. This approach takes into account the non stationarity of the environment as opposed to a super agent and has a convergence guarantee as opposed to independent Q learning, where each agent independently learns from the environment, i.e. it lies in between the extremes of IQL and centralised learning. As opposed to a Value Decomposition Network(VDN), it does not require a full decomposition of the network, hence faster, and represents a richer class of action-value functions. Here, a feed-forward neural network is used as the mixing network. The combined Q network is finally trained using the TD error of the environment.

$$L\theta = \sum_{i=1}^{b} y_i^{tot} - Q_{tot}\tau, u, s; \theta^2 \tag{2.7}$$

$$y^{tot} = r\ \gamma max_{u'} Q_{tot}\tau, u, s; \theta^- \tag{2.8}$$

The monotonicity between Q_{tot} and Q_a is ensured by,

$$\frac{\partial Q_{tot}}{\partial Q_a} \geq 0, \forall a \tag{2.9}$$

here, Q_{tot} is the mixed Q value and Q_a is the individual Q value for agent a.
This is to ensure that the global and individual argmax on the action value functions are in the same direction, i.e. the agent and mixed Q values should not be working counter to each other, operating under the assumption of cooperation.

2.4.3. Attentive Graph Neural Architectures for MARL

The paper[30] proposed for lower-level tasks, such as pure combat, in the SMAC environment. A graph attention network is utilised, encoding each agent's observation to identify which entities are closely linked(Figure 2.15). This is contrary to standard communication techniques where all the agents contribute to the decision making of every other agent. The motivation here is that the communication needs to be only the closest linked agents, which is determined by calculating the scaled graph attention.

Here, the state categorisation is based on the input state information, which is then mapped onto agents. The edges in the graph representation are identified via self attention(Figure 2.16). Finally, the distributed actions can be derived from the graph representation.

2.4.4. Tactical Planning using MCTS

This paper[31] utilizes heuristic methods, but in a hierarchical method, implemented on the *StarCraft: Brood War* environment (earlier version of StarCraftII). Here, the control has been separated, and a separate algorithm has been utilised for each level of control. The control

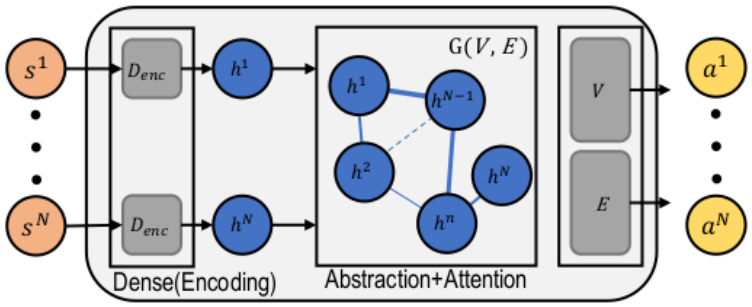

Figure 2.15: Graph abstraction and attention to ascertain connections between agents, which can further influence actions.[30]

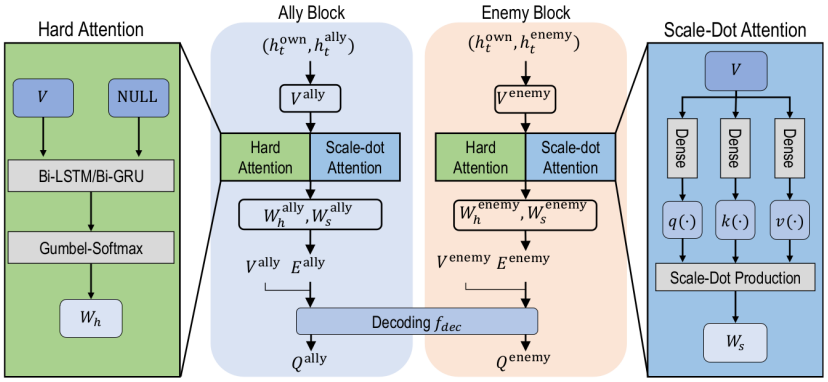

Figure 2.16: Categorized State Graph Attention Policy: Using graph attention to determine consequential connections with ally and enemy units. Hard Attention determines the existence of an edge. Scale-dot Attention determines the edge weights.[30]

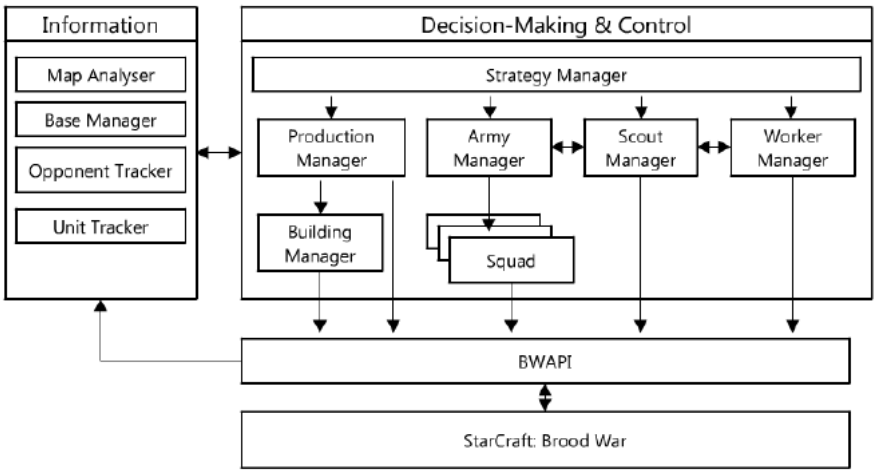

Figure 2.17: Three layered hierarchical control.[31]

separation and hierarchical network formulation are the main focus here concerning this project.

Different levels of control defined(Figure2.17):

- Strategic Control: This includes the high-level decisions of allotment of entities, engagement of entities, the composition of units, or economic decisions, such as the construction of buildings or units or mining tasks. Here, the strategic control is handled by a finite state machine.
- Tactical Control: This includes lower level engagements between units of entities; this can include deciding the route or angle of attack or whether to ambush or plan of attack. Here the tactical planning is controlled by a heuristic MCTS.
- Reactive Control: This concerns the decisions of the individual units in an engagement or any other task. This includes the lowest level control, such as which entity to attack, whether to wait for health regeneration, etc. Here the reactive control is handled by handcrafted heuristic algorithms.

2.5. General Analysis of Literature Survey

The combat simulation literature survey has pointed in the direction of inherent shared features between military simulators and real-time strategy games. This is particularly pronounced in StarCraftII with the high level of complexity and dimensionality of the game, which is shown by the comparison of StarCraftII with other real-time strategy games. Practically, this can be established by the overwhelming presence of military command and control literature, utilising StarCraftII and the advancements in this field for their analysis.

The multi-agent reinforcement learning literature review contrasts the use of separated agent controls as opposed to a single super agent, which has been the dominant strategy in solving real-time strategy games so far. This is not unlike a command and control scenario, with multiple independent decision making entities, which further draws to the conclusion that MARL is better suited for command and control tasks. It can be noticed that MARL is a heavily theoretical field with multiple legacy, tabular algorithms, which is now gaining traction with the introduction of neural function approximators into the field.
Further, the presence of hierarchical frameworks in both wargaming environment papers and StarCraft papers points to its potential. This, especially given the hierarchical structure of command and control systems.

The high level of decision making and control required, combined with low-level allotment and route planning decisions, makes StarCraftII an important grand challenge for reinforcement learning. Though less explored, the presence of the StarCraft Multi-Agent Challenge also points to this being a challenge in MARL due to the nature of the game.

2.6. Conclusion

The games of Go, Chess and StarCraftII are important benchmarks for strategy formulation and decision making algorithms. StarCraftII, in particular, is suited to test command and control algorithms and has an open source and more compatible API in most cases, as opposed to classical military simulators, making it a better testbed. Further, the inherent structure of command and control makes it better suited for utilised multi-agent algorithms, as opposed to single super agent systems, with the drawback being that multi-agent systems have a lesser convergence and being a more theoretical and less practically explored domain.

METHODOLOGY

Introduction: This chapter explains the implementational details of the projects implemented, AlphaZero, AlphaStar and Hierarchical Network. The AlphaZero implementation is the same as the paper due to the presence of resources and the computationally feasible requirements of the algorithm. In the case of AlphaStar, the chapter explains the original algorithm, then explains the mini-AlphaStar algorithm which has been implemented and explains the points of difference between the two. Finally, the Hierarchical Network applied on the SMAC environment is explained.

3.1. AlphaGo Series

Go is an ancient Chinese board game that has been considered the next milestone for artificial intelligence research after chess. However, as opposed to chess, exhaustive search algorithms could not be used here simply due to the extensive breadth and depth of the game as compared to chess. Therefore the standard approach of exhaustive tree search used successfully in other games of chess, Othello, checkers etc. , could not be used in go. The AlphaGo series, starting with AlphaGo, was the first algorithm to achieve superhuman performance in the game. The algorithms use a combination of deep neural networks and tree search to achieve this performance, specifically utilizing a novel version of MCTS with deep neural networks.

3.1.1. AlphaGo

AlphaGo[7] algorithm uses Asynchronous Policy and Value MCTS (APV-MCTS) - where the monte-Carlo tree is guided by a policy and value function with function approximation by deep convolution networks. The algorithm contains four function approximators - Rollout Policy, SL Policy Network, RL Policy Network, and Value Network. The SL policy network is a deep neural network trained on expert human moves. This network is used as the tree policy in the MCTS. The RL policy network is the architecture of the SL policy network and is initialized with the same weights. It is then trained on self play games. The RL policy network is used only for training the value network and not as a policy during play. The value network also has the same architecture as the SL policy network but has a single out-

put unit that gives an estimated value of the game position and contains an additional binary input feature for the current colour to play. The value network is initialized with the same backbone weights as the SL Policy network. Further, it is trained along with the RL policy network using policy gradient reinforcement learning. The rollout policy is a simple linear model trained on the same expert moves data as the SL policy network. This is done since the rollout policy needs to be fast to run fast multistep simulations till the end of the game.

The APV-MCTS is an extension of the MCTS algorithm. The expansion and simulation phases progress as in MCTS. In the selection phase, the action values are combined with a decaying version of the prior probabilities from the SL policy network (a variant of the Polynomial Upper Confidence Trees algorithm), which are used to make an action selection.

$$A_t = argmax_a Qs_t, a\ us_t, a \tag{3.1}$$

$$us, a = C_{puct} Ps, a \frac{\sqrt{{}_b N_r s, b}}{1\ N_r s, a} \tag{3.2}$$

Here, C_{puct} determines exploration level. Where, $us, a \propto \frac{Ps,a}{1Ns,a}$. P(s,a) - SL Network prior probabilities. Hence, the decaying prior probabilities and reliance of Q value increases exploration. In the backup phase, unlike in MCTS the rollout return is calculated as a combination of the value network and the cumulative reward rather than being solely based on the later.

$$vs = 1 - \eta v_\theta s\ \eta G \tag{3.3}$$

η controls the mixing of the values from the two evaluation methods. v_θ is trained with the rollout game results as the target.

3.1.2. AlphaGo Zero

AlphaGo Zero[14] algorithm makes a few changes to the AlphaGo algorithm and surpasses its performance solely by self play, without training on any human expert plays.

Comparison to AlphaGo

- Input gives only raw descriptions of the stone positions; the handcrafted features are not used, as in AlphaGo.
- MCTS is used during both training and live play, whereas AlphaGo used MCTS only during training.
- No SL Policy Network is used, and RL Policy Network weights are randomly initialized.
- AlphaGo Zero's MCTS does not include rollouts, and simulations end at leaf nodes.

- Policy network and Value network have been unified.
- Input gives only raw descriptions of the stone positions, i.e. not including the handcrafted features in AlphaGo.
- MCTS is used during both training and live play, whereas AlphaGo used MCTS only during training.
- No SL Policy Network is used, and RL Policy Network weights are randomly initialized.
- AlphaGo Zero's MCTS does not include rollouts, and simulations end at leaf nodes.
- Policy network and Value network have been unified.

A unified neural network is used, consisting of the policy and value networks. The input is a $19 \times 19 \times 17$ image stack, consisting of one feature plane for current board positions and seven for past configurations, and similarly for the opposing player. An additional input plane was included indicating the colour of the current play. The network is "Two-headed", where a common backbone architecture splits up and leads to two "heads" terminating at two output units, one corresponding to the policy function and one to the value function(Figure 3.1). A single board position is not a Markov state; hence previous positions, up to eight timesteps are also included. In addition, the output includes an additional unit for a pass move to be included. This network also makes use of skip connections to implement residual learning.

In AlphaGo Zero, the self play reinforcement learning is used for training the entire unified network. The games of self-play also utilize MCTS, as opposed to AlphaGo, where MCTS is not used during training and only for the gameplay. The algorithm plays against a frozen version of itself that is updated every 1,000 training steps. The neural network is then trained with the output probability vectors beings guided towards MCTS probabilities π, and the value function outputs being guided towards the self play result z. Extra noise was added to network outputs to encourage exploration. A single iteration per game was selected for training in order to avoid overfitting due to the high correlation among iterations within the same self play game.

3.1.3. AlphaZero

AlphaZero[15] further extends the AlphaGo Zero to generalize the architecture for gameplay of Go, Chess and Shogi. In addition, it also improves upon the performance of the previous algorithms.

Comparison to AlphaGo Zero

- Takes account of draw scenarios possible in chess and shogi, rather than solely win or loss.

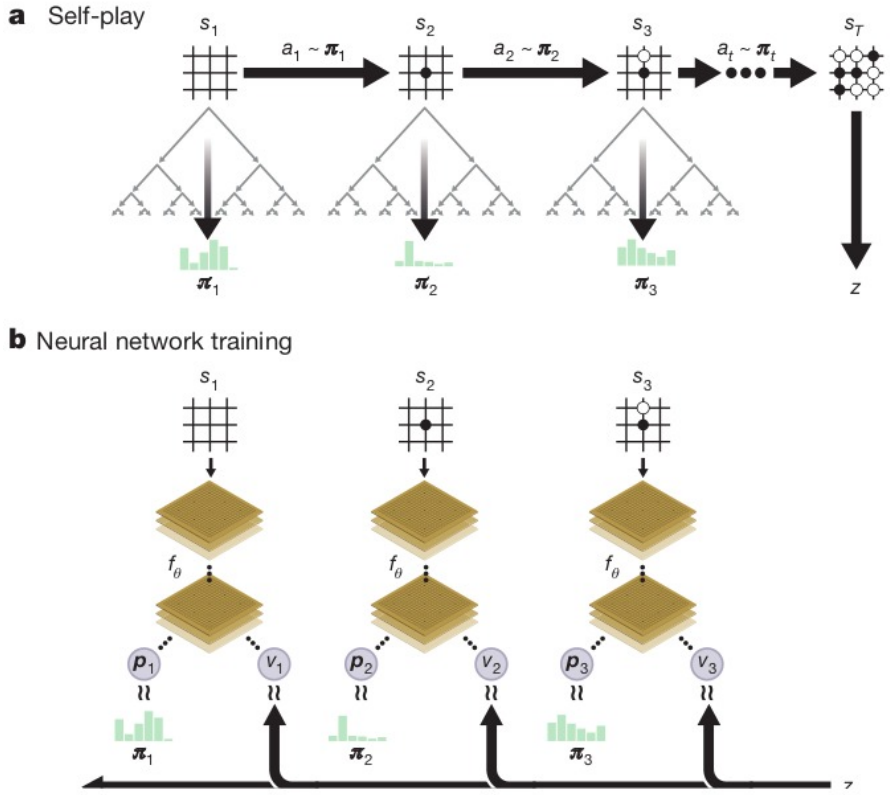

Figure 3.1: **AlphaGo Zero Training Routine**: Working of the a. Self-Play and b. Training algorithm - using the self play outcomes and guiding the model towards the APV-MCTS outputs.[14]

- Rules of chess and shogi are asymmetric; hence symmetry transforms are not applied to the input data, as in AlphaGo or AlphaGo Zero.
- AlphaZero maintains a single neural network that is continually updated, rather than maintaining versions that are evaluated and the best version selected every 'n' steps.
- Hyperparameters tuned using bayesian optimization that is game agnostic, with the exception that the noise added to the prior probabilities is scaled in proportion to the typical number of legal moves.
- A separate instance of AlphaZero was trained for every game.

The architecture remains the same across games except for input and output features, with an input - $N \times N \times MT\ L$. Where $N \times N$ is the size of the board. T is the number of timesteps of history to be included in the input. M is the number of planes per time step. N and M are game-specific; for example, N is 19 in Go and 8 in chess, and M is chess consists of one feature plane for each piece type. Compared to more traditional algorithms in each game, the use of MCTS can average out over the approximation errors introduced by the network, as opposed to alpha-beta search, which uses minimax, which propagates the largest errors back to the node.

3.2. AlphaStar

As opposed to Go, StarCraft II is an imperfect information, real-time strategy game wherein only a portion of the environment is visible to the agent, and all agents take actions simultaneously. In addition to this, the presence of combinatorial action space, and a planning horizon that exists over thousands of real-time decisions, makes it highly unsuited to a simulations based search algorithm like in the AlphaGo series. Hence, in this case, a model-free approach is taken. The AlphaStar[8] paper is also the first to achieve expert performance, with reasonable restrictions such as the Actions Per Minute, or restricting the field of view, to make it more favourable in a match against human players.

3.2.1. MDP Formulation

Formulation of the Markov Decision Process for this environment, w.r.t. the AlphaStar algorithm. In other words, a description of the environment interface, i.e. the observations and the reward formats.

Input Features

- **Baseline Features**: Various observations from self as well as the opponent agent. Utilized only in value function training.

- **Scalar Features**: the statistical information of the game, e.g., the current minerals or supply of the game
- **Entities**: List and details of all the entities under the agent and details of opponent entities in the field of observation of agent entities.
- **Minimap**: The Minimap gives an overview of the entire map, but without all enemy entity details, only observed ones.
- Observation is not Markovian; hence LSTM is used to encode history.

Reward

- Match outcome is used as a terminal reward, r_T. (-1 on a loss, 0 on a draw and 1 on a win)
- No discount is used to accurately reflect the true goal of winning.
- Pseudo-reward is used to make the agent follow human replay statistics, z, for better exploration.
- The KL divergence between the supervised and current policy is continually minimized, i.e. distillation loss.

3.2.2. Policy Network

Alphastar uses a neural network with parameters θ as its policy network(Figure 3.2).

$$\pi_\theta a_t | s_t, z = P a_t | s_t, z \tag{3.4}$$

Here, s_t is the current state; a_t is the current action to be taken; and z is the human expert replay statistics.

This network receives all observations, $s_t = o_{1:t}, a_{1:t-1}$ as inputs, and selects actions for selected entities as outputs. The network is first trained on a dataset of 967,000 expert human replays. Further, even during the reinforcement learning play, the model is provided with an additional reward for mimicking certain statistics from the human expert database, such as build order. This is done to encourage exploration since the game requires a very complex strategy; hence a simple ε exploration strategy will be unable to generate any robust exploration mechanisms. So, in such a case, closely following human expert moves enables more exploration and brings in a certain level of robustness.

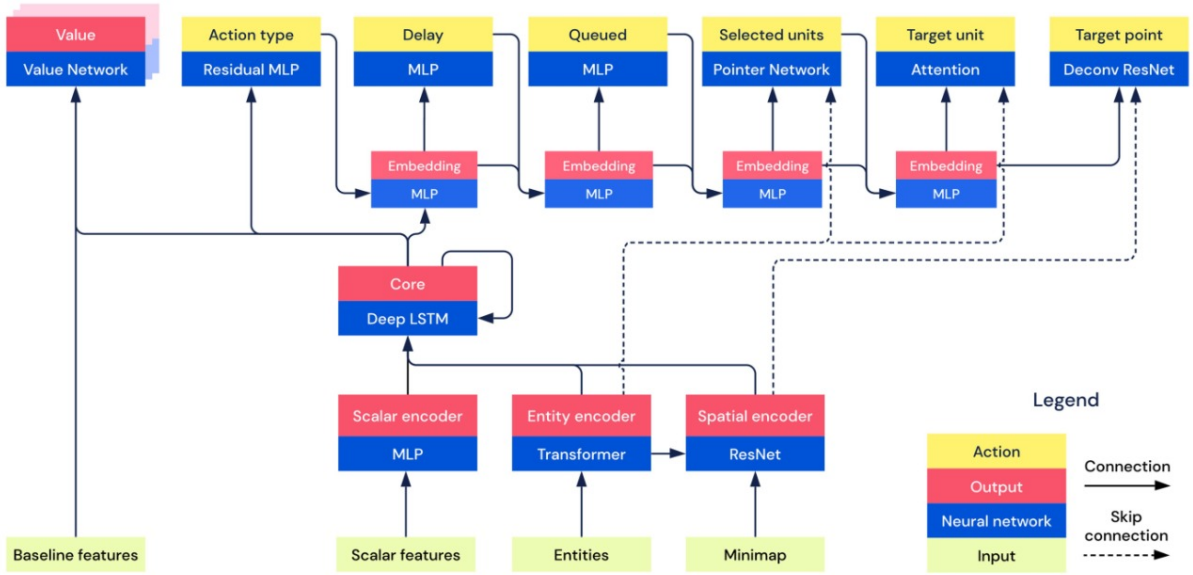

Figure 3.2: **AlphaStar Policy Network**: Policy network architecture used in the AlphaStar algorithm[8]

3.2.3. League Training

The multi-agent training method uses a novel algorithm called league training(Figure 3.3). Here the league is populated with three distinct types of agents - main agent, main exploiter and league exploiter. The main agent is trained using prioritized fictitious self-play, where it plays against present and past versions of itself, prioritized by the win rate. The parameters are then updated from the games using actor-critic reinforcement learning. The main exploiters play only against the latest iteration of main agents, and the main purpose is to identify exploits in the strategy of the main agents, thereby encouraging the main agents to address these vulnerabilities. The league exploiter agents use a similar training mechanism as the main agents but are not targeted by the main exploiters. Hence, their main purpose is to find potential exploits in the entire league. The main and league exploiters are periodically reinitialized to the SL weights in order to encourage more diversity and exploration.

3.2.4. AlphaStar Computational Complexity

A significant driver of the success of the algorithm has been the sheer amount of computation used for the training. For the original algorithm, Google DeepMind used 971,000 replays $\approx$ 38 TB. The replays are encoded in the SC2Replay format, which requires a game to be run every time the file is accessed. Hence it needs to be converted into an accessible pickle format. For training of the league, 16,000 concurrent StarCraft games have been run, roughly using 150 processors with 28 physical cores each, and 32 128-core Tensor Processing Units (TPU)[32] were used. Hence, the computation power of $\approx$ 120 Graphical Processing Units (GPU) were

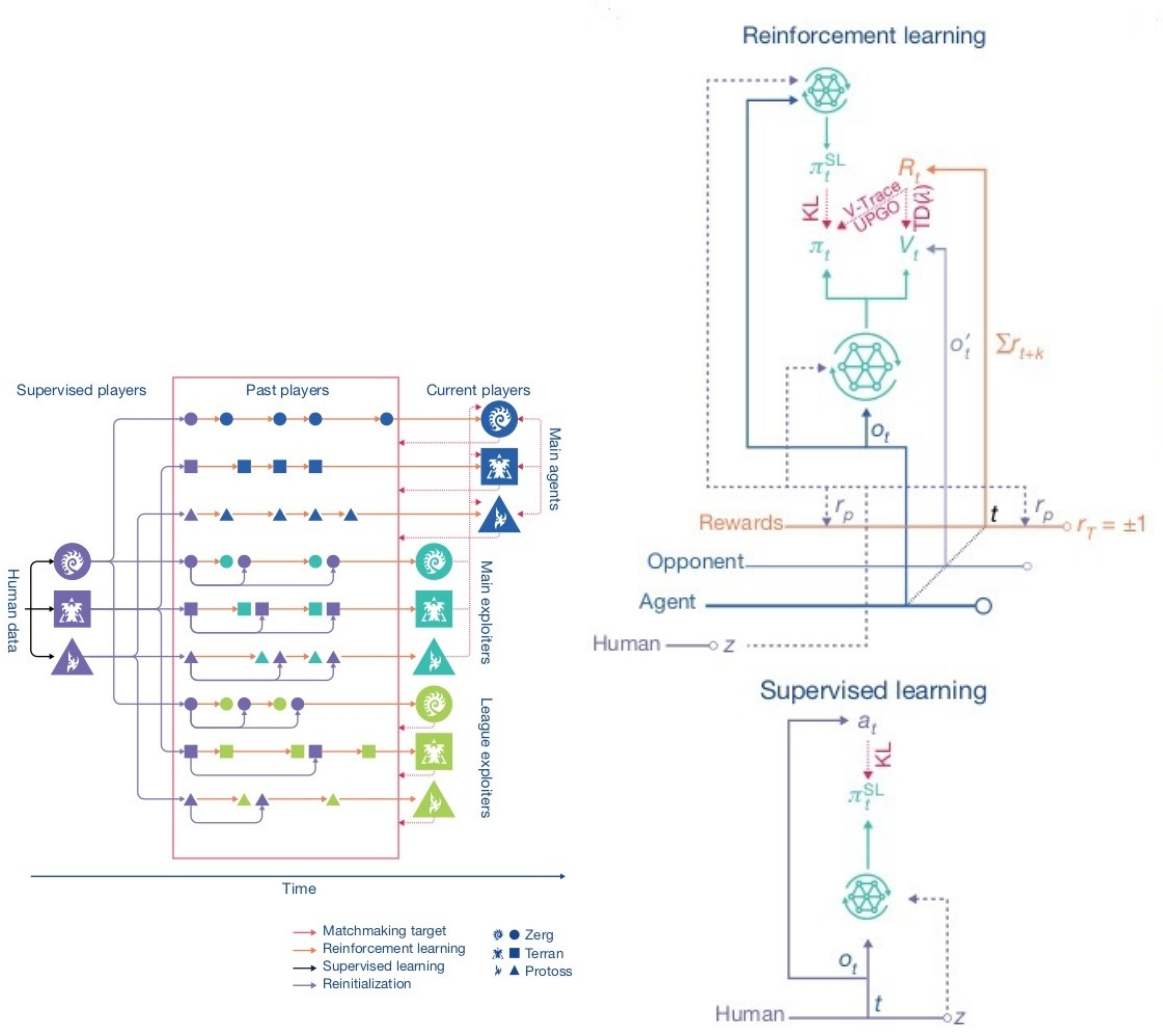

Figure 3.3: **Multi-agent League Training**: Consists of 3 types of agents as shown, the *main agent*, *main exploiter*, *league exploiter*.[8]

utilized for the league training process.

3.3. mini-AlphaStar

Mini-AlphaStar[17] is a computationally lighter version of AlphaStar that can be run on a commercially available server (NVIDIA DGX server), as opposed to the 32 128-core TPUs used by the original algorithm. Since the algorithm has been minimized, the capacity is only to attempt gameplay against the inbuilt bots/Cheater AI(Table 3.2). The algorithm is run against the inbuilt bot, using the IMPALA Actor Learner algorithm[33] as opposed to a self play paradigm.(Figure 3.4)

The preprocessing of the human replay data into a machine-readable format takes approximately 24×7 hours. The supervised learning runs for 100 epochs on the 19,000 replays, which took 36 hours to run. The reinforcement learning phase, where the mini-AlphaStar agent is

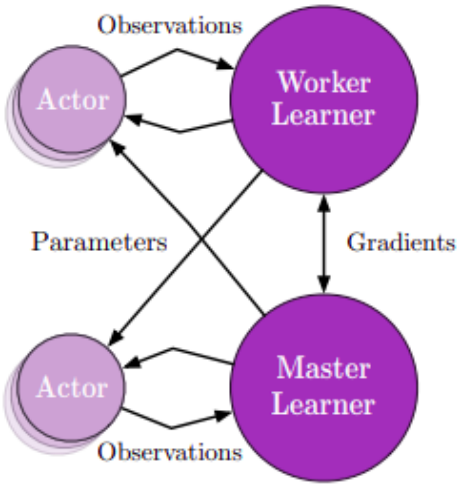

Figure 3.4: Actor learner architecture with multiple synchronous learners, with policy parameters distributed across the learners. Used in mini-AlphaStar against inbuilt bot[33]

being trained, may take 1000 hours. In addition, due to the sheer size and complexity, any initialization errors or issues of non-convergence will come to light at a very slow pace during the training, and each being very time intensive to correct, test and re-align.

Modules	**Content**
alphastarmini.core.arch	Model architecture, similar to the AlphaStar, but scaled down
alphastarmini.core.sl	Supervised Imitation Learning training module
alphastarmini.core.rl	Reinforcement Learning training module, implementation of IMPALA algorithm
alphastarmini.lib	lib functions
alphastarmini.third	third party functions

Table 3.1: mini-AlphaStar Implementation module specifications[17]

3.4. Hierarchical Multi-Agent Network

The implementation of a hierarchical network with a single upper policy and multiple lower policies. Hence, utilizing the effectiveness of both the centralized super agent and the computational efficiency of the decentralized multi-agent paradigm. This algorithm is inspired by Tactical Planning using MCTS for StarCraft[31].

The environment used for testing is the SMAC environment, hence using a purely combative multi-agent environment. Here the maps are divided into multiple difficulty levels, with the performances as shown in Table 3.3.

The upper policy is a single agent taking inputs from all the lower agents. There is no direct input or output from the environment to this policy. The input is a set of encoded vectors

	AlphaStar	**mini-AlphaStar**
Races	Protoss, Zerg, Terran	Protoss
Human Replay Size	971,000 $\approx$ 38TB	19,000 $\approx$ 750GB
Computation	$\approx$ 32 TPUs	$\approx$ 7 GPUs
Batch Size	512	96
Max Entities	64	32
Minimap Size	128	64
Embedding Size	3585	1543
Map Channels	18	18
LSTM Hidden Dim	384	128
LSTM Layers	3	1
League Learner Num	12	4
ActorLoop Num	16000	512

Table 3.2: Comparison of network architecture of AlphaStar and mini-AlphaStar[17]

	IQL	**COMA**	**VDN**	**QMIX**	**Heuristic**
2s_vs_1sc	100	98	100	100	0
2s3z	75	43	97	99	90
3s5z	10	1	84	97	42
1c3s5z	21	31	91	97	81
10m_vs_11m	34	7	97	97	12
2c_vs_64zg	7	0	21	58	0
bane_vs_bane	99	64	94	85	43
5m_vs_6m	49	1	70	70	0
3s_vs_5z	45	0	91	87	0
3s5z_vs_3s6z	0	0	2	2	0
6h_vs_8z	0	0	0	3	0
27m_vs_30m	0	0	0	49	0
MMM2	0	0	1	69	0
corridor	0	0	0	1	0

Table 3.3: Test win rate of different baseline algorithms on different maps on the SMAC environment.[23]

supplied by each lower policy, which is used as a formulation of the environment state. The output of the upper policy is a set of command vectors which guide the lower policies. To ensure that each level of the hierarchy is dependent on the other and not acting as separate entities, the layers receiving the encoded vectors from the other layers are set without bias, making it harder for each network to ignore the inter-layer inputs.

The lower policies are multiple individual agents that get their respective inputs from the environment and give individual actions. Their field of view is limited, and they can view only their nearest agents or enemies; likewise, their actionable range is also limited. Hence, they must be close enough to an enemy to take action. The command vectors from the upper policy are meant to guide them towards the team goal for the tactical level of combat while leaving them free to achieve short-term goals in reactive level combat. Along with the actions, the network also returns a response vector, which in its communication to the upper agent. The lower policy weights are shared between all the agents since the environment gives control only to one team, with the other team being inbuilt game bot controlled. Since the agents are all acting cooperatively, with a single objective and similar unit type, hence the label invariance and identical preference assumptions can be applied.

The algorithm used is asynchronous actor-critic[34], with the upper policy being guided by a sparse win rate, with a scaled down environment reward for giving a general direction. The lower policy is guided by the environment returned reward.(Algorithm 1) The exploration is executed by adding random noise to the parameter space at intervals[35].

Modules	**Content**
Architecture	Architecture modules, for lower upper and combined hierarchical policy
Env	Environment wrappers for different maps in the SMAC environment.
AsyncWorker	Asynchronous Actor Critic Worker, running the training workers
Ray_tune	Benchmarking the environment against decentralized multi agent networks.
Stable_Baseline	Benchmarking the environment against super single agent networks.

Table 3.4: Hierarchical Network Implementation module specifications

3.5. Tools Used

- **Python**: Language used for the project. Used in both mini-AlphaStar and Hierarchical Network.
- **PyTorch**:[36] Deep learning framework used to architect the function approximators of the actor-critic networks and for the IMPALA based super agent in mini-AlphaStar.
- **PySC2**:[22] The StarCraftII environment for RL. Similar to the original game and suitable

Algorithm 1 Asynchronous Actor Critic Hierarchical Algorithm

Initialize lower and upper policy actor and critic networks with random weights
while $Game\_Idx \leq N\_GAMES$ **do**
  `Load global network weights to local network`
  `Reset environment`
  for `each` $agent \in 1, n\_agents$ **do**
    $STEP \leftarrow 0$
    while Game not Done **do**
      $action, response \leftarrow local\_action\_critic$
      $reward, Done, info \leftarrow environment$
      $lower\_reward \leftarrow reward$
      $upper\_reward \leftarrow environment\_win\_rate$
      `local_actor_critic remember state, action, response, lower_reward, upper_reward`
      if $STEP\%T$ **then** ▷ Every T timesteps train the global network
        `Transfer grads from local network to global network`
        `Update global parameters using local buffer, lower reward for lower policy and upper reward for upper policy`
        `Load global network weights to local network`
        `Clear local network memory`
      end if
      $STEP = 1$
    end while
  end for
end while

for a super agent format. Used in both the original AlphaStar and mini-AlphaStar algorithms. Interacts with the game via the S2Protocol.

- **SMAC**:[23] StarCraftII environment for multi-agent tasks. Removes the high-level economic decisions, hence a purely combative multi-agent environment. Used for the Hierarchical Network.
- **PettingZoo**:[37] Used for environment wrapping of SMAC, for ease of usage, and compatibility with Ray and Stable Baselines.
- **Stable-baselines3**:[38] Used to set the benchmark in SMAC with super agent format networks.
- **Ray RLLib**:[39] Used to set the benchmark in SMAC with decentralized multi-agent networks.
- **TensorBoard**:[40] Used to track performance in both mini-AlphaStar and Hierarchical Network. Also, used to produce the result graphs in the results section.
- **NVIDIA DGX Server**: GPU server, mainly used for training mini-AlphaStar, due to the high computational complexity of the algorithm and the StarCraftII game simulator.

3.6. Conclusion

The computationally efficient version of the AlphaStar algorithm, mini-AlphaStar, has been implemented. This has the same architecture as AlphaStar, but with scaled-down network size and hyperparameters. The Hierarchical Network has also been implemented on the SMAC environment for a pure combative decentralized MARL testbed. This will be inherently more similar to a command and control paradigm.

RESULTS AND DISCUSSIONS

Introduction: This chapter lists and explains the achieved results in the two main experiments of the project, mini-AlphaStar and Hierarchical MARL Network. The discussion is focused on the achieved results, the shortcomings and whether there is any scope for improvement or if the learning is saturated in each case. There is also a discussion on the shortcomings of the original AlphaStar algorithm, specifically w.r.t this project, focusing more on the strategy of the resulting policy.

4.1. Rethinking the AlphaStar Algorithm

Defects in AlphaStar, as analyzed from replays of games between AlphaStar and human expert[41]:

- Easy Exploitation: Slight variations in common human tactics can easily exploit the system. For example, a slight modification of the standard "Cannon Rush" strategy can exploit the system, which repeatedly deploys the standard defensive strategy used against "Cannon Rush."(Figure 4.2)

- Lack of Reasoning: The algorithm does not seem to effectively reason and use units that can specifically counter other units. The algorithm uses induction to learn its generalized policy and lacks the ability of deduction. For example, the algorithm uses lurkers, invisible units which are used to scout the opponent's territory to attack, which gives away its location, negating the lurker units' invisibility factor.

- Dull Strategy: Seems to exhibit a single strategy that is the best response to a non-uniform mixture of policies, i.e. a lack of novelty. The policy learned by AlphaStar seems to be a standard average policy of the human expert replay datasets. There is no scope for learning new and novel strategies about the game of StarCraftII from replays of AlphaStar, as in the case of the AlphaGo series.(Figure 4.1)

- No Planning: There is no planning module; hence there is no significant planning for future moves. This can be seen in the repetition of strategies, verbatim without modification, that are proven to be ineffective against a particular opponent. In effect, the algorithm

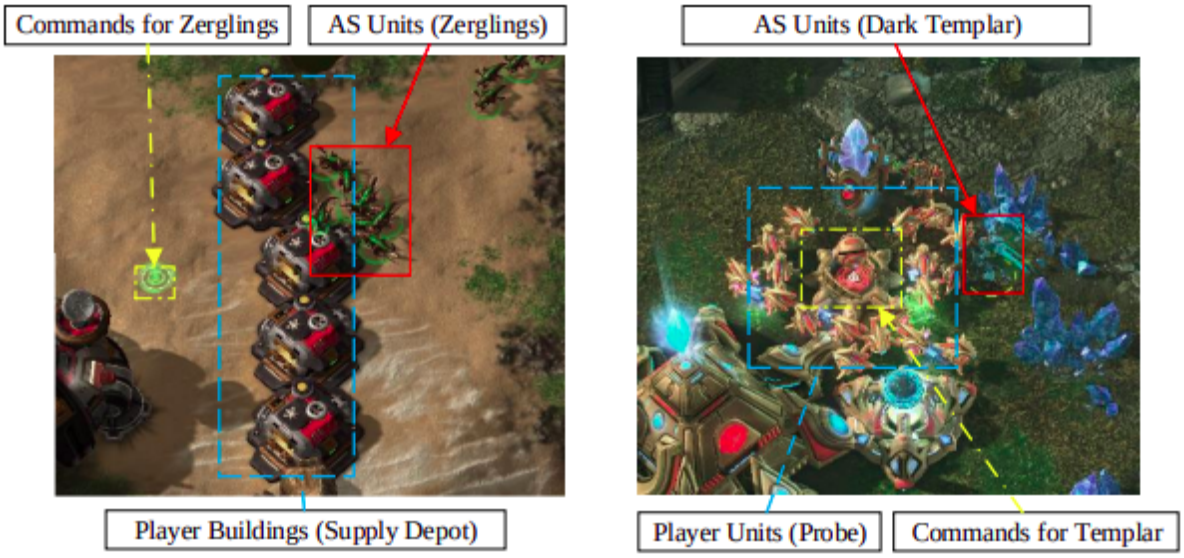

Figure 4.1: Left: Zerglings want to go in the buildings. Right: Dark Templars want to attack the Cannon.[41]

Figure 4.2: Left: Human player uses Cannon Rush to defeat AlphaStar. Right: Human player uses Lurkers to beat back AlphaStar.[41]

repeats the same strategy irrespective of whether the opponent has devised an effective counter-strategy.

Most human player testaments mentioned that the agent seemed good but beatable, unlike in AlphaGo, where it seemed supernatural. This attests to the high level of complexity of the StarCraftII game, as opposed to more classical strategy games or other real-time strategy games.

4.2. mini-AlphaStar

The algorithm is trained using the Importance Weighted Actor Learner Architecture (IMPALA) algorithm against the inbuilt bot instead of a self-play league training algorithm since the mini scaled version does not have the capacity to achieve the human skill level as the original AlphaStar algorithm. The performance of the supervised model is a 42.5% win rate. On running

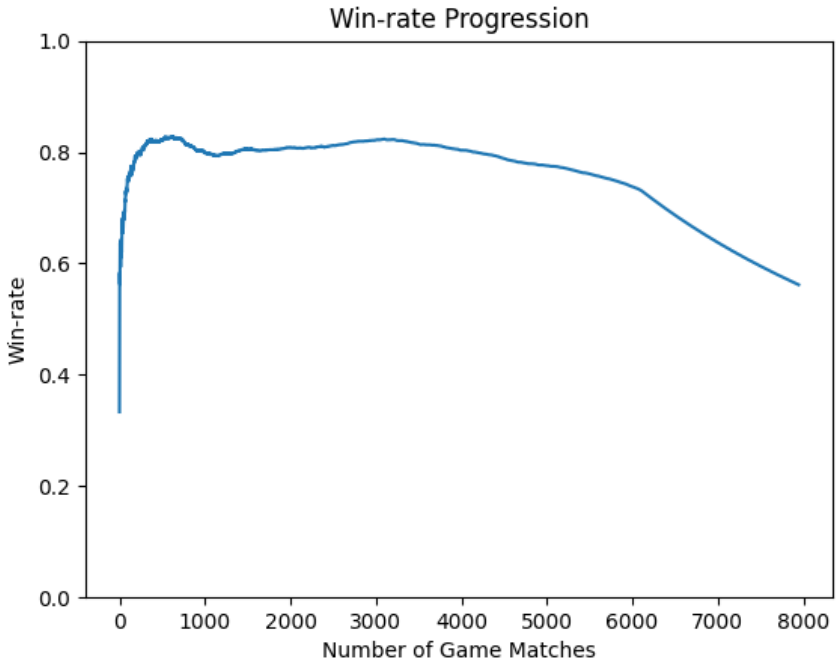

Figure 4.3: Running average win rate of mini-AlphaStar against the inbuilt level 1 difficulty bot.

the reinforcement learning algorithm, the win rate increases to 83% and stabilizes at ≈55% win rate against the level 1 difficulty inbuilt bot(Figure 4.3).

The original AlphaStar algorithm gave a 99.8% win rate against human expert players, with the fully trained self play league training algorithm. This has significantly been reduced due to the curtailed model capacity to match the computational limits. Further, it can be seen from the win rate plot that the reinforcement learning performance slowly drops. This is a testament to the game's complexity, for which reason the original algorithm continuously used the KL divergence loss to prevent the RL policy from diverging too much from the SL policy. Since the complexity of the game requires a very complex exploration strategy to approximate a majority of the state spaces and strategies, the imitation learning algorithm is the best available methodology for the same.

4.3. Hierarchical Multi-Agent Network

The algorithm gives a cyclical performance, as seen from the number of dead allies vs the number of dead enemies plot and the environment reward plot. The win rate of the network peaks at 1% on the '10m_vs_11m' map. This can be the result of the algorithm getting stuck in a cycle, such as in the "Rock Paper Scissors" game, where is strategy A beats strategy B, B beats C, but A beats C.

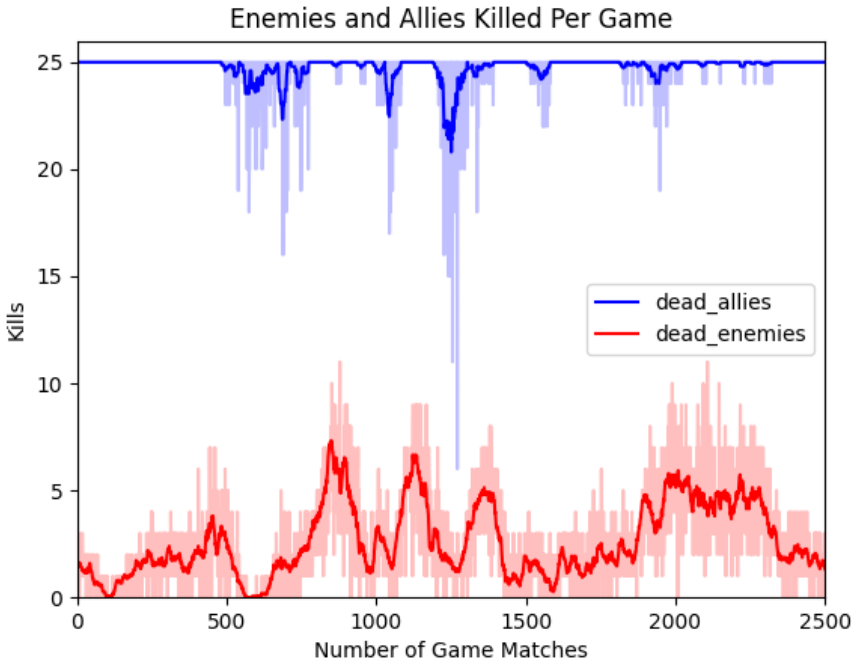

Figure 4.4: Number of kills v/s game iteration. Red line represents the number of dead enemies and blue line represents the number of dead allies.

An analysis of the games shows the following learning pattern(Figure 4.4 & Figure 4.5):

1. The agents learn to trigger their attack mechanisms and soon, permanently are on attack mode since the chance of random rewards is higher. This is encouraged by the fact that there is no negative reward for attacking an ally.

2. The agents learn to focus their line of fire onto enemy agents within their attack range. Hence, the number of dead enemies increases while the number of dead allies remains at the maximum, as in the case of a random policy.

3. The agents then learn to evade enemy fire but, in the process, lose their ability to focus fire on enemy units. Hence, it can be seen that the number of dead allies reduces along with the number of dead enemies.

As it can be seen, the algorithm gets stuck in a cycle, which could be due to a lack of differentiation of the contribution of each agent to the environment's rewards. This can be improved by utilizing a centralized critic using a mixing algorithm.

4.4. Result Analysis & Conclusion

The mini-AlphaStar algorithm is able to give an average of 55% win rate after 8000 games. The win rate improves after the supervised learning phase, which then slowly reduces with iterations. This is due to the divergence of the supervised policy from the reinforcement learning policy,

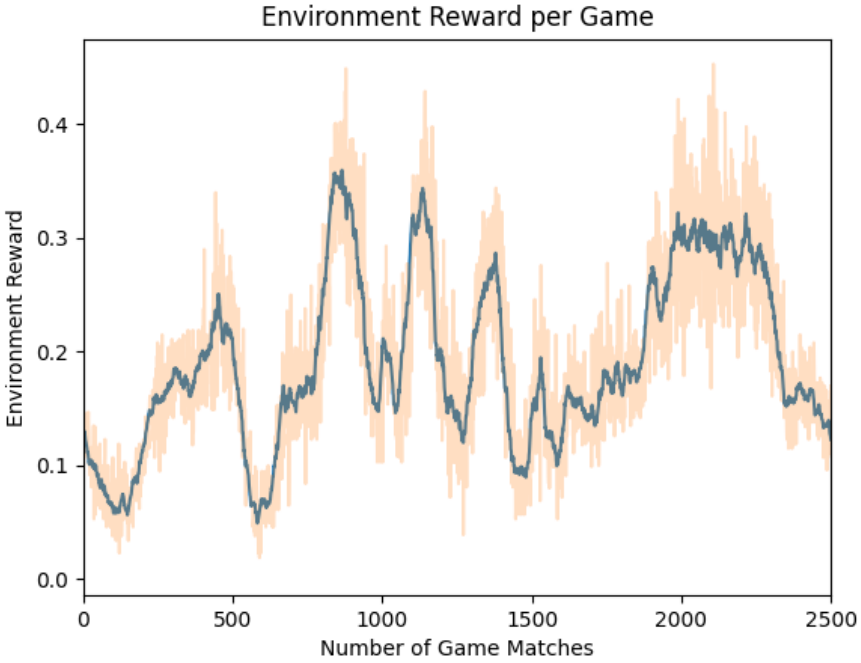

Figure 4.5: Game accumulated reward v/s game iteration.

which the original algorithm specifically handled using a Kullback-Leibler (KL) divergence loss. This cannot be implemented due to computational constraints. For the capacity of the given network, the 55% win rate is the maximum possible performance which has been achieved here.

The hierarchical MARL network has a cyclical performance, wherein it learns a strategy and re-learns an entirely new strategy which is weakly dominated by the previous strategy. This could be due to the use of combined rewards; hence every agent gets the same reward for the actions of a few agents.

The mini-AlphaStar maximum performance is achieved against the level 1 inbuilt game bot. The hierarchical implementation on the SMAC environment has been done, but there remains scope for future improvement in the form of centralized critic, such as QMIX or Value Decomposition Networks (VDN).[42]

CONCLUSION AND SCOPE OF FUTURE WORK

Introduction: This chapter concludes, and summarizes, the project. Further, there is a discussion on the scope of future work, of both mini-AlphaStar and Hierarchical MARL Network, in the project. There is also a discussion on the generic methodology followed in the implementations in the project.

5.1. Brief Summary of the Work

The project's focus was on implementing single, and multi-agent reinforcement learning focused on real-time strategy games. The implementation is in three stages: AlphaZero, mini-AlphaStar, and Hierarchical MARL Network. Further, the case is made for using StarCraftII as an effective test bed for AI command and control algorithms. The implementation included both model-based and model-free methods, with the focus being more on model-free methods due to the heavy computational demands of the environments used.

The general methodology followed for the project:

- **Game Interface**: Deciding the game environment to be utilized, perfect or imperfect information, and accordingly using the corresponding game engine or reinforcement learning environment modules, such as OpenAI Gym, GymGo or PySC2, etc. Further, deciding on the inputs from the environment to be fed into the algorithms.
- **Algorithm and Architecture**: Entails deciding on the type of reinforcement learning algorithm to be used, model-free or model-based, or single or multi-agent. Further, go over the related work to decide on further granularities of the possible algorithms that can be used, such as between a search-based algorithm or a model-free actor-critic algorithm, and set up a neural network architecture if required.
- **Supervised Learning**: Utilizing a dataset of human experience or moves (in case of games) to start the algorithm at a certain base level, if needed. It might be necessary to start at this base level in more complex cases with a vast combinatorial action space.

- **Reinforcement Learning**: Building on the supervised weights by games of self-play or single play in the environment. In some cases, the self-play can also use fictitious self-play or look-ahead strategies depending on the deployed algorithm. In the case of self-play, multiple different versions of the algorithms at various stages can be deployed and changed depending on performance at set intervals.
- **Exploration and diversity**: Setting the appropriate exploration weight to ensure a certain amount of exploration, rather than the continued exploitation of a suboptimal strategy. This can be done naively in some cases with the help of a random action selection, but in more complex environments, it requires a more complex exploration strategy, for example, as in AlphaStar, trying to replicate human moves to ensure a complex exploration strategy.
- **Evaluation**: Select an optimal metric for evaluating the games of play of the algorithm, which can be based solely on the cumulative reward or some other more complex metric. For example, in the case of AlphaGo, the metric used was an anchored version of the BayesElo[7] rating (official Go rating).
- **Analysis**: Further analysis of the comparative scores of the different versions of the model at various stages of play or with different variations, especially compared to the supervised baseline model.

5.2. Conclusions

The mini-AlphaStar algorithm has been implemented and trained on a *DGX server* for $\approx$ 9,000 games. The performance at the current model capacity is a maximum of 50% win rate, which has been achieved. The trained RL agent can be used to study the strategies against the inbuilt bot. Despite the limited win rate, it can be used as an effective baseline for a command and control network due to the additional complexity of the StarCraftII game.

The hierarchical MARL model has also been implemented on the SMAC network, and the necessary benchmarks have been established for future work. The hierarchical network has more scope for improvement, compared to the mini-AlphaStar algorithm, in which case, with the current algorithms, only more data and more computational power can improve the stagnant results. The algorithm can be used for further work in the command and control paradigm due to its inherent shared structure and usage of both a centralized command and decentralized subordinate agents.

Finally, the efficacy of StarCraftII as a testbed for combat simulations has also been established. This has been compared with other classical combat simulators and proven better in terms of the API interface and non-proprietary open-sourced software. Moreover, the complex-

ity of StarCraftII in a pure combat multi-agent scenario has also been established via the SMAC environment.

5.3. Future Scope of Work

- The mini-AlphaStar algorithm has peaked at 50% win rate. The model performance cannot be improved without increasing the model capacity/complexity. This cannot be done due to computational constraints.
- To improve the computational complexity, the algorithm can be further decentralized. A decentralized algorithm would reduce the computational requirements enabling the capacity to be further increased.
- To improve the reinforcement learning performance, the IMPALA algorithm must be run along with a KL divergence loss to keep the SL policy close to the RL policy. Alternatively, this has to be handled by a complex long-term exploration strategy instead of the currently existing random exploration. The SL imitation policy is the closest to the complex exploration strategy needed.
- Hierarchy, MARL Network performance, is stuck in learning cycles; this can be improved by utilizing a centralized critic with algorithms such as QMIX or VDN.

REFERENCES

Journal / Conference Papers

[3] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Playing atari with deep reinforcement learning," 2013. DOI: 10.48550/ARXIV.1312.5602.

[4] T. Li, K. Zhu, N. C. Luong, *et al.*, "Applications of multi-agent reinforcement learning in future internet: A comprehensive survey," 2021. DOI: 10.48550/ARXIV.2110.13484.

[6] M. Campbell, A. Hoane, and F.-h. Hsu, "Deep blue," *Artificial Intelligence*, vol. 134, no. 1, pp. 57–83, 2002, ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(01)00129-1.

[7] D. Silver, A. Huang, C. J. Maddison, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, ISSN: 1476-4687. DOI: 10.1038/nature16961.

[8] O. Vinyals, I. Babuschkin, W. M. Czarnecki, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, Nov. 2019, ISSN: 1476-4687. DOI: 10.1038/s41586-019-1724-z.

[9] R. A. Klausen, R. Alm*a*ng, A. Brattli, E. Lund, and G.-I. Rosvold, "Battlefield strategy from deep reinforcement learning," *North Atlantic Treaty Organization Science and Technology Organization*, 2018.

[10] V. G. Goecks, N. Waytowich, D. E. Asher, *et al.*, "On games and simulators as a platform for development of artificial intelligence for command and control," *The Journal of Defense Modeling and Simulation*, vol. 0, no. 0, p. 15 485 129 221 083 278, 0. DOI: 10.1177/15485129221083278. eprint: https://doi.org/10.1177/15485129221083278.

[11] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai," AIIDE'08, pp. 216–217, 2008.

[14] D. Silver, J. Schrittwieser, K. Simonyan, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017, ISSN: 1476-4687. DOI: 10.1038/nature24270.

[15] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," 2017. arXiv: 1712.01815 [cs.AI].

[17] R.-Z. Liu, W. Wang, Y. Shen, Z. Li, Y. Yu, and T. Lu, "An introduction of mini-alphastar," 2021. DOI: 10.48550/ARXIV.2104.06890.

[19] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai.," *Bijdragen*, Jan. 2008.

[22] O. Vinyals, T. Ewalds, S. Bartunov, *et al.*, "Starcraft ii: A new challenge for reinforcement learning," 2017. DOI: 10.48550/ARXIV.1708.04782.

[23] M. Samvelyan, T. Rashid, C. S. de Witt, *et al.*, "The StarCraft Multi-Agent Challenge," *CoRR*, vol. abs/1902.04043, 2019.

[24] J. R. Surdu, G. D. Haines, and U. W. Pooch, "Opsim: A purpose-built distributed simulation for the mission operational environment," 1999.

[26] H. Wang, H. Tang, J. Hao, X. Hao, Y. Fu, and Y. Ma, "Large scale deep reinforcement learning in war-games," pp. 1693–1699, 2020. DOI: 10.1109/BIBM49941.2020.9313387.

[27] P. Casgrain, B. Ning, and S. Jaimungal, "Deep q-learning for nash equilibria: Nash-dqn," 2019. DOI: 10.48550/ARXIV.1904.10554.

[28] J. Hu and M. P. Wellman, "Nash q-learning for general-sum stochastic games," *J. Mach. Learn. Res.*, vol. 4, no. null, pp. 1039–1069, Dec. 2003, ISSN: 1532-4435.

[29] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. Foerster, and S. Whiteson, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning," 2018. DOI: 10.48550/ARXIV.1803.11485.

[30] W. J. Yun, S. Yi, and J. Kim, "Multi-agent deep reinforcement learning using attentive graph neural architectures for real-time strategy games," pp. 2967–2972, 2021. DOI: 10.1109/SMC52423.2021.9658625.

[31] D. J. N. J. Soemers, "Tactical planning using mcts in the game of starcraft," 2014.

[32] N. Jouppi, C. Young, N. Patil, and D. Patterson, "Motivation for and evaluation of the first tensor processing unit," *IEEE Micro*, vol. 38, pp. 10–19, May 2018. DOI: 10.1109/MM.2018.032271057.

[33] L. Espeholt, H. Soyer, R. Munos, *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," 2018. DOI: 10.48550/ARXIV.1802.01561.

[34] V. Mnih, A. P. Badia, M. Mirza, *et al.*, "Asynchronous methods for deep reinforcement learning," 2016. DOI: 10.48550/ARXIV.1602.01783.

[35] M. Plappert, R. Houthooft, P. Dhariwal, *et al.*, "Parameter space noise for exploration," 2017. DOI: 10.48550/ARXIV.1706.01905.

[36] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., pp. 8024–8035, 2019.

[37] J. K. Terry, B. Black, N. Grammel, *et al.*, "Pettingzoo: Gym for multi-agent reinforcement learning," 2020. DOI: 10.48550/ARXIV.2009.14471.

[38] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[39] E. Liang, R. Liaw, P. Moritz, *et al.*, "Rllib: Abstractions for distributed reinforcement learning," 2017. DOI: 10.48550/ARXIV.1712.09381.

[40] Martín Abadi, Ashish Agarwal, Paul Barham, *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, Software available from tensorflow.org.

[41] R.-Z. Liu, "Rethinking of alphastar," 2021. DOI: 10.48550/ARXIV.2108.03452.

[42] P. Sunehag, G. Lever, A. Gruslys, *et al.*, "Value-decomposition networks for cooperative multi-agent learning based on team reward," AAMAS '18, pp. 2085–2087, 2018.

Reference / Hand Books

[5] P. Cramton, *Strategic Games*.

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second. The MIT Press, 2018.

[20] J. Mańdziuk, *MCTS/UCT in solving real-life problems*. Jan. 2018, pp. 277–292, ISBN: 978-3-319-67945-7. DOI: 10.1007/978-3-319-67946-4_11.

[25] U. S. o. A. Department of Defense, *JOINT MILITARY SYMBOLOGY, DEPARTMENT OF DEFENSE INTERFACE STANDARD*.

Web

[1] D. Silver, *Introduction to reinforcement learning*, Available at https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver. Accessed on 01/2022.

[2] D. Research and D. Organization, *Cair, drdo*, Available at https://www.drdo.gov.in/labs-establishment/about-us/centre-artificial-intelligence-robotics-cair. Accessed on 02/2022.

[12] A. Ng, *Introduction to machine learning*, Available at https://www.coursera.org/learn/machine-learning. Accessed on 01/2022.

[13] J. Tsitsiklis, *Introduction to probability*, Available at MIT OCW Probabilistic systems analysis and applied probability Fall 2010 Accessed on 01/2022.

[16] R. Liu, W. Wang, Y. Yu, and T. Lu, *Mini-alphastar*, `https://github.com/liuruoze/mini-AlphaStar`, 2021.

[21] B. Entertainment, *Starcraftii: Wings of liberty*, Available at `https://starcraft2.com/en-us/game`. Accessed on 04/2022.