SMAC-lite: A lightweight SMAC-based environment for Multi-agent Reinforcement Learning Research

Tawqir Sohail

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2022

Abstract

Multi-agent reinforcement learning (MARL) is an active area of research in machine learning with many important practical applications. The StarCraft Multi-Agent Challenge (SMAC), a set of fully-cooperative partially observable MARL test environments using StarCraft II, is an important standardized benchmark in this field. Despite its value, SMAC remains inaccessible to many researchers in this field as it requires expensive computer hardware to use in practice.

This project introduces SMAC-lite, a MARL test environment that can be used as an alternative to SMAC, based on SMAC's `5m_vs_6m` test environment. We show that SMAC-lite is significantly faster than SMAC whilst showing comparable performance on the QMIX and IQL MARL algorithms. With this project, we aim to introduce and lay the groundwork for an alternative to SMAC that is more accessible and inclusive to researchers working in this field.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Tawqir Sohail*)

Acknowledgements

First and foremost, I would like to thank my wife Martha. This work would not have been realized without her unending love and support. I would also like to thank my co-supervisor, Arrasy, whose sage guidance and advice has helped shaped this project into its final form.

Table of Contents

Chapter 1

Introduction

Reinforcement learning is a branch of research in machine learning that deals with learning how to take the best actions over time to maximize an overall reward. Reinforcement learning emerged in the early 1980s [1] as a confluence of several streams of work in areas ranging from animal learning psychology [2] to solving problems in optimal control theory [3]. This field has grown rapidly since then and is now considered to be one of the primary paradigms of machine learning [4].

In reinforcement learning, one or more agents choose actions based on observations from their given environment to minimize punishment and maximize reward. The best actions given the state of the environment must be inferred through repeated trial and error experiences, much in the same way a human might improve their chess skills by playing many practice games. Like in chess, present actions may affect rewards and punishments far into the future. These unique challenges distinguish reinforcement learning from other types of machine learning [1]. Figure 1.1 shows a high-level view of a reinforcement learning system. The area of reinforcement learning that our work addresses is multi-agent reinforcement learning (MARL). These are reinforcement learning tasks where an environment is shared by multiple agents.

Due to the highly general modelling framework supplied by reinforcement learning, it is a powerful toolkit to tackle some of the most challenging problems in artificial intelligence (AI). Reinforcement learning has been applied to realize the advent of self-driving cars [6], to understand how to optimally allocate common pool[1] resources [7] and to beat skilled human players at complex video games such as StarCraft II [8]. Indeed, due to the raw complexity of the latter [8] combined with its highly competitive

[1]Common pool resources are natural or manufactured resources such as water, fishing quotas, energy supplies, etc.

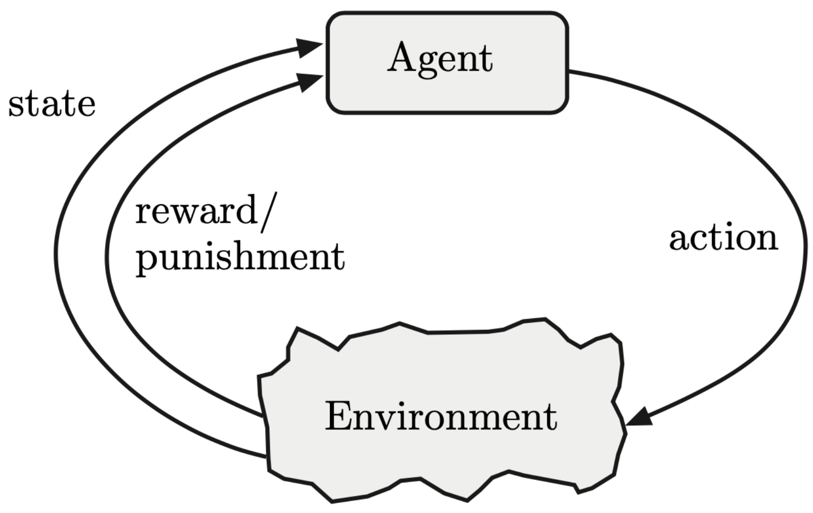


Figure 1.1: A block diagram showing the high-level architecture of a reinforcement learning system [5].

professional e-sports leagues, this research has been subject to particular media attention, with the Guardian describing it as a “landmark achievement”[2] in AI.

In general, StarCraft II is understood to be an excellent test environment for reinforcement learning [9] and much work has been done in this domain. One such work is the StarCraft Multi-Agent Challenge (SMAC) [10], a framework that leverages the rich game dynamics of StarCraft II for MARL research. Specifically, SMAC focuses on MARL problems where agents must work together, each with a limited view of their environment, to maximize a common reward. It consists of a set of mini-challenges derived from, but distinct to the full game of StarCraft II. Each mini-challenge consists of two teams of units in various configurations competing to win a game. The ‘friendly’ team consists of agents controlled by a MARL algorithm with the ‘enemy’ team comprising of units controlled by StarCraft II’s hand-coded AI. Each mini-challenge aims to be uniquely idiosyncratic and challenging. SMAC is different from much previous work in reinforcement learning using StarCraft II, which has focused on single-agent reinforcement learning focusing on playing the full game from the point of view of a single human player [8, 11, 12]. Although there has been sporadic work to apply StarCraft II to MARL research [13, 14, 15, 16, 17, 18, 19, 20], SMAC is the first concerted effort to create a standardized test suite to systematically drive this research domain forward. Despite its value to the field in its existing form, SMAC has some

[2] `https://bit.ly/3T20aXu`

weaknesses. It is cumbersome to install as it requires the full game of StarCraft II due to its use of the StarCraft II Learning Environment (SC2LE[3]) [21] to create and control the game environment. Moreover, due to this dependency, it is slow to run, particularly as training MARL algorithms typically require millions of game-play experiences. This limits the use of SMAC to well-funded research groups who can afford the expensive computing resources required to use SMAC in practice. As SMAC is quickly becoming a common benchmark for MARL research [22, 23, 24, 25, 26], this excludes many researchers and research groups who do not have access to these resources.

This project introduces SMAC-lite, a lightweight alternative to the SMAC framework. In particular, we create a MARL test environment loosely based on the SMAC `5m_vs_6m` mini-challenge. We show that SMAC-lite is a functionally valid replacement for the `5m_vs_6m` mini-challenge with significantly faster execution times. We achieve this by creating a new MARL environment from scratch using the Griddly [27] framework driven by the EPyMARL [28]. Our implementation also removes the any dependency on StarCraft II, allowing us to greatly reduce the installation footprint of SMAC-lite compared to SMAC. Although we only provide a single environment compared to SMAC's plethora of mini-challenges, we show it is possible to reap the benefits of the `5m_vs_6m` mini-challenge using less powerful computing resources whilst also reducing its installation footprint. SMAC-lite aims to lay the groundwork required to make standardized MARL benchmarks accessible to more researchers working in this field. With this project, we aim to introduce a more computationally accessible MARL test environment which can easily be extended to include more mini-challenges from SMAC and beyond.

[3]`github.com/deepmind/pysc2`

Chapter 2

Background and Motivation

2.1 StarCraft II and SMAC

StarCraft II[1] is a military science-fiction video game published by Blizzard Entertainment. The game comprises individuals or teams of individuals competing against each other or a built-in handcrafted AI to manage resources and win battles. Players must play as one of three 'races' - Protoss[2], Terran[3] or Zerg[4]. This is one of the most interesting aspects of the game, as the races are very well-balanced but require totally different strategies to play effectively [9]. All units in the game belong to one of these races. SMAC's `5m_vs_6m` mini-challenge, which SMAC-lite is based on, is composed of two teams of Marines[5], the basic infantry units of the Terran race. Another interesting aspect of StarCraft II is the strategic considerations required to play different aspects of the game. These can be be divided into two distinct areas.

- **Macromanagement (macro)**: This encompasses high-level, economic and long-term strategic considerations such as creating and managing infrastructure, building armies and expanding territory.
- **Micromanagement (micro)**: This covers low-level control of units or teams of units to determine where they should move, if they should attack or retreat, etc. Players with good micro strategy tend to have units with longer lifespans.

These are both interesting aspects of the game that must be considered in order

[1] `starcraft2.com`
[2] `starcraft.fandom.com/wiki/Protoss`
[3] `starcraft.fandom.com/wiki/Terran`
[4] `starcraft.fandom.com/wiki/Zerg`
[5] `starcraft.fandom.com/wiki/Marine_(StarCraft_II)`

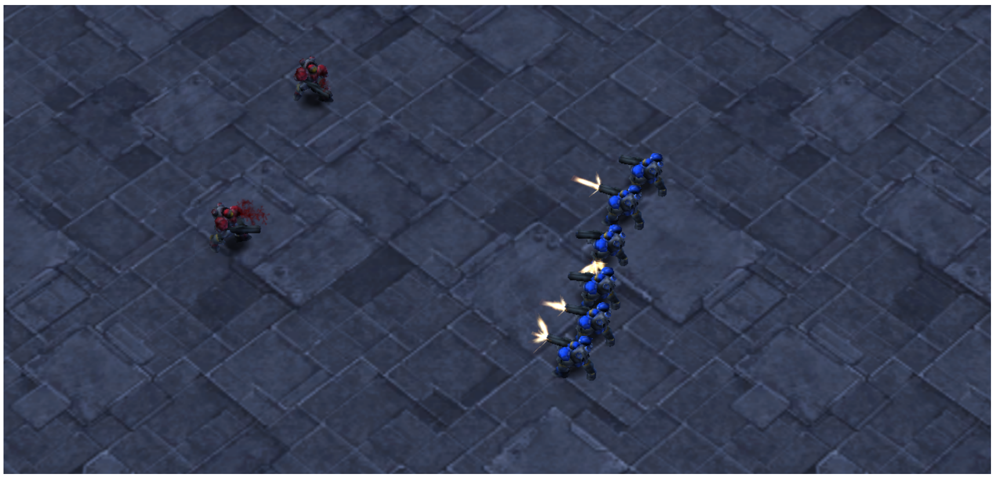


Figure 2.1: A screenshot of SMAC's rendering of the `5m_vs_6m` mini-challenge during training. The graphics used are the same as in the full game of StarCraft II. This depicts a team of enemy Marines firing on the agent team of Marines.

to effectively play the full-game of StarCraft II. However, the focus of SMAC and SMAC-lite is *only* on micro, as this is more amenable to being modelled as a MARL problem. Moreover, this allows the two to be compared.

StarCraft II has been shown to be an excellent test environment for reinforcement learning [9]. This is partly due to its enormous number of states [9], which far exceed those of traditional reinforcement learning test environments such as chess [29] and backgammon [30]. Consequently, StarCraft II has been well studied in the context of this field. Much of this research has focused on playing the full game of StarCraft II from the perspective of a single human player [8, 11, 12], framing the game as a single-agent reinforcement learning problem with an emphasis on macro strategy. In recent years, the crowning achievement of this stream of research has been AlphaStar [8], which has beaten 99.8% of human players to attain Grandmaster rank in the game. This is a significant achievement, as StarCraft II is hugely popular with a high skill ceiling, with its professional e-sports leagues being a lucrative source of income for many top human players. As impressive as this is, single-agent reinforcement learning is not suitable for modelling many real-world challenges, as many are inherently multi-agent in nature. Additionally, most real-world problem settings have noisy input channels where all agents may only have access to a small amount of information from the environment but must still learn to coordinate effectively [10].

The StarCraft Multi-Agent Challenge (SMAC) [10] is a framework designed to leverage the StarCraft II game environment for MARL research. It is written using the SC2LE and PySC2 frameworks[6] [21], which are designed to allow easy interfacing with the StarCraft II game engine. SMAC is runnable via PyMARL[7] [10], an open-source framework that has been released alongside SMAC to drive MARL experiments using SMAC as a training environment. Although there has been sporadic work to use StarCraft II as an environment for MARL [13, 14, 15, 16, 17, 18, 19, 20], SMAC is the first concerted effort to create a standardized test-bed to drive this field forward [10]. SMAC consists of a set of mini-challenges focusing purely on micro strategy and therefore distinct from the full StarCraft II game. Each mini-challenge consists of two teams of units in various configurations attempting to win by eliminating all units belonging to the opposing the team. The 'friendly' team is controlled using MARL algorithms while the enemy team is controlled by StarCraft II's handcrafted game AI. Each mini-challenge has its own challenges and idiosyncrasies and provides a qualitatively challenging environment for MARL research [10].

Due to its qualities, SMAC has been growing in popularity in MARL literature as a metric for benchmarking MARL algorithms [22, 23, 24, 25, 26]. Despite its efficacy as a test environment, SMAC has some issues that make it challenging to use in practice. Due to its dependence on the StarCraft II game engine, SMAC requires installation of the full game of StarCraft II. Although the process of installing the game is easy, it is undesirable as it requires around 30 gigabytes of disk space [8] despite SMAC using only a small portion of the game logic.

More importantly, it is computationally demanding, as StarCraft II is a complex game with many underlying processes and graphically intensive game rendering. The intrinsic coupling between SMAC and the StarCraft II game logic means that it is slow compared to many other MARL environments, such as those provided by `pettingzoo`[31][9]. Furthermore, reinforcement learning models typically need to be trained over millions of game-play experiences, which in the case of SMAC, is impractical without expensive high-performance computer hardware. This limits MARL research with SMAC to well-funded research groups who can afford the required hardware. As SMAC is now considered a standard benchmark in this domain [32], its demanding computational requirements exclude many researchers and research groups from utilizing it. SMAC-lite aims to provide a fast lightweight alternative to SMAC

[6]`github.com/deepmind/pysc2`
[7]`github.com/oxwhirl/pymarl`
[8]`us.battle.net/support/en/article/27575`
[9]`pettingzoo.ml/envs`

by removing all dependencies on StarCraft II. It provides a single MARL environment loosely based on the `5m_vs_6m` mini-challenge from SMAC, built with Griddly and usable via EPyMARL.

2.2 Griddly and EPyMARL

Griddly [27][10] is a open-source library designed for building reinforcement learning environments. It provides a YAML [11] based configuration specification called GDY (Game Driven YAML), which can be used to define a complete game environment, including objects, their behaviours and how they interact with other objects in the environment. The ability to define and alter game environments by only making configuration changes allows for rapid prototyping and iteration of game environments. In addition to this, Griddly's underlying game engine is written entirely in C++, using the Vulkan SDK[12] to render observational states in the game. The result of this is a game engine that is fast, lightweight and able to render some games at more than 30,000 frames per second [27]. In contrast, the frame rate for StarCraft II is capped at 60 frames per second[13].

Griddly is well-suited to our use-case for several reasons. It is easy to learn, easy to install and very fast. Moreover, it has been designed specifically for reinforcement learning research and offers features such as partial observability and reward allocation out-of-the-box, reducing the software development overhead for research.

Our Griddly environment is driven using EPyMARL[14] [28], a MARL framework derived from PyMARL[15] [10], but with some additional features. It provides several out-of-the-box implementations of modern MARL algorithms, as well as a robust framework to run, manage and record experiment results. We chose EPyMARL over PyMARL as it implements additional MARL environments (Level-Based Foraging [33, 34][16] and Multi-Robot Warehouse[17]), allowing us to adapt their existing implementations for our use-case.

[10]`griddly.readthedocs.io/en/latest/index.html`
[11]`yaml.org`
[12]`vulkan.org`
[13]`us.battle.net/support/en/article/141374`
[14]`github.com/uoe-agents/epymarl`
[15]`github.com/oxwhirl/pymarl`
[16]`https://github.com/uoe-agents/lb-foraging`
[17]`https://github.com/uoe-agents/robotic-warehouse`

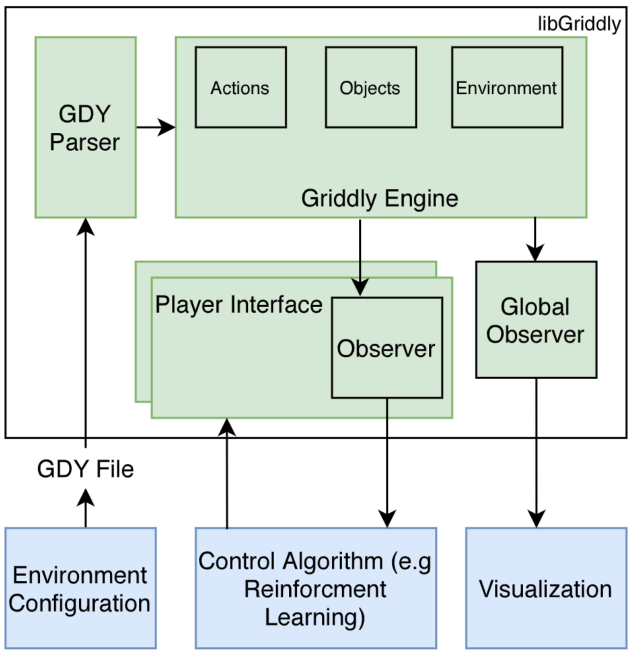


Figure 2.2: A high-level architecture diagram showing the various components of the Griddly framework. [27]

2.3 Multi-agent reinforcement learning (MARL)

As discussed previously, multi-agent reinforcement learning (MARL) is an area of research in reinforcement learning that studies problems where multiple independent agents share a common environment. In a MARL problem setting, the aim of each agent is to maximize its own long-term reward by interacting with the environment and the other agents [35]. MARL problems can be fully cooperative, where all the agents must learn to work together to maximize a common reward. An example of such a problem might be a team of robot warehouse workers who must coordinate and work together to minimize stock delivery times. In contrast, fully competitive MARL problems are such that the agents are competing against each other to maximize their own goals, which are often conflicting. As a result, many fully competitive MARL problems are zero-sum, which means the total rewards of the agents sum to zero [35]. An example of a fully-competitive problem might be many agents trading on a stock exchange, with each striving to maximize their own gain at the expense of the other agents. Many real-world MARL problems do not necessarily conform to these categorizations and are a mix of the two.

There are numerous benefits to framing reinforcement learning problems as MARL problems. Due to their decentralized nature, MARL systems are much more robust compared to their single-agent counterparts [36]. If an agent fails in a MARL system, the remaining agents can pick up the slack [36]. Moreover, agents can easily be added and removed from a system, allowing for highly flexible scalable systems [36]. MARL systems are also easier to run in parallel on computer hardware, which can lead to significant performance advantages over single-agent systems [36]. Agents in MARL task settings can also help each other learn via experience sharing [36]. More ‘skilled’ agents can serve as teachers for other agents [37] or agents can learn by imitating more skilled agents [38].

Naturally, there are also certain challenges that must be considered when working with MARL. One that is unique to MARL is non-stationarity. In multi-agent settings, the environment is constantly modified by the actions of agents, so from the perspective of a single agent, the environment is not stationary. Each agent is effectively faced with a moving-target problem where its best policy[18] is not static and is constantly influenced by all the other agents [36]. This violates the Markov property, which says that the state of the environment must encapsulate all aspects of the past agent-environment interactions and is intrinsic to the convergence properties of many single-agent reinforcement learning approaches [1]. Our chosen benchmarking algorithms, IQL [39] and QMIX [40], have different approaches to deal with non-stationarity. Another challenge which pervades all of reinforcement learning is partial observability. Most real-world data sources and data input streams are faulty, imperfect or fail to capture important information about an environment [10]. For example, an agent in a football team has a limited view of the football pitch at any given time but must still learn to coordinate with the whole team in order to win a game. Although there are approaches available to mitigate these problems [41], dealing with partial observability is an active and important area of research.

The focus of SMAC-lite, like SMAC, is on *partially observable fully cooperative* MARL problems. In research literature, these are known as Dec-POMDPs [41]. This is an important sub-field of MARL research with important real-world applications. Like SMAC, SMAC-lite aims to lay the groundwork for standardized benchmarks to drive Dec-POMDP research forward, albeit with smaller computational overheads. For more information about Dec-POMDPs, we refer the reader to [41].

In order to compare SMAC-lite and SMAC, we have chosen to use the IQL [39]

[18]A policy is a function that defines a probability distribution over all the possible actions given a state [1].

and QMIX [40] algorithms. These have been selected as they use different learning paradigms and therefore allow us to gain some understanding on how this affects performance between these environments. In addition to this, we use both algorithms to benchmark *both* SMAC and SMAC-lite, allowing us to understand the suitability of SMAC-lite as functional alternative to SMAC.

- **IQL [39] :** IQL uses independent learning, where each agent learns a decentralized state-action value function that is conditioned only on the actions and observations of each individual agent. Consequently, each agent treats all other agents as part of the environment.
- **QMIX [40] :** Compared to IQL, QMIX is more sophisticated. It uses centralised training with decentralised execution, where agents are allowed a view of the whole environment during training, but must condition their policies only on local observations. It uses the approach of value decomposition [42] to break down the joint state-action value function of all agents into individual state-action values per agent.

2.4 Related Work

As StarCraft II has been shown to be an excellent test environment for reinforcement learning [9], there have been several efforts to apply it to this purpose. Much of this work focuses on macro strategy, using single-agent reinforcement learning to play the full game from the perspective of a human player [8, 11, 12]. Although there has also been sporadic work to use the game in MARL environments [13, 14, 15, 16, 17, 18, 19, 20], there has been a clear absence of any standardized benchmarking framework that can decisively be used to compare MARL algorithms. SMAC is the first framework with the explicit aim of providing such a tool. In contrast to previous work applying single-agent reinforcement learning to StarCraft II, SMAC uses the rich environment dynamics of the game to create a new set of mini-challenges. Despite SMAC becoming well established as a standard benchmark in MARL literature [32, 22, 23, 24, 25, 26, 28], to our knowledge, there has not been any concerted effort into optimizing it to be less computationally demanding, and therefore, accessible to more researchers.

However, outside of the realm of StarCraft II, there have been other efforts to standardize testing in MARL. The Japanese video game Bomberman[19] has been used

[19] www.konami.com/games/bomberman

as the basis for a MARL test environment [43], also with multiple mini-challenges facilitating both cooperative and competitive MARL. However, Bomberman lacks the rich environment dynamics of StarCraft II [10] and is therefore a less challenging test-bed. Simulated robot football is a game which has also been used for fully-cooperative MARL research in the past. This type of environment uses simple physics to simulate a game of football, where a team of agents aim to maximize their rewards by working together to keep possession of a football within a designated area while simultaneously stopping an enemy team stealing from it. This started with Keepaway soccer [44], built on top of RoboCup [45], which later evolved into the more challenging Half Field Offensive task [46, 47] requiring agents to score goals in addition to the standard Keepaway soccer behaviour. Despite the value of robot football as a MARL test environment, it lacks the diversity of the mini-challenges presented by SMAC, which is better suited to test the performance of MARL algorithms in many differing scenarios. More recently, there have been efforts to introduce more MARL test scenarios driven by the EPyMARL framework [28]. These include the Level-Based Foraging game [48] and RWARE [28]. However, these environments also lack the diversity of SMAC.

In general, most research around MARL test environments are one-off toy problems with limited usage, making it hard to meaningfully measure performance between different works of research [10]. SMAC is the first standardized benchmark in this field to provide a diverse set of challenges for Dec-POMDPs, and with SMAC-lite, we aim to lay the foundations of a more efficient version of SMAC.

Chapter 3

SMAC-lite

3.1 Initial Approach

Our initial efforts to create a more lightweight version of SMAC ultimately proved unfruitful. In order to understand exactly what the computationally demanding aspects of SMAC are, we use a popular profiling library called `py-spy`[1] to dissect its performance. Profilers allow software engineers and researchers to understand which components of a program take the largest proportion of time to execute by repeatedly sampling its process. `py-spy` is well-suited for our purposes as it has a very low-overhead and runs in a process independent of the process it is profiling, therefore not influencing the results.

Our profiling results can be seen in Figure 3.1. This shows a flame-graph[2] which can be used to visualize how much time SMAC spends running different functions as part of its execution. This is the flame-graph for training the QMIX [40] algorithm on the `5m_vs_6m` scenario. The vertical axis represents the call-stack of the program and the horizontal axis represents the program executing through time. Accordingly, the horizontal bars at the top of the graph represent high-level function calls and bars below them represent their subroutines. From Figure 3.1, we can see that a significant proportion of time is spent inside the function at line 69 in `episode_runner.py`. In the PyMARL code-base used to run SMAC, this corresponds to the function that applies the algorithm's predicted actions to the game environment, computes the resulting observations and renders these observations in the form shown in Figure 2.1. From Figure 3.1, we can see that this function call takes up almost half of the program's time

[1] `github.com/benfred/py-spy`

[2] `brendangregg.com/FlameGraphs/cpuflamegraphs.html`

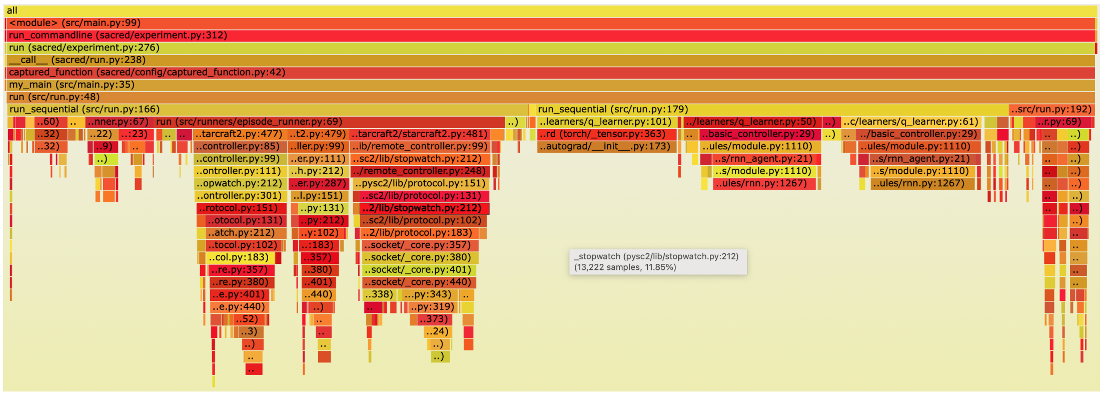


Figure 3.1: The output of `py-spy` when used to profile a training run on the `5m_vs_6m` SMAC mini-challenge. The vertical axis shows the call stack and the horizontal axis shows the program executing through time. Each horizontally oriented block represents time spent in a function call.

during training. This is surprising as one would expect the game environment itself to take a small proportion of the overall time compared to the QMIX training algorithm, which uses computationally intensive deep learning subroutines as part of its operation.

Based on this information, we hypothesized that the slow execution of the environment is due to the rendering of the observational states of the game. One of the most computationally expensive aspects of video game execution in general is graphical rendering of in-game lighting [49], which led us to this hypothesis, as StarCraft II is designed to have rich lighting effects using a sophisticated shader framework [50]. While realistic lighting is conducive to an enjoyable game-play experience, it is highly unnecessary in the context of reinforcement learning, where efficiency of the environment takes precedence over the rendering of observational states. Given that the game logic and rendering in SMAC occur within the SC2LE game engine [21], we started investigating the feasibility of refactoring this code-base in order to separate these two components. Our proposed transformation would theoretically allow us to construct a bespoke efficient and lightweight module to render the environment's observational states. This design change is conveyed by Figure 3.2. It would have allowed us to replace the StarCraft II graphics engine, which is superfluous for MARL research, with a much simpler and faster rendering engine which would utilize the raw outputs from the newly refactored game logic component. This approach would have had many benefits. The modularity of the refactored SC2LE engine would allow easy experimentation with different rendering engines without any changes required to the decoupled StarCraft II

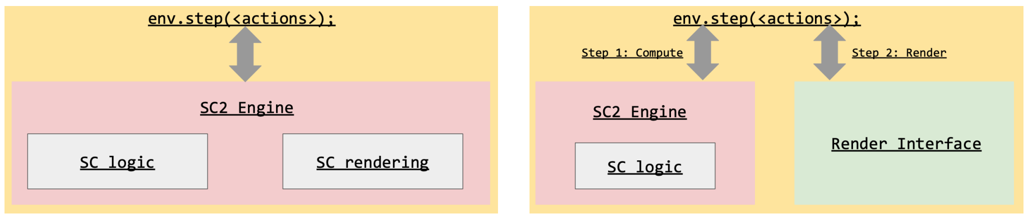


Figure 3.2: The figure on the left is a high-level architecture diagram showing the design of SMAC. The figure on the right shows our initial proposal for SMAC-lite.

game logic. Moreover, specifying and documenting an interface between the StarCraft II game logic and the rendering engine would allow independent iteration on the game logic component and the game rendering component. This design change would have also allowed us to easily port *all* the SMAC mini-challenge environments to use a more efficient rendering engine, instead of just focusing on a particular mini-challenge.

However, upon further investigation, this approach ultimately proved to be infeasible. SC2LE provides an interface to drive StarCraft II with code, but does not control the game logic nor the rendering. It uses a streaming websocket[3] connection to act as an intermediary layer of communication between SMAC and the full StarCraft II game, the latter of which encapsulates all the game logic *and* rendering. This would have moved our refactoring approach downstream to the StarCraft II code-base. Notwithstanding the difficulty of such an endeavour in the time afforded for this project, modification of StarCraft II is strictly prohibited under the End User License Agreement (EULA) [4] from Blizzard.

In addition to this, we discovered that the Linux client for StarCraft II, which has been developed by Blizzard especially for machine learning research [21], has no rendering functionality. As Linux is the dominant platform used to conduct machine learning research, most users of SMAC would likely not be specifically affected by the computational overhead of StarCraft II rendering. These factors prompted us to seek out other approaches to optimize SMAC.

3.2 Griddly Approach

Our final SMAC environment has been written using Griddly [27] and is runnable via EPyMARL [28]. Griddly exposes its environments using wrappers from OpenAI's Gym

[3]developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

[4]www.blizzard.com/en-us/legal/fba4d00f-c7e4-4883-b8b9-1b4500a402ea/blizzard-end-user-license-agreement

framework [5], a standard software development paradigm in reinforcement learning research, allowing us to successfully integrate our Griddly SMAC-lite environment with EPyMARL. This integration is key to the fairness of our experiments, as both the SMAC and SMAC-lite experiments use shared libraries to execute all our experiments.

3.2.1 Game Environment

SMAC-lite consists of a single environment based on the `5m_vs_6m` mini-challenge from SMAC. This mini-challenge consists of two teams of Marines competing to win a game by eliminating all units in the opposing team. Each game can be considered to be an episode[6], which ends when one of the teams is eliminated or the maximum number of time-steps is reached and the game times out. The agent team is considered to have won the game only if the enemy team is eliminated before the game times out.

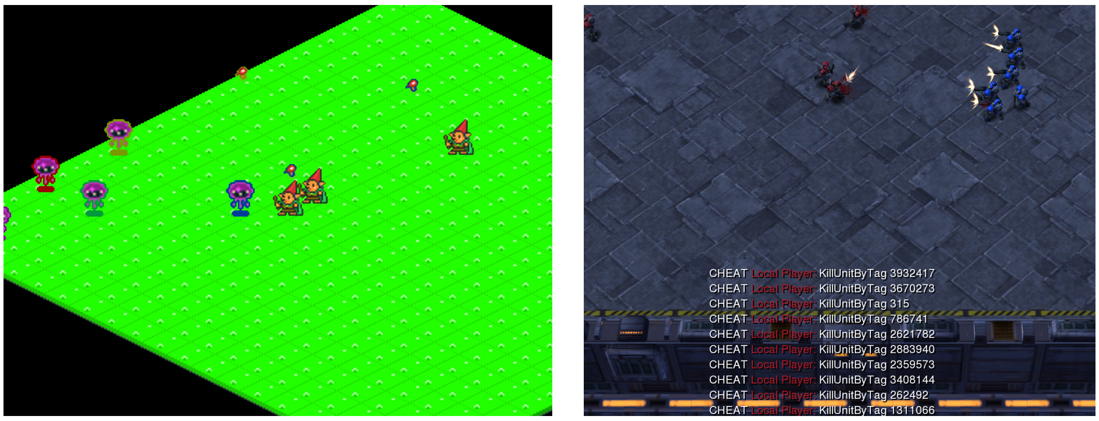
`5m_vs_6m` has been chosen as the environment to base SMAC-lite on. The SMAC authors classify the SMAC mini-challenges with three difficulty ratings - `Easy`, `Hard` and `Super-Hard`. They report that QMIX, the general best performer from their benchmarking results, achieves an impressive 95% test win rate on all `Easy` scenarios [10]. They also report that it makes meagre gains on `Super-Hard` scenarios [10]. Therefore, the `Hard` category stands out as a natural choice from which to select a mini-challenge to model SMAC-lite after. In particular, we initially selected `5m_vs_6m` as all units in this scenario are identical and have identical action spaces. Although the final version of SMAC-lite does not conform to this, this significantly minimized complexity in the environment and aided with debugging in the early stages of the project. Figure 3.3 shows a side-by-side view of SMAC-lite and `5m_vs_6m` from SMAC.

The final form of SMAC-lite differs from `5m_vs_6m` in terms of the behaviour of the enemy units. In `5m_vs_6m`, both teams consist of Marine units who shoot each other when they are within range. While friendly units can shoot in SMAC-lite, enemy units must instead chase down friendly units and eliminate them by coming into direct contact with them. Although this behaviour differs from `5m_vs_6m` and somewhat detracts from the asymmetric aspect of the original mini-challenge, our aim is to compensate for this by inducing interesting behaviours such as ‘kiting’[7], which is a mainstay in human micro-strategy.

[5]`www.gymlibrary.ml/content/wrappers/`

[6]In reinforcement learning, an episode is the full sequence of states, actions and rewards between the start state (beginning of the game) and the terminal state (win, loss or time-out).

[7]`liquipedia.net/starcraft2/Kiting`



```
CHEAT Local Player: KillUnitByTag 3932417
CHEAT Local Player: KillUnitByTag 3670273
CHEAT Local Player: KillUnitByTag 315
CHEAT Local Player: KillUnitByTag 786741
CHEAT Local Player: KillUnitByTag 2621782
CHEAT Local Player: KillUnitByTag 2883940
CHEAT Local Player: KillUnitByTag 2359573
CHEAT Local Player: KillUnitByTag 3408144
CHEAT Local Player: KillUnitByTag 262492
CHEAT Local Player: KillUnitByTag 1311066
```

Figure 3.3: The screenshot on the left is a screenshot of the SMAC-lite environment under isometric rendering. The one on the right is the original `5m_vs_6m` environment from SMAC.

In `5m_vs_6m`, enemy units are controlled by StarCraft II's handcrafted AI, with the majority of their behaviours consisting of chasing and firing on the agent units when they come within range. They also have a fixed amount of health that must be depleted before they are eliminated from the game. Enemy units in SMAC-lite also exhibit this behaviour as they also have a fixed amount of health. However, there are larger differences in the general behaviour of enemy units between `5m_vs_6m` and SMAC-lite. In `5m_vs_6m`, the behaviour of units is controlled by the handcrafted AI built into the StarCraft II game logic. As we are strictly prohibited from reverse engineering or creating a derivative work from StarCraft II due to Blizzard's EULA, which includes the behaviour of the enemy units, we instead use a simple heuristic. This heuristic is the A* search algorithm, a path-finding algorithm to find the shortest path between a source and target object. The source object in SMAC-lite is each enemy unit which is programmed to move towards the target object. The target object is nearest MARL controlled friendly agent unit. Although this is a simple heuristic compared to StarCraft II's handcrafted AI, it is highly suitable for SMAC-lite's use-case as it is very fast [51]. A* is the most popular path-finding algorithm used in game environments, with much work having gone into optimizing its performance [51]. Moreover, it has been shown to be optimal for path-finding problems with a single target object [51]. For more information on the A* algorithm and its application in game environments, we direct the reader to [51].

3.2.2 Rendering

The rendering of observational states in a reinforcement learning environment is an invaluable amenity, allowing visualization of the environment for purposes such as debugging and observing interesting behaviours. Griddly provides several out-of-the-box rendering modes for its environments, which can be used to specify the visual richness of the rendered states.

- **Isometric:** This is the graphically richest rendering supported by Griddly. The game arena is displayed from a wide-angle camera view, giving the 2-dimensional space the impression of a 3-dimensional space. Figure 3.4 shows SMAC-lite rendered in isometric mode. This mode requires sprites[8] to represent the objects in the environment. The sprites used for SMAC-lite are derived from the Griddly repository[9].

- **Sprite 2D:** This rendering mode is similar to isometric mode but with the camera view from directly above the game arena, showing an unambiguous 2-dimensional space. Sprites are also required for this type of rendering. Sprite 2D rendering has not been implemented for SMAC-lite, as it is not sufficiently different from Block 2D rendering.

- **Block 2D:** This type of rendering is similar to Sprite 2D rendering, but the sprites specified must be simple 2-dimensional shapes. Figure 3.5 shows an example of this for SMAC-lite.

- **Vector:** This is the simplest and fastest rendering mode supported by Griddly. Each object is represented as unit squares with different colours and does not require any sprites or shapes to be specified. Figure 3.5 also shows an example of this. The friendly units are orange and enemy units are green.

Although Griddly rendering is highly efficient compared to StarCraft II, we disable rendering for our experiments in order to ensure fair training time comparisons between SMAC and SMAC-lite, as SMAC also disables rendering during training on the Linux platform.

[8]In computer graphics, a sprite is an image used to represent objects. Friendly agent units are represented with floating jelly objects and enemy units are represented with gnomes.

[9]`github.com/Bam4d/Griddly`

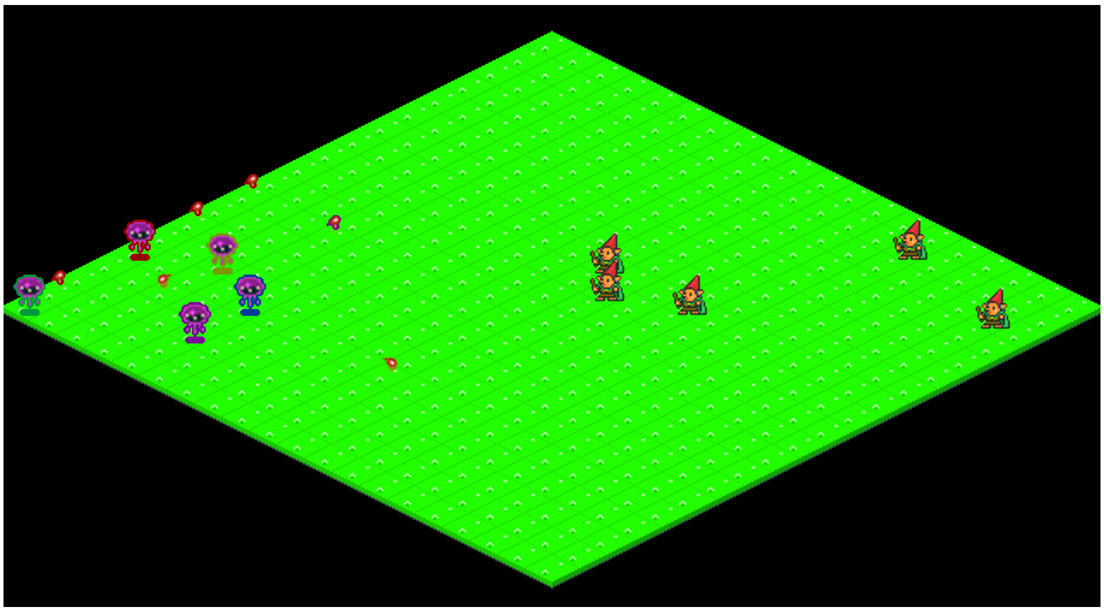


Figure 3.4: SMAC-lite rendering in isometric mode in Griddly.

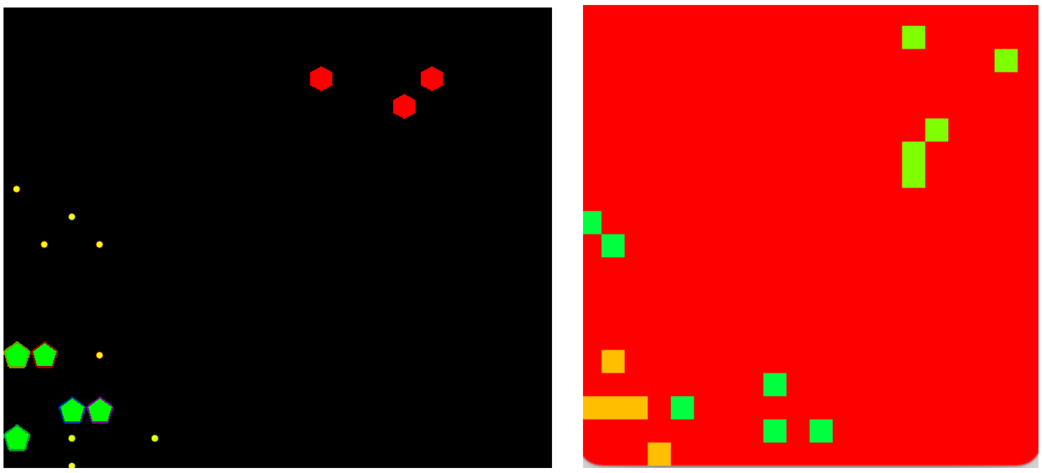


Figure 3.5: SMAC-lite rendering in Block 2D and vector modes on the left and right respectively.

3.2.3 Action Space

The action space for agents in SMAC-lite is discrete, comprising of either moving or firing in one of the four cardinal directions. Eliminated agents cannot take any actions. This is equivalent to the action space in `5m_vs_6m` in SMAC. Furthermore, like `5m_vs_6m` and SMAC in general, projectiles fired by SMAC-lite agents have a range that is lower than each agent's partially observable view. This helps to induce human-like game-play by encouraging agents to move before firing on the enemy units [10]. In SMAC-lite, there is added complexity in this regard as agents must learn to move *away* from enemy units whilst staying within firing range. Projectiles fired by agent units in SMAC-lite have a range of 6 units while the agents partially observable space is 10 by 10 units.

3.2.4 Observation Space

In addition to a global observation space, SMAC-lite also supports a partial observation space for each agent. This is essential to impactful MARL research, as most practical MARL problems are partially observable [10]. As stated previously, SMAC-lite uses a 10 by 10 observation space for each agent with each agent at the centre of its space. This value has been chosen such that the observation space for each agent is adequately large compared to their firing range. This is illustrated in Figure 3.6. This facilitates the use of the centralized training with decentralized execution training regime.

SMAC-lite uses the default observation space provided by Griddly[10]. This consists of a separate one-hot encoded vector of the game arena for agent units, their projectiles and enemy units (three vectors in total). On the other hand, SMAC uses a handcrafted observation space, with some elements of the feature vector consisting of StarCraft II specific attributes such as `shield` and `unit_type`. Although Marines have shields in StarCraft II, they have not been implemented in SMAC-lite agent units, as enemy units are programmed to eliminate friendly units on contact. Unit type is not applicable to SMAC-lite, as all there are only two types of units and this information is represented in Griddly's default observation space format. Given the irrelevance of some components of SMAC's handcrafted observation space to SMAC-lite, we choose to use Griddly's default observation space as it represents a complete view of SMAC-lite's observation space.

[10]`griddly.readthedocs.io/en/latest/getting-started/observation%20spaces/index.html`

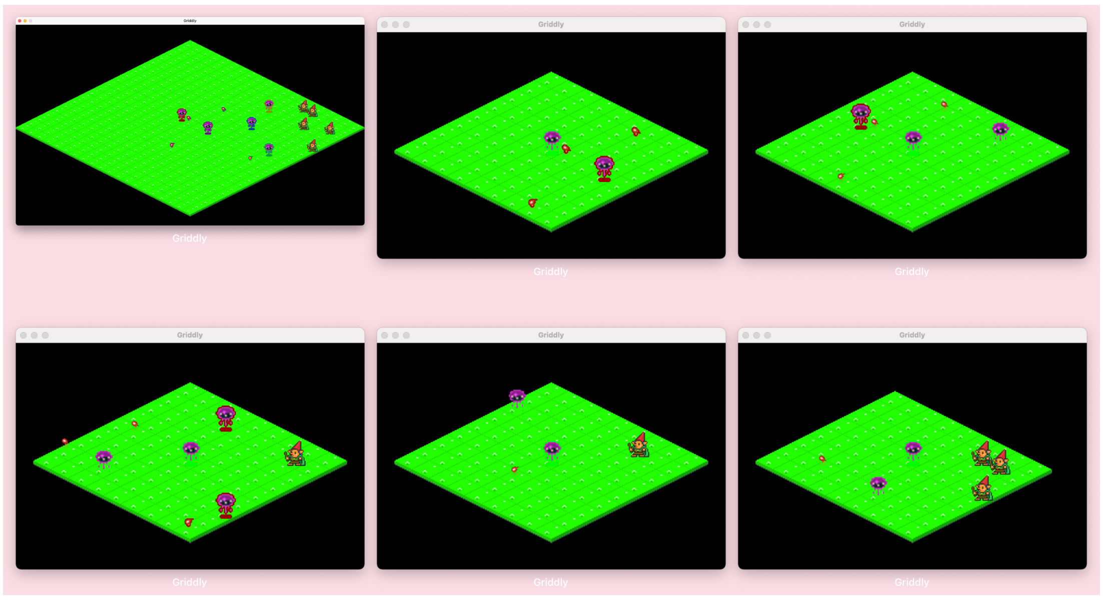


Figure 3.6: A series of windows illustrating partial observability in SMAC-lite. The window on the top left corner is the global observation space and the other windows show the observation space for each of the five agents.

3.2.5 Reward

The reward functions for SMAC and SMAC-lite are almost identical. We use a variation of the *shaped reward* function specified in SMAC [10], as this is the default reward function and is used in our `5m_vs_6m` experiments. SMAC and SMAC-lite award 10 points for each eliminated enemy unit and 200 points for each game that is won. Both award hit-point damage. SMAC awards hit-point damage computed with a complex function which accounts for enemy and friendly unit shields and health. SMAC-lite awards 1 point each time an enemy unit is hit with a projectile.

In addition to this, SMAC-lite awards -10 and -200 for each eliminated agent unit and for losing a game respectively. This encourages agents to avoid being eliminated and ultimately to win games. This has been added to compensate for a lack of complex hit-point reward function in SMAC-lite.

Chapter 4

Experiments, Results and Analysis

4.1 Experiment Setup

The purpose of our experiments is to understand the properties of SMAC-lite compared to SMAC's `5m_vs_6m` mini-challenge, in order to conclude if it can be considered a suitable alternative to `5m_vs_6m`. We devise the experiments based on specific criteria that must be satisfied in order for this to be the case.

- **MARL Benchmark Experiment:** This experiment tests the performance of IQL and QMIX between SMAC and SMAC-lite. The relative performance of these algorithms between both environments should be the same in order to conclude SMAC-lite can be used an alternative to `5m_vs_6m`. This tests the criteria that the two environments can be used interchangeably. We train both IQL and QMIX on both environments with the default PyMARL parameters as recommended by SMAC authors [10]. We report the *median test win rate* as our evaluation metric as this is recommended by the SMAC authors [10]. The test win rate is the proportion of times the agent team wins the game out of the total number of games played over a fixed number of training time-steps. This evaluation procedure is conducted at fixed regular intervals during training.

- **Timing Experiment:** This experiment tests the assertion that the SMAC-lite environment is faster to train with compared to the `5m_vs_6m` mini-challenge environment. This is the primary aim of this project and the second criteria that must be satisfied in order to justify SMAC-lite as an alternative to SMAC. This is evaluated by comparing the mean wall-clock time taken for SMAC and SMAC-lite to execute on the QMIX and IQL algorithms for a fixed number of time-steps.

We run both experiments using identical sets of hyperparameter values. These are the parameters specified in PyMARL and recommended by the SMAC authors [10].

4.1.0.1 Training Architecture and Hyperparameters

All agent networks use a deep recurrent Q-network made up of gated recurrent units with a 64 dimensional hidden state with a single fully connected hidden layer before and after the hidden layer. This is similar to the network proposed in [52]. The loss function used for training is root mean squared propagation (RMSProp), trained with a learning rate of 5×10^{-4}. All MARL algorithms are trained for approximately 2 million time-steps with a 100 time-step limit for each episode (the game timeout). Exploration is performed using ε-greedy action selection on each agent's own output. The value of ε is annealed from 1.0 to 0.05 over 20,000 time-steps and then fixed for the rest of the training schedule. The value of the discount factor γ is fixed to 0.99.

Our evaluation procedure is similar to that of [16] and SMAC [10]. Every 10,000 timesteps, we pause training and run 32 test episodes where agents greedily select actions with a decentralized mechanism. The proportion of times the agents win the game is reported as the test win rate.

Each set of experiments for the MARL benchmark experiment is run 10 independent times for statistical significance. We use the median test win rate to exclude outliers, as reinforcement learning experiments are known to have notoriously high variance [53]. For the timing experiment, we use the obvious choice of wall-clock time as our metric, as we are primarily interested in understanding if, given identical hardware and hyperparameters, SMAC-lite executes faster than SMAC. We conduct 5 independent training runs for each experiment set in the timing experiment. We choose this number to balance time constraints against statistical significance.

4.1.0.2 Experiment Hardware

All experiments for this project have been executed on Google Cloud computing infrastructure. For the MARL benchmark experiment, we run our experiments on a single NVIDIA Tesla P100 GPU on a Linux server with 16 CPU cores. For the timing experiment, we use a Linux server with 16 CPU cores without a GPU. We omit the GPU to reflect timings under more common hardware specifications. Although we use a large number of CPU cores for each training run in the timing experiment, this is close to the number of CPU cores found in many high specification laptops[1] designed for software development and research.

4.2 Results and Analysis

4.2.1 MARL Benchmark Experiment

Figure 4.1 and Figure 4.2 track the test win rate over 2 million training time-steps for `5m_vs_6m` and SMAC-lite respectively. QMIX outperforms IQL in both environments. This is perhaps unsurprising, as QMIX uses the centralized training with de-centralized training paradigm, allowing it to reason more adeptly over joint information from multiple agents [28]. It is interesting to note that the difference in performance between QMIX and IQL is greater for SMAC compared to SMAC-lite. This is difference is most likely due to differences between the `5m_vs_6m` and SMAC-lite environments, as other variables are fixed between both experiments. IQL, in particular, performs significantly better on the SMAC-lite environment. This may imply that IQL is particularly well-suited to SMAC-lite. Although it uses a naïve approach to MARL compared to QMIX, IQL has been shown to be a surprisingly strong benchmark in many MARL scenarios [54, 55]. Furthermore, QMIX performs more strongly on `5m_vs_6m` compared to SMAC-lite. This could be attributed to the differences in environment dynamics between SMAC and `5m_vs_6m`, particularly the behaviour of the enemy units, as this is where the environments diverge the most. However, both analyses on the relative strength of IQL and weakness of QMIX in SMAC-lite compared to SMAC is inconclusive and requires further investigation.

In general, SMAC-lite satisfies the functionality criteria for the MARL benchmark experiment, as it is qualitatively equivalent to SMAC in its performance rankings of our MARL benchmarking algorithms. In this regard, we can conclude SMAC-lite is a functionally suitable alternative to SMAC.

4.2.2 Timing Experiment

Table 4.1 shows the wall-clock timing statistics for the training times on both environments using both MARL algorithms. It shows that given identical hyperparameters, including total number of time-steps, SMAC-lite is significantly faster compared to

[1]`ultrabookreview.com/20056-core-i9-portable-laptops`

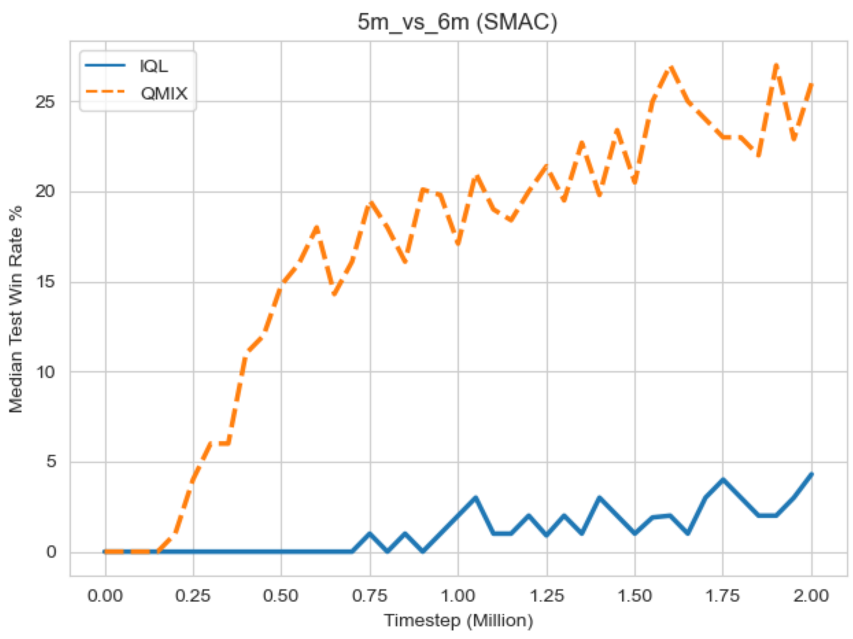


Figure 4.1: This is the median test win rate percentage plotted against the experiment time-steps for the `5m_vs_6m` SMAC environment for IQL and QMIX.

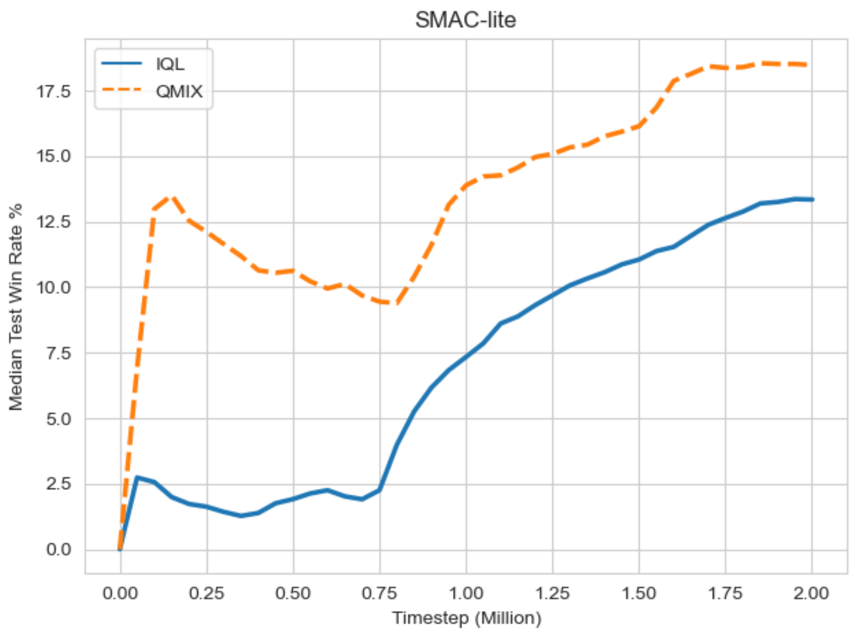


Figure 4.2: This is the median test win rate percentage plotted against the experiment time-steps for the SMAC-lite environment for IQL and QMIX.

		5m_vs_6m (SMAC)	**SMAC-lite**
QMIX	**Mean**	5:59:59	3:29:57
	Median	6:04:16	3:31:25
	Max.	6:10:52	3:39:32
	Min.	5:40:06	3:13:43
IQL	**Mean**	5:59:12	2:28:27
	Median	5:59:45	2:28:09
	Max.	6:05:20	2:31:41
	Min.	5:52:30	2:26:49

Table 4.1: This table lists the wall-clock time for training both the SMAC 5m_vs_6m and SMAC-lite environments on both QMIX and IQL. Timings are listed in the format Hours:Minutes:Seconds and are rounded to the nearest second. For example, the mean time taken to train QMIX on the 5m_vs_6m environment is 5 hours, 59 minutes and 59 seconds.

SMAC. This satisfies our second criteria to designate SMAC-lite as a suitable replacement for 5m_vs_6m. Specifically, on mean average, SMAC-lite is around 1.7× faster on QMIX and around 2.4× faster on IQL compared to SMAC. These are significant improvements on SMAC, and definitively shows SMAC-lite is a more computationally efficient test environment compared to SMAC.

Chapter 5

Conclusion and Further Work

This paper presents SMAC-lite, a new environment based on SMAC's `5m_vs_6m` mini-challenge for testing MARL algorithms oriented towards Dec-POMDPs. We motivate the need for SMAC-lite by highlighting its similarities and differences with SMAC. Following on from this, we devise experiments that allow us to reason about SMAC-lite and its relationship to `5m_vs_6m`. We conclude SMAC-lite is a suitable alternative to `5m_vs_6m` by showing that it is functionally similar to `5m_vs_6m`, as the performance rankings of the IQL and QMIX algorithms are preserved between the two. Moreover, we show that SMAC-lite is significantly faster compared to `5m_vs_6m`.

With this work, we aim to introduce a MARL test environment that can be used as a computationally efficient alternative to `5m_vs_6m`. In addition to this, we aim to lay the groundwork for an extensible efficient framework that has the same aims and values as SMAC, but is usable in practice by more researchers. One of SMAC's key strengths is its large suite of mini-challenges, which collectively provide a diversity of easy to challenging test environments. Despite the fact that SMAC-lite currently consists of a single environment, we hope it can be extended with further work to include a diverse set of environments. These environments could be novel or based on existing SMAC mini-challenges, or be a combination of the two, like SMAC-lite. We also encourage further work to understand the quantitative effects of differing environment dynamics between SMAC-lite and `5m_vs_6m` on the IQL and QMIX algorithms. This would likely uncover valuable insights into these algorithms and how to best use them.

In general, we have shown that it is possible to reap the benefits of SMAC with reduced computational resources.

Bibliography

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[2] Robert A Rescorla. A theory of pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. *Current research and theory*, pages 64–99, 1972.

[3] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[4] Taiwo Oladipupo Ayodele. Types of machine learning algorithms. *New advances in machine learning*, 3:19–48, 2010.

[5] Hamid Tizhoosh and Graham Taylor. Reinforced contrast adaptation. *Int. J. Image Graphics*, 6:377–392, 07 2006.

[6] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *ArXiv*, abs/1610.03295, 2016.

[7] Julien Pérolat, Joel Z Leibo, Vinicius Zambaldi, Charles Beattie, Karl Tuyls, and Thore Graepel. A multi-agent reinforcement learning model of common-pool resource appropriation. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[8] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[9] Santiago Ontañón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. A survey of real-time strategy game ai research and

competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in Games*, 5(4):293–311, 2013.

[10] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

[11] Zhen-Jia Pang, Ruo-Ze Liu, Zhou-Yu Meng, Yi Zhang, Yang Yu, and Tong Lu. On reinforcement learning for full-length game of starcraft. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019.

[12] Peng Sun, Xinghai Sun, Lei Han, Jiechao Xiong, Qing Wang, Bo Li, Yang Zheng, Ji Liu, Yongsheng Liu, Han Liu, and T. Zhang. Tstarbots: Defeating the cheating level builtin ai in starcraft ii in the full game. *ArXiv*, abs/1809.07193, 2018.

[13] Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *CoRR*, abs/1609.02993, 2016.

[14] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip H. S. Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1146–1155. JMLR.org, 2017.

[15] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018.

[16] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for

deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304. PMLR, 2018.

[17] Nantas Nardelli, Gabriel Synnaeve, Zeming Lin, Pushmeet Kohli, Philip HS Torr, and Nicolas Usunier. Value propagation networks. *arXiv preprint arXiv:1805.11199*, 2018.

[18] Yue Hu, Juntao Li, Xi Li, Gang Pan, and Mingliang Xu. Knowledge-guided agent-tactic-aware learning for starcraft micromanagement. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, page 1471–1477. AAAI Press, 2018.

[19] Kun Shao, Yuanheng Zhu, and Dongbin Zhao. Starcraft micromanagement with reinforcement learning and curriculum transfer learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 3(1):73–84, 2018.

[20] Christian Schroeder de Witt, Jakob Foerster, Gregory Farquhar, Philip Torr, Wendelin Boehmer, and Shimon Whiteson. Multi-agent common knowledge reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[21] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

[22] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre M. Bayen, and Yi Wu. The surprising effectiveness of MAPPO in cooperative, multi-agent games. *CoRR*, abs/2103.01955, 2021.

[23] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. QPLEX: duplex dueling multi-agent q-learning. *CoRR*, abs/2008.01062, 2020.

[24] Tonghan Wang, Heng Dong, Victor R. Lesser, and Chongjie Zhang. ROMA: multi-agent reinforcement learning with emergent roles. *CoRR*, abs/2003.08039, 2020.

[25] Yaodong Yang, Jianye Hao, Ben Liao, Kun Shao, Guangyong Chen, Wulong Liu, and Hongyao Tang. Qatten: A general framework for cooperative multiagent reinforcement learning. *CoRR*, abs/2002.03939, 2020.

[26] Yali Du, Lei Han, Meng Fang, Ji Liu, Tianhong Dai, and Dacheng Tao. Liir: Learning individual intrinsic reward in multi-agent reinforcement learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[27] Chris Bamford, Shengyi Huang, and Simon M. Lucas. Griddly: A platform for AI research in games. *CoRR*, abs/2011.06363, 2020.

[28] Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V. Albrecht. Comparative evaluation of multi-agent deep reinforcement learning algorithms. *CoRR*, abs/2006.07869, 2020.

[29] Jonathan Baxter, Andrew Tridgell, and Lex Weaver. Reinforcement learning and chess. In *Machines that learn to play games*, pages 91–116. 2001.

[30] Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994.

[31] Justin K. Terry, Benjamin Black, Ananth Hari, Luis S. Santos, Clemens Dieffendahl, Niall L. Williams, Yashas Lokesh, Caroline Horsch, and Praveen Ravi. Pettingzoo: Gym for multi-agent reinforcement learning. *CoRR*, abs/2009.14471, 2020.

[32] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. Benchmarking multi-agent deep reinforcement learning algorithms. 2020.

[33] Stefano V Albrecht and Subramanian Ramamoorthy. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. *arXiv preprint arXiv:1506.01170*, 2015.

[34] Stefano V Albrecht and Peter Stone. Reasoning about hypothetical agent behaviours and their parameters. *arXiv preprint arXiv:1906.11064*, 2019.

[35] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of Reinforcement Learning and Control*, pages 321–384, 2021.

[36] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. Multi-agent reinforcement learning: An overview. *Innovations in multi-agent systems and applications-1*, pages 183–221, 2010.

[37] Jeffery A Clouse. Learning from an automated training agent. In *Adaptation and learning in multiagent systems*. Citeseer, 1996.

[38] Bob Price and Craig Boutilier. Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research*, 19:569–629, 2003.

[39] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[40] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International conference on machine learning*, pages 4295–4304. PMLR, 2018.

[41] Frans A Oliehoek and Christopher Amato. *A concise introduction to decentralized POMDPs*. Springer, 2016.

[42] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[43] Cinjon Resnick, Wes Eldridge, David Ha, Denny Britz, Jakob Foerster, Julian Togelius, Kyunghyun Cho, and Joan Bruna. Pommerman: A multi-agent playground. *arXiv preprint arXiv:1809.07124*, 2018.

[44] Peter Stone, Gregory Kuhlmann, Matthew E Taylor, and Yaxin Liu. Keepaway soccer: From machine learning testbed to benchmark. In *Robot Soccer World Cup*, pages 93–105. Springer, 2005.

[45] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The robot world cup initiative. In *Proceedings of the first international conference on Autonomous agents*, pages 340–347, 1997.

[46] Shivaram Kalyanakrishnan, Yaxin Liu, and Peter Stone. Half field offense in robocup soccer: A multiagent reinforcement learning case study. In *Robot soccer world cup*, pages 72–85. Springer, 2006.

[47] Matthew Hausknecht, Prannoy Mupparaju, Sandeep Subramanian, Shivaram Kalyanakrishnan, and Peter Stone. Half field offense: An environment for multiagent learning and ad hoc teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*, volume 3. sn, 2016.

[48] Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Shared experience actor-critic for multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[49] Alexander Watson. Deep learning techniques for super-resolution in video games. *arXiv preprint arXiv:2012.09810*, 2020.

[50] Dominic Filion and Rob McNaughton. Effects & techniques. In *ACM SIGGRAPH 2008 Games*, pages 133–164. 2008.

[51] Xiao Cui and Hao Shi. A*-based pathfinding in modern computer games. *International Journal of Computer Science and Network Security*, 11(1):125–130, 2011.

[52] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*, 2015.

[53] Johan Bjorck, Carla P Gomes, and Kilian Q Weinberger. Is high variance unavoidable in rl? a case study in continuous control. *arXiv preprint arXiv:2110.11222*, 2021.

[54] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

[55] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. *arXiv preprint arXiv:1702.03037*, 2017.