Universiteit Leiden

Master Computer Science

[Automated City-Planning via Dynamic RL in *Micropolis*]

Name: [Samy Hashim]
Student ID: [S3458717]

Date: [22/12/2023]

Specialisation: [Artificial Intelligence]

1st supervisor: [Mike Preuss]
2nd supervisor: [Matthias Müller-Brockhausen]

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Over the last few years, Reinforcement Learning (RL) has begun to showcase stellar achievements in various problem-solving domains, including video game environments. While RL has excelled historically in well-structured and relatively simplistic settings, its usage in complex and dynamic environments presents a broad new frontier of innovation and exploration. This study seeks to explore the utilisation of dynamic RL within the context of a complex gaming environment, specifically focusing on *Micropolis*, a city-building simulation. By utilising the Synchronous Advantage Actor-Critic (A2C) and Proximal Policy Optimization (PPO) algorithms, the aim is to develop and evaluate an automated RL agent via a standardised framework which can serve as a reproducible starting point for further research and development.

The methodology utilised in this study involves supplying the agent with a comprehensive representation of the game environment, alongside defining a reward structure based on various applicable metrics, such as population and zoning diversity. Through heavy iterative experimentation and refinement, we attempt to advance the application of dynamic RL in complex interactive environments. By doing so, we hope that this work will help contribute to advancements in Artificial Intelligence (AI) that extend beyond simple gaming environments, demonstrating the potential of RL as powerful tool for tackling complex challenges in real-world scenarios.

All code used can be found at `https://github.com/MinasMayth/gym-city`.

Contents

1 Introduction

In recent years, RL has a emerged as a prominent cornerstone in the field of AI, showcasing great achievements in various domains. Groundbreaking examples like AlphaZero [Silver et al., 2017] and Dreamer [Hafner et al., 2024] indicate the immense potential of RL algorithms in tackling complex challenges with an unprecedented level of efficiency and adaptability. This surge in research has spurred exploration into a diverse array of applications, extending beyond "traditional" domains into realms like video games, where the prowess of RL has come to be highly respected [Torrado et al., 2018].

The utilisation of RL techniques in gaming environments has been particularly impressive, with notable projects like the OpenAI Atari library showcasing the newfound impact of RL in this domain [Mnih et al., 2013]. These avenues of research have not only extended the boundaries of what is achievable in RL-based gaming, but have also served as prototypes for exploring fundamental principles of RL in dynamic environments. By leveraging RL methods, agents can interactively learn to navigate intricate game situations, adapt strategies in response to changing scenarios, and ultimately achieve superhuman performance across a broad range of gaming challenges.

However, while RL has demonstrated remarkable successes in well-defined, single-objective settings, its application to complex gaming environments presents a whole new frontier of exploration and innovation. Dynamic RL tasks involve environments where conditions and objectives continuously change and evolve, requiring agents to adapt their strategies in real-time. The search for solutions that can handle such fluid challenges is inherently difficult, yet simultaneously holds immense promise for real-world applications where conditions are rarely static in nature.

In this context, our research attempts to explore the utilization of dynamic RL within the complex gaming environment of *Micropolis*, a city-building simulation. *Micropolis* presents an intriguing testing environment for RL algorithms due to its complex and multifaceted nature, where our agent must manage zoning, infrastructure, connectivity and population growth simultaneously. The dynamic interactions between these varying aspects of the game create a rich and unpredictable environment that is suitable for developing RL strategies in more advanced settings.

Our study focuses on utilising modern RL libraries, such as StableBaselines3 (henceforth referred to as SB3), to develop and evaluate a RL agent capable of efficiently learning and adapting within the *Micropolis* environment. Unlike much simpler, static tasks, the agent in *Micropolis* must continuously adjust its strategies based on the current state of the city, balancing short-term gains with long-term sustainability. This requires the neural network to accommodate a deep understanding of the intricate dynamics at play in the environment, with the capacity of responding to unforeseen changes and challenges.

A significant component of our research is the usage of the SB3 library, which provides a comprehensive toolset of RL algorithms implemented in a modular framework. SB3 promotes the development of RL agents by offering robust tools for policy optimization, model evaluation and environment management. This allows for a far more streamlined and efficient experimentation process than more traditional approaches, enabling rapid testing and refinement of algorithms. Additionally, the highly automated nature of the training process ensures that what the agent learns is mainly driven by environmental interactions and the selection of algorithm and

hyperparameter values. This level of autonomy in training is important for developing RL agents that can operate independently and safely in real-world scenarios where human oversight is minimal or even unavailable.

Through the process of iterative experimentation and refinement, we aim to show how RL can effectively be applied to a dynamic, complex environment. This includes not only fine-tuning our reward functions and hyperparameters [Sections 4.2.3, 4.3.3] but also employing advanced techniques such as environment vectorization to enhance the agent learning process. By running multiple instances of the *Micropolis* environment in parallel, our agent can experience a wider variety of scenarios in a shorter amount of time, leading to a more robust and generalisable approach [Section 4.1].

The general aim of this research is to contribute to the growing body of knowledge in the domain of dynamic RL, hopefully helping foster advancements that could transcend gaming and extend into real-world domains. By developing RL agents that can handle the dynamic environment of *Micropolis*, we aspire to highlight the potential of RL as a powerful tool for tackling complex challenges in AI and beyond. Furthermore, by using a standardised library such as SB3, this study hopes to help standardise the various RL approaches developed over the last few decades and foster an easier environment for future researchers to build upon, rather than having to build their own algorithms from scratch again.

We seek to address several core research questions: How can we design reward functions that effectively balance short-term and long-terms objectives in a dynamically changing environment? What kind of strategies can be employed to ensure that RL agents can generalise their learning to a wide array of scenarios within an environment? What kind of process does it take to develop an agent capable of solving these kind of environments, and how can we apply this knowledge to other tasks and situations? In answering these questions we attempt to apply and standardise some of the possible research methodologies for using RL in city-planning specific applications, in the hope of enabling future RL-practitioners to build upon existing work more effectively.

Moreover, the application of RL to *Micropolis* is not merely an exercise in theory but also has practical implications for urban planning. The ability of RL agents to simulate and optimise city development approaches can inform real-world decision-making processing, providing a valuable tool for city planners and policymakers alike. We hope that the insights gained from this research will contribute to the design of more sustainable and efficient urban environments, showcasing the broader societal impact of technological advancements in RL.

1.1 Micropolis

Figure 1: *Micropolis*, a city-building game[1]

Micropolis is the RL environment used in this study and is an open-source clone of *SimCity Classic*[2], a famous city-building game developed by Will Wright and published by Electronic Arts in 1989 [Wright and Joffe, 1989]. A normal game begins with a random, procedurally generated landmass with possible bodies of water ranging from rivers to lakes to seas. The game features a square grid of discrete tiles where players can construct power plants, power lines, roads, rails, various zoning (residential, commercial, or industrial), and services like fire stations or police departments, parks, stadiums, and airports. In addition to this there is a "query" tool which a player can use to inspect the density, value, crime, growth and pollution rates of any tile on the map.

Zoning in a tile can only develop if powered, which is distributed from power plants via adjacent zones, buildings and power lines. The type and degree of development depends on various different factors such as how near the zone is to roads or railways, parks or even water bodies. Furthermore, tax rates, crime and pollution levels, and city-wide demand all influence existing development ratios as would happen in a real city. Unlike later iterations of *SimCity* where new population enters the city via a road connected to the outside world, in the first version of the game new population simply appears in it's assigned residential zoning at the start of a game. Later on when transport buildings such as airports and ports are constructed, a visual representation of planes and boats moving to and from such buildings will indicate population change. A rail system is also implemented in the game, but it is not necessary to build any train stations, only tracks. The simulation will then automatically generate a railway car that moves through different neighbourhoods of the city and facilitates transport.

[1] Image sourced from `http://wiki.laptop.org/go/Image:Simcity_screenshot2.png`

[2] `https://web.archive.org/web/20140403150258/http://www.simcity.com/en_US/product/simcity-classic`

City builders start with a preset amount of money, depending on the difficulty level, which they can then use to start building their city. Game difficulty can be set to easy, normal or hard, which simply adjusts how much money players start with, the harder the difficulty the less money they receive. Players later earn revenue through taxation once an initial population has been established. Beyond just construction players can also adjust tax rates and the budget for individual services such as fire or police, increasing/decreasing their effectiveness. Traffic flows between zones and buildings connected via road tiles, being a relatively simple system in this version of the game but still serving as an indicator for how population is moving between different areas.

Amenities and services such as fire departments and police are distributed via a radius of influence. In later *SimCity* games, when a specific crime, sickness or fire occurs in a given zone a service vehicle will be dispatched from the nearest relevant building until it reaches the problem area and resolves the problem. The first iteration by Wright did not contain the complexity for this functionality and thus any placed police or fire department will simply reduce risk within a certain area surrounding it, with the reduction being stronger the closer one is to the building.

In addition to this, the *Micropolis* environment incorporates various disasters, such as fires, earthquakes, and monster attacks, which add a layer of complexity and unpredictability to the game. This has become somewhat of a trend in the *SimCity* games, especially with occurrences such as alien or monster attacks serving as a somewhat absurd element of danger. In terms of utilisation for RL, if added to the training-runs these events could challenge the agent to develop robust strategies for disaster prevention, management and recovery alongside its normal city building nature. While we will not include any of these events in the current study, this could provide an interesting avenue for further research.

The dynamic nature of *Micropolis*, with its many interconnected systems and evolving challenges, make it a suitable environment for exploring the capabilities of RL in urban planning. The game requires the agent to make quick decisions that not only address short-term, immediate needs but also take into account the long-term consequences. We hope that this will facilitate the development of strategies that balance growth, sustainability and resilience. This level of complexity begins to mirror real-world urban planning scenarios, providing a rich environment for advancing RL techniques.

2 Related Works

The foundation of this research lies in the work conducted by Sam Earle in his paper titled "Using Fractal Neural Networks to Play Micropolis and Conway's Game of Life at Variable Scales" [Earle, 2020]. Earle's study delved into the realm of machine learning applied to automata-based games, focusing specifically on *Micropolis* and Conway's Game of Life [3]. The primary objective of that research was to develop a transferable machine learning model capable of mastering these intricate games using a novel approach centered around fractal neural networks. [4]

[3] See `https://conwaylife.com/`

[4] Without his paper none of our work would have been possible, and thus we are extremely grateful for the work he has done.

Previous work done has laid the groundwork for our own exploration into the application of machine learning techniques, particularly RL, within the context of *Micropolis*. Earle's emphasis on transferability underscored the importance of developing models that can generalize across different game environments, which we do take note of, but the overall aim of our work is more shifted towards the standardization and reproducibility of RL algorithms via the SB3 library and thus our focus lies in a different direction.

Through a thorough examination of prior work, key insights and methodologies that inform our own approach are identified. This includes, for example, the utilisation of the Power Puzzle mini-game and the focus on different variations of the reward function (see Section 4.3.2). Furthermore, we desire to expand upon previous work by exploring the challenges and opportunities presented by the dynamic nature of *Micropolis* with the toolset provided by the SB3 library. We would like to note that while Earle's paper served as a critical starting point for our exploration, our research has moved in a direction independent of his and we seek to distinguish ourselves by performing our own avenues of research and re-evaluating many prior assumptions to consider if they still fit into our contemporary framework.

The rest of this section will be a general literature review of the studies relevant to this paper. We will start with some domain-specific applications by studying existing implementations of dynamic RL and the usage of RL in urban planning and simulation games. Then we will focus on some important techniques such as the exploration/exploitation trade-off and the specific algorithms used in this paper. Finally, we will consider literature on the SB3 library itself. We would like to note that in no manner is this list of selected papers exhaustive, as there is an extensive body of literature covering the various topics addressed in this review.

2.1 Domains of Application

This section explores the various domains where RL has been used in research, focusing on urban planning, video and board games, and other dynamic environments.

2.1.1 Dynamic RL

Dynamic RL can be described as the domain of RL that addresses environments which are typically complex in nature and will change over time. A pertinent study in this area is "Model-Based Reinforcement Learning for Atari", which highlights the application of RL in video games, which are some of the most dynamic and complex environments currently available to us [Kaiser et al., 2024]. This study demonstrates how RL agents are able to adapt to varying game dynamics. The researchers show that incorporating dynamic elements into their algorithms improve performance and adaptability, which ultimately provides good insights into how similar methods could be applied to urban planning scenarios where the environment is also constantly evolving.

Further advancements in dynamic RL include its usage in the area of network function scheduling and risk-sensitive optimization. The study "Dynamic VNF Scheduling: A Deep Reinforcement Learning Approach" introduces a deep RL approach to solve the virtual network function scheduling problem in complex, dynamic scenarios, showcasing improved performance compared to more traditional approaches [Zhang et al., 2023].

Bellinger et al., in their paper "Dynamic Observation Policies in Observation Cost-Sensitive

Reinforcement Learning" address the potentially high costs of state measurement in RL [Bellinger et al., 2024]. The authors propose a policy that reduces the number of decision steps and measurements needed for their algorithms to function, improving efficiency in complex applications like robotics and environment monitoring. This research highlights the importance of managing the complexity of the observation space in dynamic environments and proposing resource-saving approaches to optimise training.

In "Reinforcement Learning with Human Feedback: Learning Dynamic Choices via Pessimism" [Li et al., 2023], the authors discuss learning dynamic choices through human-integrated feedback in offline RL environments. This paper proposes a method that has the agent estimate human behaviour policies and state-action value functions, then calculates the reward function to optimise policy decisions. This integration of human feedback into dynamic RL frameworks is important for real-world application and connects to some of the work shown in [Earle, 2020], where Earle does occasionally influence the agent during its training process to optimise learning.

2.1.2 RL in Urban Planning

Urban planning is a quite dynamic and complex field that can highly benefit from the application of advanced machine learning methods, including RL. An important paper in the subdomain of traffic management is the study by Wei et al. titled "Intellilight: A reinforcement learning approach for intelligent traffic light control", which demonstrates how RL can be utilised to greatly reduce traffic congestion by learning optimal traffic light policies [Wei et al., 2018]. Another paper in the same research area, the study "Urban Path Planning Based on Improved Model-based Reinforcement Learning Algorithm" by Wang et al. instead focuses on mitigating traffic congestion through advanced path planning techniques using RL algorithms [Wang et al., 2023].

Further advancements in the field include RL research in autonomous driving and vehicle speed planning. The study "Speed planning for connected and automated vehicles in urban scenarios using deep reinforcement learning" by [Li et al., 2022] proposes a strategy to optimize factors such as fuel economy, driving safety, and travel efficiency of hybrid electric vehicles. Another relevant study is "Action and Trajectory Planning for Urban Autonomous Driving with Hierarchical Reinforcement Learning" by Lu et al., which seeks to address decision-making in highly complex scenarios with the presence of other vehicles and obstacles [Lu et al., 2023].

Another significant contribution is the work on optimizing energy consumption in smart cities using RL. The paper "Reinforcement learning for demand response: A review of algorithms and modeling techniques" by Ramón Vázquez-Canteli & Nagy explores how RL can be utilized to manage energy distribution efficiently in urban settings [Vázquez-Canteli and Nagy, 2019].

Not much work has been done with RL in the area of city planning itself, which is why we believe the paper by Earle to be a pioneering work [Earle, 2020]. An RL algorithm that can function on the complex level of a city-building simulation and simultaneously manage energy consumption, traffic flow and path planning would be able to combine approaches from a variety of previous studies to establish much more comprehensive strategies to working with urban environments.

2.1.3 RL in Gaming

Games, which are often good examples of dynamic environments, present a unique and fascinating challenge for RL due to their complex and dynamic natures. RL approaches have shown significant promise in playing and mastering these games by learning strategies via interaction with the game environment.

Highly notable examples include the use of RL in games like *DoTA* and *StarCraft II*. In these games, a RL agent must learn to manage resources, make tactical and strategic decisions, and adapt to dynamically changing conditions. Techniques like deep Q-networks (DQN) have been vital in allowing agents to learn from high-dimensional inputs, such as game states represented via pixel data. The foundational work by Mnih et al. (2013) introduced DQN's effectiveness in playing Atari games from raw pixel inputs [Mnih et al., 2013]. Another significant contribution to the field by Mnih et al. is their paper on achieving human-level control in certain video games [Mnih et al., 2015], showcasing technical advancements achieved by the authors.

Model-based RL approaches, such as those demonstrated in "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" [Schrittwieser et al., 2020], work with predictive models of the environments to enhance the efficiency of learning. These models often cannot be implemented in complex gaming environments, but when they can be they allow the RL agent to easily simulate potential future states and make more informed decisions, a capability which can be highly useful in situations where foresight and planning are crucial.

As discussed earlier, Earle's *Micropolis* paper is also a good example of RL usage in gaming [Earle, 2020]. Another paper co-authored by Earle is "Exploring open-ended gameplay features with Micro RollerCoaster Tycoon" [Green et al., 2021], where the authors attempted to use RL to help build theme parks in an open-source version of *Rollercoaster Tycoon*, with a focus on using evolutionary algorithms. While there was a wide diversity in high-performing designs, the authors were able provide some interesting insights regarding different evolutionary initialization and growth methods, alongside the effectiveness of resource constraints on agent development.

A further example is the paper from Silver et al., which showcases RL's capability to achieve superhuman performance in strategic games through self-play [Silver et al., 2017]. Also Tessler et al. were able to show that RL agent could improve their performance by integrating external sources knowledge, such as gaming instruction manuals, into their training process [Tessler et al., 2016]. Further worthy mentions are landmark projects such as AlphaGo [Silver et al., 2016], which utilized deep RL to master the game of Go, and OpenAI's *DoTA 2* bot [OpenAI et al., 2019a], which showcased the capability of RL to handle real-time strategy games. These projects highlight the potential of RL to tackle intricate tasks through continuous learning and adaptation.

Micropolis itself is a standard example of a complex simulation game with strategic thinking involved. While it contains different mechanics to RTS games such as *Starcraft II* or board games like Chess, RL agents still are well suited to working with these kind of environments due to the easily definable observation and action spaces, alongside the interactive nature of the game that bases development and progress on user play and strategies.

2.2 Algorithms and Techniques

This section dives into the specific algorithms and some pertinent techniques used in RL, focusing on exploration/exploitation trade-off, efficient and stable training, and the Synchros Advantage Actor-Critic (A2C) and Proximal Policy Optimization (PPO) algorithms implemented in this research.

2.2.1 Exploration/Exploitation Trade-off

In RL, the exploration/exploitation trade-off is a foundational concept where a given agent must balance between exploring their environment to discover new strategies and exploiting its current knowledge to maximise reward. Exploration can, for example, involve trying completely new actions to gain more information about the simulation, which can lead to discover new and efficient strategies. On the other hand, exploitation will allow for an agent to maximise reward on a local optimum to its fullest potential. Any useful algorithm must achieve the right balance between these two extremes to function effectively.

Several useful approaches have been proposed to manage this trade-off. Epsilon-greedy is a simple yet well-established approach where the agent explores with probability ϵ and otherwise follows an exploitation strategy [Watkins, 1989]. More advanced approaches include Upper Confidence Bound, which determines actions based on a combination of their estimate value and how certain that estimate is [Auer, 2002], and Thompson Sampling, which utilises the concept of probability matching to choose actions based on how likely they are to be optimal [Thompson, 1933].

In highly advanced RL algorithms such as PPO and Soft Actor-Critic (SAC) [Haarnoja et al., 2018], mechanisms such as entropy regularisation are used to encourage agent exploration by adding an entropy coefficient to the objective function of the algorithm which penalises more deterministic policies. PPO also utilises a clipping mechanism to stabilise gradient updates and balance exploration and exploitation. These methods have been vital in achieving a balance that adapts to the needs of the environment, thus enhancing agent learning and performance. A notable paper is "Deep Exploration via Bootstrapped DQN" by Osband et al., which introduces Bootstrapped Deep Q-Networks (Bootstrapped DQN) to drive deep exploration strategies, indicating quite significant improvements in exploration efficiency [Osband et al., 2016]. Another study of note is that of [O'Donoghue et al., 2017], which showed that an agent with double uncertainty estimates develops more effective exploration strategies. In our research the A2C and PPO algorithms employ the entropy regularisation mechanism to great effect, with it being one of the hyperparameter values optimised via our Hyperparameter Optimisation (HPO) strategies.

Finally, we can also briefly mention that the concept of transfer learning in RL is also important in improving exploration efficiency. Taylor and Stone discuss how learned knowledge from previous tasks can be used to effectively enhance exploration in new tasks, making the exploration-exploitation trade-off more efficient [Taylor and Stone, 2009]. While transfer learning is not an important component of this study, it is a major aspect of [Earle, 2020], and future research might seek to employ this concept to more effectively generalise urban planning strategies to new tasks.

2.2.2 Efficient and Stable Training for RL

Recent advancements in RL have also focused on improving how efficient and stable training is. Techniques like PPO and SAC have been critical in creating stable training processes and enhancing the overall performance of RL agents [Schulman et al., 2017]. These methods address some of the inherent challenges in RL, such as sample efficiency and exploration-exploitation balance, making them highly relevant to all RL-based research, including our own.

Moreover, the use of model-based RL, as explored by Hafner et al. in the widely popular Dreamer models [Hafner et al., 2024], has shown that incorporating predictive models of the environment can significantly improve the efficiency of learning. By predicting future states and rewards, model-based approaches enable agents to plan and make informed decisions.

2.2.3 Planning Vs Learning RL Algorithms

In the realm of RL, algorithms can generally be split into two broad definitions: planning algorithms and learning algorithms. Planning algorithms, such as dynamic programming (not to be confused with dynamic RL) and Monte Carlo tree search, aim to compute an optimal policy by explicitly modeling the dynamics of the environment and searching for the best sequence of actions to maximize cumulative reward [Torrado et al., 2018]. These algorithms have been successful in domains where the environment's dynamics are well-understood, well-defined and relatively static.

On the other hand, learning algorithms, like Advantage-Actor Critic [Mnih et al., 2016] and DQN [Mnih et al., 2015], learn an optimal policy through interaction/play with the environment, gradually updating their strategies over time based on observed experiences. These algorithms are well-suited for environments where the dynamics are complex, partially observable, or subject to change. Learning algorithms are especially suitable for video game environments in which the possible state space is far too complex to completely model, and thus learning via interactions is the preferred method of training.

2.2.4 Synchronous Advantage Actor-Critic (A2C)

A2C has become one of the most widely used learning algorithms in RL due to its robustness and simplicity. Asynchronous Advantage Actor-Critic (A3C), from which A2C is derived, was introduced by Mnih et al in their paper "synchronous methods for deep reinforcement learning" [Mnih et al., 2016]. Researchers built upon A3C by synchronizing parallel actor-learners, which effectively reduced the computational complexity of the algorithm while still retaining all of the benefits of the actor-critic framework [5]. In this type of framework, the "actor" network updates the policy directly, while the "critic" network evaluates the policy by estimating a given value function.

A2C has showcased significant success in a variety of benchmark environments, including those of the Atari 2600 games, where it has convincingly outperformed traditional Q-learning algorithms [Mnih et al., 2016]. The ability of A2C to handle both discrete and continuous action spaces makes it quite the versatile tool for a wide range of RL tasks. Additionally, A2C's synchronous nature ensures more stable and consistent updates than it's predecessor, which

[5] See `https://openai.com/index/openai-baselines-acktr-a2c/`

can be very beneficial in complex or network-based environments where asynchronous updates might lead to high instability.

The application of A2C in our research leverages its strengths in stable policy updates and efficient handling of high-dimensional state and action spaces. In the context of *Micropolis*, these attributes are crucial for developing an RL agent that can learn effective policies in a highly dynamic and intricate environment. By utilizing A2C, the aim is to achieve a balance between exploration and exploitation, enabling the agent to adapt to the evolving conditions of the simulation and optimize its performance across multiple objectives.

Furthermore, the usage of A2C is extremely wall-clock time efficient compared to other algorithms such as PPO when using vectorized environments, decreasing training time by many factors. Since A2C is optimised to run on the CPU and with multiple workers it can effectively utilise environment vectorization while still being extremely stable and providing good results.

2.2.5 Proximal Policy Optimization (PPO)

PPO is another prominent learning algorithm in the realm of RL, and a variant of A2C. Introduced by Schulman et al. [Schulman et al., 2017], PPO addresses some of the inefficiency issues found in earlier iterations of policy gradient methods. PPO optimizes the policy by performing multiple epochs of stochastic gradient ascent on objective functions, but it makes sure that the updates are not too large by using a clipped objective function. This clipping mechanism helps in maintaining a balance between exploration and exploitation, preventing drastic policy changes that could destabilize learning.

The effectiveness of PPO has been demonstrated in various challenging environments. As mentioned above, OpenAI's applications of PPO in their DoTA 2 bots have showcased the algorithm's ability to handle complex, multi-agent environments with high-dimensional state and action spaces [OpenAI et al., 2019a]. Moreover, PPO has been successfully applied in robotics, where it has been used to train agents to perform continuous control tasks, further proving its versatility and robustness [OpenAI et al., 2019b].

The widespread adoption of PPO in both academic research and practical applications highlights its strengths in dealing with complex, dynamic environments. Its ability to maintain stability while learning efficient policies makes it quite an ideal choice for environments like *Micropolis*, where the environment's complexity and the need for adaptability are paramount. By leveraging A2C and PPO, our research aims to harness the strengths offered by these algorithms to develop an RL agent capable of mastering the intricate dynamics of simulated urban planning.

2.3 Stable-Baselines3 (SB3)

The final section of this literature review is dedicated to the Python RL library utilised in this study. SB3 is an open-source RL framework and the successor of Stable Baselines, providing a reliable and standardized framework for research and practical applications alike. The library has been instrumental in standardizing RL research, making experiments more reproducible and alleviating stress for future RL practitioners. Developed by Raffin et al., SB3 offers robust implementations of popular algorithms such as PPO, A2C, and others, making it highly useful for anyone working with Open AI Gym environments.

The paper "Stable Baselines3: Reliable Reinforcement Learning Implementations" by Raffin et al. introduces and provides a comprehensive overview of the library and its capabilities [Raffin et al., 2021]. The authors highlight SB3's design principles, including ease of use, simplicity, and flexibility. The library's modular architecture allows users to easily implement and test new algorithms and their own custom environments, allowing for rapid experimentation and development without long set-up times.

A highly significant advantage of SB3 is its emphasis on reproducibility. The library includes standardized training scripts and detailed documentation, ensuring that experiments can be easily replicated. This focus on reproducibility addresses a common challenge in RL research, where the lack of standardized practices often leads to inconsistent results. By providing a reliable and consistent framework, SB3 helps to advance the field by enabling more rigorous and comparable research studies.

The effectiveness of SB3 is further demonstrated through various studies and experiments. For example, in the domain of robotics, researchers have used SB3 to develop algorithms for real-time control of power systems, ensuring stability and efficiency in voltage regulation tasks [Feng et al., 2012]. The earlier study mentioned in this paper by [Zhang et al., 2023] actually applied SB3 to dynamic network function scheduling, achieving significant performance improvements over non-standardized methods.

Further research has also integrated SB3 into various innovative applications. Mukherjee and Vu present a stability-guaranteed distributed RL framework for interconnected linear subsystems, which relies on SB3 for its RL algorithms [Mukherjee and Vu, 2021]. Their work demonstrates the use of SB3 in ensuring the stability of complex systems through distributed learning. Shah et al. discussed stable RL in unbounded state spaces, which is particularly relevant for large-scale systems [Shah et al., 2020]. Their research highlights SB3's ability to handle large state spaces and maintain stability.

Osinenko et al. provided a critical examination of stabilizing RL methods, contributing to the discourse on how SB3 can be employed to ensure stability in RL applications [Osinenko et al., 2022]. Their work underscores the importance of stability in practical RL deployments and how SB3 can be part of the solution. Beckenbach et al. (2022) proposed a stabilizing RL approach for sampled systems with partially unknown models, further validating SB3's utility in real-world scenarios where complete system knowledge is often unavailable [Beckenbach et al., 2022].

SB3's modularity and flexibility also allow for integration with other tools and frameworks, such as OpenAI Gym and PyTorch, enhancing its usability in diverse research settings. This interoperability ensures that SB3 can be easily adapted to a wide range of applications, from academic research to industrial deployment. We advocate for the widespread usage of SB3, especially for the above-mentioned reasons of standardization and reliability.

3 Methodology

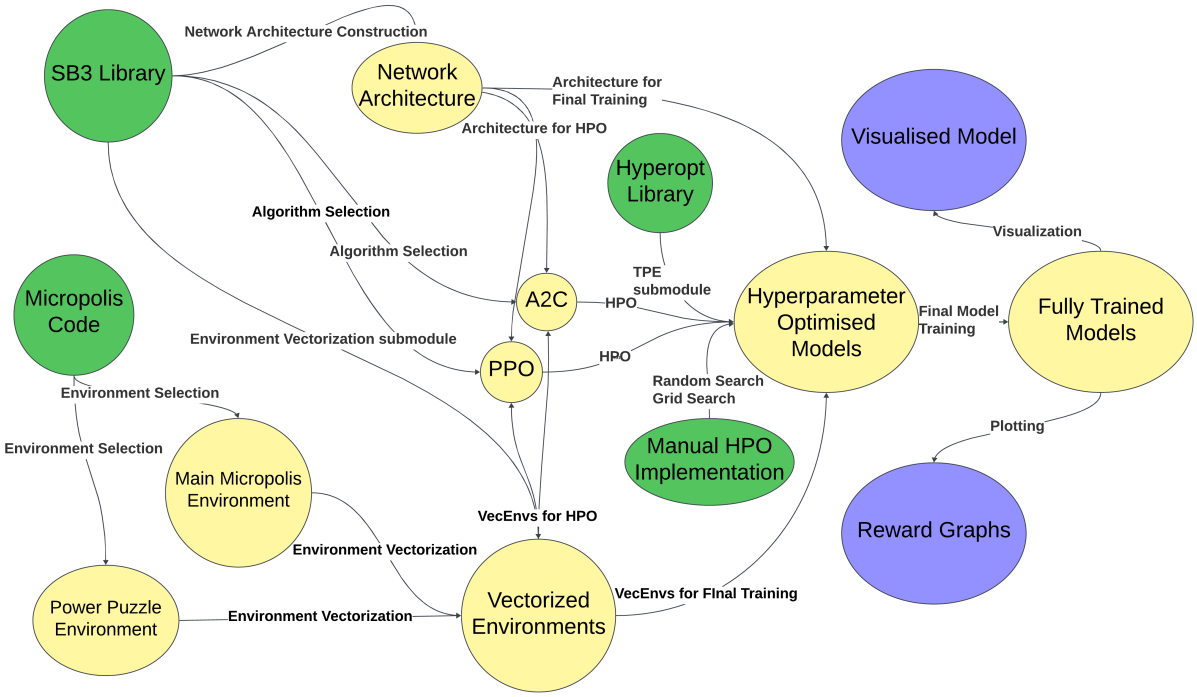


Figure 2: General concept map of our system. Code sources are displayed in green, while final products generated are displayed in blue.

This section provides a comprehensive overview of the methodology employed in this research, focusing on the integration of RL techniques within the context of the *Micropolis* environment,. The methodology encompasses three main components: the *Micropolis* environment itself, the utilization of the SB3 RL library, and the specifics of agent training. The integration of RL into *Micropolis* presents unique challenges and opportunities, requiring a nuanced understanding of both the game's design principles and the capabilities of modern RL algorithms.

3.1 *Micropolis* Environment

Micropolis was released as an open-source software in 2008 under the One Laptop Per Child Program. Developer Don Hopkins refactored the code, incorporating a Python-based GUI. The game engine, written in C++, communicates with Python via a predefined set of shared functions using Swift. These functions enable Python to initialize, call, and query the game engine, allowing our environment to interact with and observe the game map.

Moreover, human players can modify the map through the GUI during both training and inference stages. The agent acknowledges these modifications in real-time, treating the player's actions as its own during training. Practically, this means actions initiated by the player, such as building structures on specific tiles, are queued for the agent to execute at the next available opportunity, instead of selecting actions randomly from its distribution. Consequently, the agent can learn to favor or disfavor certain player actions based on their impact on rewards, mirroring its learning process for its own actions [Earle, 2020].

In a game as open-ended as *Micropolis*, the objective of human play often involves understanding and mastering the simulation's underlying principles. Humans naturally have a simplistic understanding of how cities are supposed to be built, most likely due to lifelong observations of their own environments. Thus they may find it easier to learn how cities should be structured even if they have no prior city-planning experience. However, when integrating an RL agent into the game, a completely new challenge arises in navigating through the complex interplay between the game's design and the agent's neural architecture, which presents its own set of problems and opportunities for exploration [Friedman, 1999].

3.2 StableBaselines3

Even though access to previous work was available, obtaining a satisfactory baseline for city development proved to be extremely difficult. The original Python environment was recreated as closely as possible using Anaconda and deploying a Python 3.6 environment on the Leiden ALICE (Academic Leiden Interdisciplinary Cluster Environment) server. However, the custom networks used by Earle and provided in his final repository update did not work. After setting up and testing the *Micropolis* gym environment, no good training results were obtained. This was frustrating, as we had hoped at least recreate the level of performance displayed by previous work in this area. After an email discussion with Earle, his suggestion of using SB3, an easily deployable RL library in Python, was adopted.

As will be seen, the adoption of SB3 and the iterative refinement of the reward function were critical in overcoming initial challenges and enhancing the performance of the RL agent in *Micropolis*. Through persistent experimentation and adjustments, we were able to develop an agent that could effectively manage the complex dynamics of city development, demonstrating the importance of both robust RL libraries and well-designed reward functions in achieving desired outcomes in complex simulation environments.

StableBaselines3 also offers support for parallelized training using multiple CPU cores or distributed computing frameworks like Ray and MPI via vectorized environments. This enables efficient utilization of computational resources and accelerates the training process, particularly for algorithms like PPO and A2C, which benefit from running multiple instances of the environment in parallel.

Moreover, StableBaselines3 provides a user-friendly interface for defining custom environments and neural network architectures, allowing researchers and practitioners to tailor RL experiments to their specific needs. The library integrates seamlessly with popular deep learning frameworks such as PyTorch and TensorFlow, leveraging their capabilities for efficient neural network training and deployment.

3.3 Algorithms

In this paper, two learning algorithms, A2C and PPO, are employed. The choice to use learning algorithms over planning algorithms for the *Micropolis* environment is influenced by several factors:

1. Untapped Potential: Deep RL algorithms have been explored less than the more traditional planning approaches and thus much is still not understood about how they actually function. It is believed that these algorithms possibly have the capacity to perform far

better than any other Machine Learning (ML)-based approach, but the exact configuration and process of creating the ideal algorithm is still not fully established and thus further research in this domain is warranted.

2. Complexity of the Environment: *Micropolis* presents a very complex and dynamic environment with many interconnected variables and uncertainties. Planning algorithms usually struggle to cope with such a level of complexity, as they require an "explicit" model of the environment's dynamics, which may be challenging to establish accurately for *Micropolis*.

3. Scalability: Learning algorithms, particularly deep reinforcement learning approaches, have been shown to be scalable to large and complex environments. *Micropolis* encompasses a vast virtual world with numerous buildings and configurations, making it well-suited for learning-based approaches that can effectively handle high-dimensional state and action spaces.

4. Adaptability: Learning algorithms excel in environments where the dynamics are not easily known or may change over time. *Micropolis* exhibits complex behaviour and non-linear interactions between various elements, making learning-based approaches that can adapt and evolve their strategies based on observed outcomes quite suitable.

5. Generalization: Learning algorithms have the potential to generalize across different scenarios and environments, whereas planning algorithms often require re-calibration or re-engineering for each new problem instance. Given the emphasis on transferability in the original work by Earle, leveraging a learning algorithm aligns well with the goal of developing a model capable of mastering multiple simulation games beyond just *Micropolis*.

6. Exploration and Exploitation: Learning algorithms naturally incorporate mechanisms for exploration, allowing the agent to discover new strategies and adapt its behavior over time. This ability to balance exploration and exploitation is crucial in *Micropolis*, where optimal strategies may vary depending on the evolving state of the city.

The decision to use the SB3 library and automate the training process without user interference further underscores the commitment to developing robust, independent RL agents. This approach ensures that the agent's learning is driven by its interactions with the environment, fostering the development of strategies that are resilient to the complexities and unpredictability of the *Micropolis* simulation.

3.3.1 A2C

In this study, the synchronous version of the Advantage Actor-Critic (A2C) algorithm is employed as the primary framework for the RL approach to mastering *Micropolis*. A2C is a popular deep RL algorithm that combines elements of both policy-based methods and value-based methods [Mnih et al., 2016].

Consider the typical scenario in RL where an agent interacts with an environment E across discrete time intervals. At each time step t, the agent observes a state s_t and chooses an action a_t from a set $\mathcal{A}$ based on its policy π. In response, the agent transitions to the next state s_{t+1} and receives a reward r_t. This cyclic process repeats until the agent reaches a terminal state (or a maximum number of episodes is reached), after which it restarts. The total return R_t,

calculated as the sum of discounted rewards from time step t onwards with a discount factor γ (where $0 < \gamma \leq 1$), represents the accumulated reward. The primary objective of the agent is to maximize the expected reward from each state s_t [Mnih et al., 2016].

The action-value function $Q^{\pi}(s, a)$ estimates the expected return when taking action a in state s following policy π. The optimal action-value function $Q^{*}(s, a) = max_{\pi} Q^{\pi}(s, a)$ denotes the maximum achievable action value for state s and action a across all policies. Similarly, the value function $V^{\pi}(s)$ under policy π signifies the expected return when starting from state s and following policy π [Mnih et al., 2016].

In value-based model-free RL, the action-value function is approximated using a function approximator like a neural network. Let $Q(s, a; \theta)$ represent an approximate action-value function with parameters θ. θ can then be updated using various RL algorithms. One such algorithm is Q-learning, which directly approximates the optimal action-value function $Q^{*}(s, a)$ with $Q(s, a; \theta)$. In one-step Q-learning, the parameters θ are updated iteratively by minimizing a sequence of loss functions. Each loss function $L_i(\theta_i)$ is defined as the squared difference between the current action value and the one-step return, where s' denotes the subsequent state.

This method adjusts the action value $Q(s, a)$ towards the one-step return. However, a limitation of one-step methods is that they only directly impact the value of the state-action pair s, a associated with the received reward. Other state-action pairs are influenced only indirectly through the updated action value $Q(s, a)$. Consequently, this indirect impact may slow down the learning process as it requires numerous updates to propagate rewards to relevant preceding states and actions [Mnih et al., 2016].

Unlike methods focusing on value, policy-based model-free approaches directly define and update the policy $\pi(a|s; \theta)$ by adjusting its parameters θ through gradient ascent on the expected return $\mathbb{E}[R_t]$. A prominent example of this approach is the REINFORCE family of algorithms introduced by Williams in 1992 [Williams, 1992]. The standard REINFORCE algorithm updates the policy parameters θ based on the gradient of the logarithm of the policy multiplied by the return R_t, providing an unbiased estimate of the gradient of the expected return.

To enhance the accuracy of this gradient estimate while maintaining its unbiased nature, a learned function of the state $b_t(s_t)$, referred to as a baseline, can be subtracted from the return. This adjustment results in a gradient that is the product of the gradient of the logarithm of the policy and the difference between the return and the baseline.

Advantage Actor-Critic combines value-based and policy-based approaches by learning both a value function $V(s_t; \theta_V)$ and a policy $\pi(a_t|s_t; \theta)$ given a state observation s_t [Wang et al., 2017]. The learned policy tells the agent how to "act", while the value function "criticizes" the actions suggested by the policy in order to determine how good they are. Both the policy and value function often share intermediate neural network layers before splitting off into separate networks for each output. Furthermore, rather than using experience replay such as might be done with DQN, Advantage Actor-Critic utilises multiple threads working in parallel to run episodic training, which in combination update one overhead global optimizer [Mnih et al., 2016].

Algorithm 1 Synchronous Advantage Actor-Critic (A2C) [Wang et al., 2018]

// Assume parameter vectors θ and θ_v
Initialize step counter $t \leftarrow 1$
Initialize step counter $E \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$
  $t_{\text{start}} = t$
  Get state s_t
  repeat
    Perform action a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t+1$
  until terminal s_t or $t - t_{\text{start}} == t_{\max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t,\theta_v) & \text{for non-terminal } s_t \end{cases}$ // Bootstrap from last state
  for $i \in \{t-1,\ldots,t_{\text{start}}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ: $d\theta \leftarrow d\theta + \nabla_\theta \log \pi(a_i|s_i;\theta)(R - V(s_i;\theta_v)) + \beta_e \frac{\partial H(\pi(a_i|s_i;\theta))}{\partial \theta}$
    Accumulate gradients wrt θ_v: $d\theta_v \leftarrow d\theta_v + \beta_v(R - V(s_i;\theta_v))\frac{\partial V(s_i;\theta_v)}{\partial \theta_v}$
  Perform update of θ using $d\theta$ and of θ_v using $d\theta_v$
  $E \leftarrow E+1$
until $E > E_{\max}$

3.3.2 PPO

PPO is also employed for the RL approach in mastering *Micropolis*. PPO is a popular and powerful deep RL algorithm that improves upon the stability and performance of previous policy gradient methods, offering a simpler yet effective alternative to algorithms like Trust Region Policy Optimization [Schulman et al., 2017].

As before, an agent interacts with an environment E across discrete time intervals and overall functions very similarly to the actor-critic architecture. However, PPO improves upon such policy gradient methods by incorporating a surrogate objective function that restricts the extent of policy updates at each training step. This approach ensures that updates are neither too large (which could destabilize training) nor too small (which could slow down learning). Specifically, PPO employs a clipped objective function to maintain a balance between exploration and exploitation while ensuring stable policy updates.

The PPO algorithm optimizes the following objective function:

$$L^{CLIP}(\theta) = \arg\max_\theta \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t,a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t,a_t)) \right)$$

where $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio between the new and old policies, ϵ is a hyperparameter that controls the clipping range, and A is an estimate of the advantage function.

The advantage function $A(a_t, s_t)$ is defined as the difference between the action-value function $Q(a_t, s_t)$ and the value function $V(s_t)$. In practice, the advantage function can be estimated

using various methods, such as the Generalized Advantage Estimation (GAE), which provides a balance between bias and variance in the advantage estimates [Kayid, 2019].

PPO employs the following steps during training:

1. **Collect Trajectories**: The agent interacts with the environment and collects trajectories consisting of states, actions, rewards, and other relevant information.
2. **Estimate Advantages**: The advantage function is estimated using the collected trajectories. GAE can be used to compute more accurate and stable advantage estimates.
3. **Compute Policy Updates**: The surrogate objective function $L^{CLIP}(\theta)$ is optimized using gradient ascent, adjusting the policy parameters θ to maximize the expected return.
4. **Update Value Function**: Simultaneously, the value function $V(s_t)$ is updated to minimize the prediction error between the estimated and actual returns.

PPO's clipped objective function plays a crucial role in maintaining the stability and performance of the policy updates. By limiting the magnitude of policy changes, PPO prevents drastic alterations that could lead to sub-optimal policies or instability during training.

The combination of policy optimization and value function updates in PPO allows the agent to adapt effectively to the evolving environment of *Micropolis*, making it a robust approach for handling the game's intricate and multifaceted challenges.

Algorithm 2 Proximal Policy Optimization (PPO) [Kayid, 2019]

Input: initial policy parameters θ_0, initial value function parameters ϕ_0
for $k = 0, 1, 2, \ldots$ **do**
Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
Compute rewards-to-go $\hat{R}_t$.
Compute advantage estimates $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
Update the policy by maximizing the PPO-Clip objective:
$\theta_{k+1} = \arg\max_\theta \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t))\right)$
// typically via stochastic gradient ascent with Adam.
Fit value function by regression on mean-squared error:
$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t\right)^2$
// typically via some gradient descent algorithm.

3.4 Hyperparameter Optimisation (HPO) Methods

Below we introduce the methodology behind which HPO methods we use in this paper, beginning with Random and Grid Search and followed by an overview of the Tree-Structured Parzen (TPE) Estimator approach. While we only utilise three techniques, there are many different approaches to HPO which can aid in optimising hyperparameter configurations. For example, SMAC3 is a popular BO-based method for optimizing complex, structured hyperparameter spaces based on RF-surrogate models [Lindauer et al., 2022], and could likely also be well utilised for the RL tasks in this paper. Other approaches include making use of evolutionary algorithms or expert-prior supported algorithms rather than a probability-based approach, and

all of these techniques have found widespread implementation in a variety of Python libraries and frameworks [6].

3.4.1 Random and Grid Search

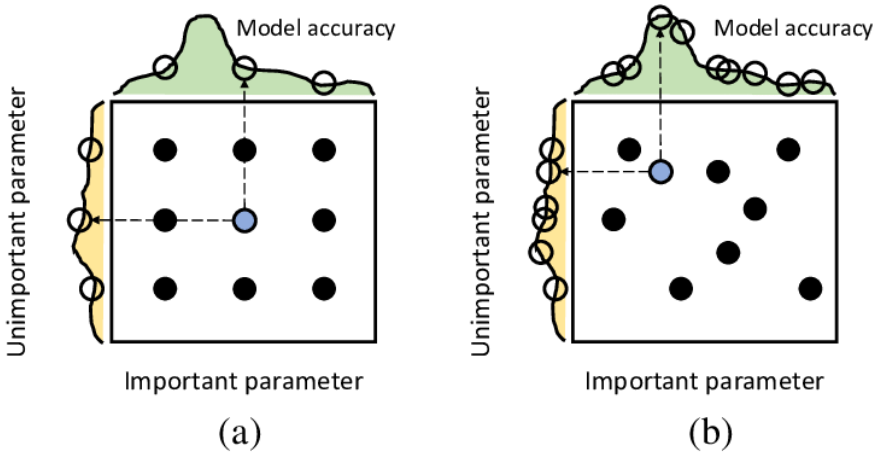


Figure 3: A comparison of (a) grid search and (b) random search HPO approaches [Pilario et al., 2020]

Grid search is one of the simplest HPO methods and used to be one of the most effective for decades due to the lack of technical overhead, the trivialization of implementation and parallelisation and decent reliability [Bergstra and Bengio, 2012]. Just a step above manual HPO, grid search functions by testing configurations in an equally distributed hyperparameter space, with the ranges being specified beforehand by the ML researcher.

However, the paper by [Bergstra and Bengio, 2012] showed that random search trials empirically and theoretically were more efficient the grid search. While still not utilising any form of Bayesian Optimisation (BO), random search finds better models than grid search by effectively searching a larger, less promising configuration space. Furthermore it is still extremely easy to implement, and a ML researcher with some knowledge of good hyperparameter ranges for the specified problem can help improve efficiency even further.

3.4.2 Tree-Structured Parzen Estimator (TPE)

TPE is a more sophisticated approach to HPO first introduced in [Bergstra et al., 2011]. This approach is a variant of BO that functions iteratively by utilising two density functions, one over the good parameter configurations and another over the remaining configurations. It then tries to maximise the ratio of good hyperparameter configurations over the second group by selecting the most suitable parameters for the next step that correspond to the greatest Expected Improvement.

[6] A useful, yet by no means fully comprehensive, overview of different HPO methods can be found at `https://www.ml4aad.org/hpo-overview/hpo-tools/hpo-packages/`

Algorithm 3 Tree-structured Parzen Estimator (TPE) [Watanabe, 2023]

Input: N_{init} (The number of initial configurations), N_s (The number of candidate to consider in the optimization of the acquisition function), Γ (A function to compute the top quantile γ), W (a function to compute weights ${w_n}_{n=0}^{N+1}$), k (A kernel function), B (A function to compute a bandwidth b for k).
Output: Optimized hyperparameters
$\mathcal{D} \leftarrow \emptyset$
for $n = 1, 2, \ldots, N_{init}$ **do** ▷ Initialization
  Randomly pick x_n
  $y_n := f(x_n) + \epsilon_n$ ▷ Evaluate the (expensive) objective function
  $\mathcal{D} \leftarrow \mathcal{D} \cup (x_n, y_n)$
while Budget is left **do**
  Compute $\gamma \leftarrow \Gamma(N)$ with $N := |\mathcal{D}|$ ▷ Splitting algorithm
  Split $\mathcal{D}$ into $\mathcal{D}^{(l)}$ and $D^{(g)}$
  Compute ${w_n}_{n=0}^{N+1} \leftarrow W(\mathcal{D})$ ▷ Weighting algorithm
  Compute $b^{(l)} \leftarrow B(\mathcal{D}^{(l)})$, $b^{(g)} \leftarrow B(\mathcal{D}^{(g)})$ ▷ Bandwidth selection
  Build $p(x|\mathcal{D}^{(l)})$, $p(x|\mathcal{D}^{(g)})$ ▷ Use ${w_n}_{n=0}^{N+1}$ and $b^{(l)}$, $b^{(g)}$
  Sample $S := {x_s}_{s=1}^{N_s} \sim p(x|\mathcal{D}^{(l)})$
  Pick $x_{N+1} := x^* \in \arg\max_{x \in S} r(x|\mathcal{D})$ ▷ The evaluations by the acquisition function
  $y_{N+1} := f(x_{N+1}) + \epsilon_{N+1}$ ▷ Evaluate the (expensive) objective function
  $\mathcal{D} \leftarrow \mathcal{D} \cup (x_{N+1}, y_{N+1})$

3.5 Agent Training

The implementation of our agent training is mostly adopted from Earle's work, with a few minor changes. In our approach, the agent is also provided with a 2D "image" representation of the city board, where each pixel corresponds to a tile and each channel represents different tile states. The same three extra channels of local information, including population density (to detect development in specific zones), traffic density, and power availability for each tile are also provided, alongside certain channels of global information. Slight changes were made to the observation space to normalise inputs, as outlined in the Power Puzzle section later on.

At each step, the agent observes the current state of the gameboard and outputs a chosen action and coordinates to place that action. The actions that can be chosen are dependent on the environment setup, and will be explained more in detail in later sections (4.2.1, 4.3.1). In this action space, the agent specifies a single build to be executed before the next step.

During training, the agent receives rewards at each step based a given reward function. Each step in the environment corresponds to 100 ticks in the game engine, maximizing positive feedback for population-inducing tile configurations, especially in the early stages of training. Furthermore, the agent is provided with virtually unlimited funds and adjustments to tax rates and service budgets have been prohibited. The reasoning behind this is that these adjustments lie outside the scope of the current project, but we advocate that these are definitely potential avenues for future research.

In our research, we experimented with maps ranging from 16×16-tile to 32×32-tile build areas. This size range strikes a balance between providing enough space for meaningful learning and limiting computational resources required. Smaller map sizes would hinder learning as the agent

will frequently overwrite previous builds before meaningful progress is made, and larger map sizes make it more difficult for the agent to easily create connections. Finally, for consistency, a map size of 16×16 was settled on, as it was desirable for the agent to be able to learn with the least amount of computational time necessary.

In terms of infrastructure, we utilised the ALICE Leiden HPC cluster for running our experiments [7]. For all A2C runs we utilised the cpu-short, cpu-medium and cpu-long partitions on this cluster which allowed for up to 12 CPUs to be assigned simultaneously per job. The CPUs utilised were a mixture of Intel Xeon Gold 6126 2.6GHz 12-Core, Intel Xeon Gold 6226R 2.90 GHz 16-Core and AMD EPYC 9534 3.55GHz 64 cores. For the PPO runs we were limited to using 1 GeForce RTX 2080TI GPU per job on the gpu-short, gpu-medium and gpu-long partitions, due to compatibility issues with newer GPUs available on the relevant nodes.

3.6 Network Architecture

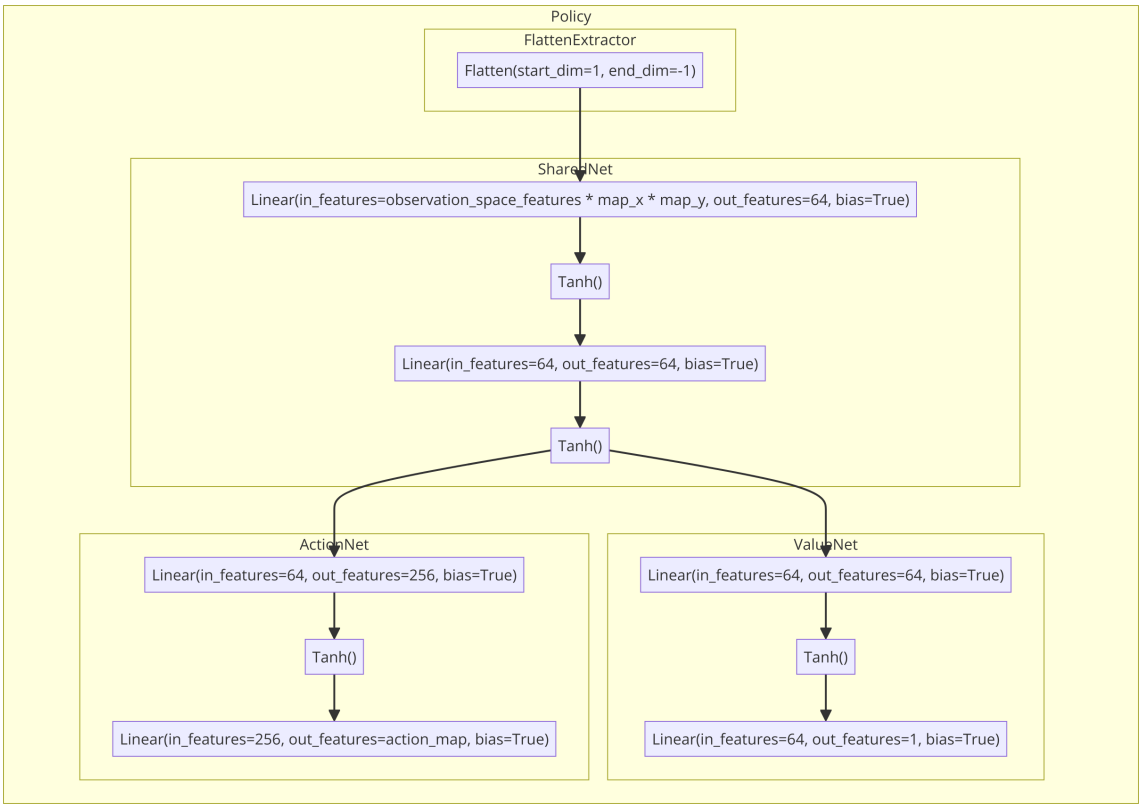


Figure 4: Neural Network Architecture

Quite some experimentation was done with different network architectures to test the learning capacity and complexity of the algorithms. We originally used the default fully connected network provided by the SB3 library, which comprised only of a shared network with 2 layers and 64 units per layer for both PPO and A2C. During our experimentation phase we also incorporated further layers and more neurons per layer into our networks in an attempt to increase the versatility of the model, but ultimately it was theorised that reducing the complexity of the

[7] https://www.universiteitleiden.nl/en/research/research-facilities/alice-leiden-computer-cluster

network would not greatly decrease performance while limiting the amount of computational resources needed.

We also experimented with using different feature extractor architectures. We first tested some of the pre-built extractors provided by SB3, such as the ones provided by the CNNPolicy. However, these extractors only work when the observation space is provided in a very exact format. For example. the CNN extractor would only function with an image observation with each element in the input array taking a value between 0 and 255. Our input, while image-like, was not a raw extract from the environment but rather a curated collection of different map-states. Some effort was put into trying to obtain image observations from the environment but this we deemed too difficult to implement.

We then implemented our own convolutional network feature extractor. This extractor functioned similarly as to what is shown in [Earle, 2020], with one convolutional layer of 5x5 stride followed by a convolutional layer of 3x3 stride, with 32 channels in each layer. While this did function, we did not recognise any major increase in performance over a normal flatten extractor and we were unsure of how this would directly impact our model training, so in the final architecture we decided to omit this modification.

The final architecture used for our networks is relatively simplistic. Rather than using CNNs as done in [Earle, 2020] the observation space is flattened before passing it to the shared network, containing two dense layers with 64 nodes each before splitting off into the action network with one layer of 256 nodes and the value network with one layer of 64 nodes. After each of the hidden dense layers a Tanh() activation function is used. We utilise the "MlpPolicy" class from the SB3 library for automatic integration of non-image inputs into the SB3 algorithm framework. This architecture, while simple, has the capacity to obtain good working results within both the Power Puzzle mini-game and the main *Micropolis* environment, leading us to adopt it for our final experiments.

4 Experimental Analysis

4.1 Environment Vectorization

Environment vectorization was a challenging yet intriguing aspect of the agent we worked on implementing. Vectorized environments are a built-in feature of SB3, allowing multiple instances of the environment to run in parallel. This parallelism enhances training efficiency by increasing the diversity of experiences the agent encounters within a given time frame, thus speeding up the learning process.

The primary reasons for using vectorized environments in training the A2C and PPO algorithms for *Micropolis* include increased experience diversity, improved sample efficiency, reduced training time, and enhanced stability of learning. Parallel environments provide more training samples per unit of time, enhancing sample efficiency. This means the agent can learn from more experiences in each training iteration, leading to faster convergence and better performance. Vectorization significantly reduces the wall-clock time required for training. Since multiple environments run in parallel, the agent can collect experiences from several environments simultaneously, accelerating the overall training process.

The increased volume of data from vectorized environments can also lead to more stable

learning. The agent's updates are based on a larger and more varied set of experiences, which can help in stabilizing the learning process and avoiding overfitting to specific patterns in a single environment. By employing environment vectorization, we aimed to leverage these benefits so as to maximise our agent's learning performance in *Micropolis*. This approach ensured that more effective strategies for managing the city's complex dynamics could be developed by the agent, leading to superior performance and competency in the game.

4.2 Power Puzzle

4.2.1 Setup

Micropolis, with its complex nature, presents a unique challenge for designing agents that can effectively navigate the large action space available to them. To properly assess the capability of our agents to actually learn and perform according to a given reward function, a mini-game named "Power Puzzle", presented in Earle's work, was adapted.

In this mini-game, at the start of each episode of gameplay, between 1 and 5 residential zones are randomly placed on the map, followed by the placement of a power plant. The action space available to the agent is restricted to one channel: it may solely build power wires and is unable to overwrite existing structures on the map. This simplifies the action space greatly and lets the agent focus on spatial reasoning and building skills.

The objective of the mini-game is for the agent to discover the most optimal path to make connections between any configuration of power plants and zones. It is noteworthy that the design of this mini-game lends itself to a cellular automaton solution, as mentioned by [Earle, 2020]. However, for the purposes of this study, we refrain from utilizing such methods and rather focus on assessing the capabilities of our networks to learn and adapt strategies for this task.

As stated above, this mini-game serves as a fundamental testbed for evaluating the adaptability and learning capacity of agents within the *Micropolis* environment. By systematically varying the complexity of the game board by changing the number of residential zones, we aim to gain insights into the strategies developed by agents and how they navigate spatially dispersed objectives. Furthermore, the Power Puzzle mini-game is used to prove that the environment works as intended and that a given agent actually has the capacity to learn from interacting with the environment.

We focus on utilising the A2C algorithm for the Power Puzzle mini-game as it only serves to function as a testing prototype for our environment and we are not interested in running PPO on it, as we would likely obtain minimally improved results with far more computation time required.

4.2.2 Development Process

Unfortunately, getting the mini-game to work proved to be far more challenging than anticipated. While the original code from previous work was available online on a GitHub repository, it did not function seamlessly with any of the SB3 algorithms. Uncertainty lingered regarding whether the issue lay with the network, the environment, or another aspect of the agent itself. Initially, efforts were directed towards modifying the reward function, following a similar line of thought to that of which was being followed in the main environment concurrently. Earle's

paper mentioned rewarding the agent at each step for population on the game board and also for each zone connected to the power grid. While both variables were successfully implemented in the reward function, the agent failed to exhibit improvement beyond what random runs achieved. Reward graphs indicated that the agent would quickly converge and then cease to learn further.

Struggling with the reward function, attention shifted to testing different iterations of the network. Adjustments were made to the complexity by varying the number of nodes and layers, but no insights were gained regarding whether the network itself was problematic. Additionally, experimenting with adding a CNN as a feature extractor to enhance spatial reasoning did not yield performance improvements [Earle, 2020]. Realizing the modular nature of SB3, it became apparent that a completely different environment could be easily integrated into the algorithm to test both the network and the algorithm itself. Consequently, the agent successfully learned to perform in both "CartPole" and the ATARI "Alien" environments, confirming that there were no issues with the network or the algorithm. Instead, the issue was identified to be with the environment.

The next step involved simplifying the environment as much as possible. Since the Power Puzzle mini-game exists within the standard gym city environment, resolving issues with it would significantly aid in achieving our ultimate goal. To streamline the process, all unnecessary variables, functions, and calculations were removed, aiming to simplify the environment to its core components. This approach facilitated a clearer overview of the environment's operations and the specific changes made, aiding in the troubleshooting process.

In our next experiments, we focused on testing both the observation and action spaces of the environment. Initially, efforts were directed towards simplifying the observation space by removing extraneous information, retaining only the initial map. A new approach was then implemented, where a 2D numpy array was created to represent the integer values of buildings located at specific coordinates on the map. However, these attempts were unsuccessful. Upon further examination of the SB3 documentation, it became apparent that the observation space should be bounded and normalized within a specified range during environment initialization. While the code found in previous work specifies this range to be between -1.0 and 1.0, testing revealed that certain parts of the observation, particularly the population and traffic density maps, returned values higher than 1.0. To address this issue, a maximum cap of density at 100 was set, and these values were then normalized by dividing them by the maximum cap. Additionally, it was discovered that global scalar layer computations of zoning demand sometimes returned values higher than 1.0, which was identified as a bug and subsequently fixed. While these adjustments did not immediately resolve environment issues, they ensured a thorough understanding of how the observation space was constructed and that it complied with SB3 guidelines.

Following the adjustment of the observation space, attention shifted to the action space. The original implementation defined the action space simply as a discrete space of $num.tools * mapwidth * mapheight$. While this sufficed for the Power Puzzle environment, it could pose significant challenges in the actual *Micropolis* environment. Upon consulting the OpenAI Gym documentation, it was realized that the action space could be re-implemented as a Multidiscrete environment. This approach would provide separate heads for the tool, x coordinate, and y coordinate chosen, thereby reducing the complexity of the action space and potentially improving agent performance.

After adjusting both the action and observation space to satisfaction, attention returned to refining the reward function. Eventually, a moment of enlightenment occurred. Reflecting on the success of the "CartPole" environment, where the reward function was simply a measure of episode length, a realization dawned upon the author of this paper. The Power Puzzle problem involves connecting sources to sinks to generate population, and the faster this connection occurs, the better the agent's performance. Thus, shorter episode lengths correspond to better agent performance, if a terminal condition is implemented in which the episode would end if all sinks are connected to a source. This insight led to the solution: stipulating a sparse reward function that would reward the agent at the end of each episode based on how quickly it completed the task. This formulation ensured that the problem was presented in a format understandable to the agent.

4.2.3 HPO

While some initial hyperparameter configurations were available from previous work for the main *Micropolis* environment, we still deemed it necessary to perform our own HPO to ensure the code was running correctly and the agent had the capacity to learn. Furthermore, Earle's paper does not mention any concrete hyperparameter values or ranges for the Power Puzzle mini-game, leaving these to be determined independently. Initial experiments with grid search were unsuccessful, partially due to the issues with the environment mentioned above. We then worked on a manual implementation of random search, where a small collection of different hyperparameter configurations were tested. Random search is an easy-to-use HPO method that can still return good results, especially if the researcher performing the HPO has a rough idea of what kind of hyperparameter values would be good to use [Bergstra and Bengio, 2012]. All results are averaged over 4 vectorized environments with seeds of 1, 2, 3, and 4 to increase the reliability of the results and remove elements of randomness, run for about 2.5 million frames.

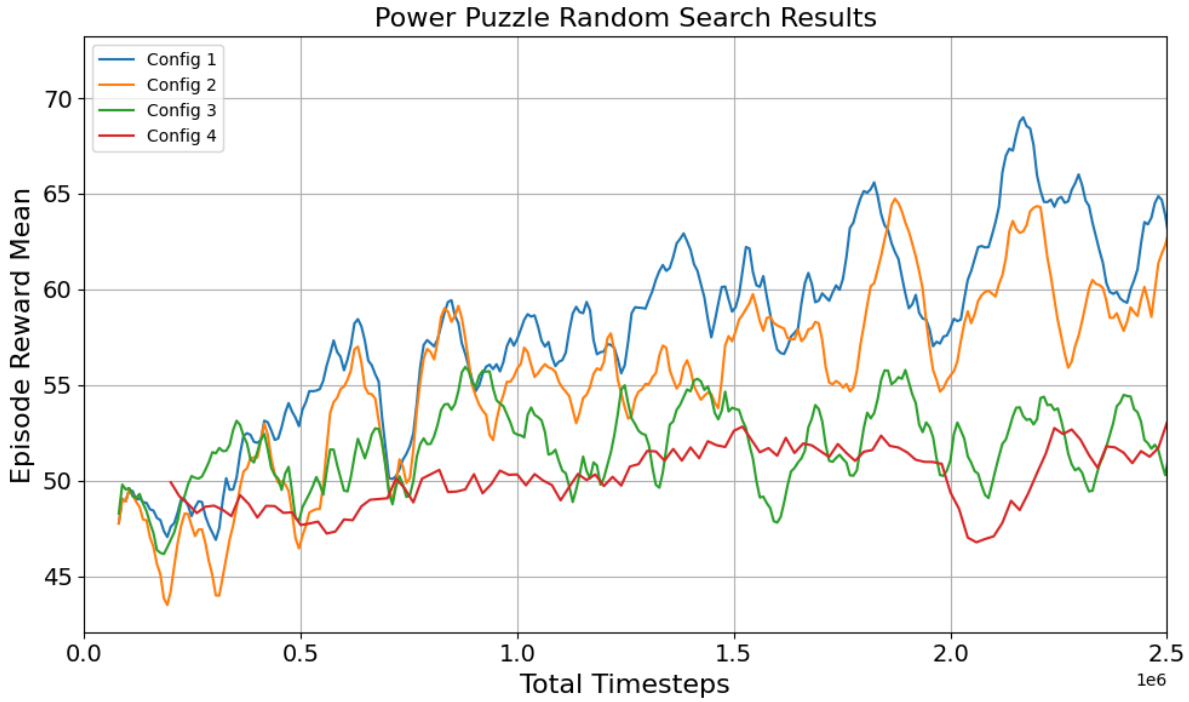


Figure 5: Random search results for four different hyperparameter configurations of the Power Puzzle agent using the A2C algorithm

The x axis indicates the number of timesteps which directly corresponds to the number of frames the agent observes during training, with the exponential number in the lower right indicating the magnitude of measurement. The Y axis indicates mean reward attained per episode at the current timestep. The legend indicates which colour corresponds to which hyperparameter/algorithm configuration. As can be seen, configuration 1 and 2 outperforms the other runs, with configuration 1 performing slightly better. The hyperparameter values for configuration 1 can be found in table 1.

After completing and implementing the random search runs we also implemented a BO algorithm called TPE via the hyperopt Python library [8]. We ran the HPO algorithm over 20 trials for 5 million frames each for about 18 hours. Shown below are a range of 5 runs sampled from these trials, including the best performing run of configuration 1.

[8] `https://github.com/hyperopt`

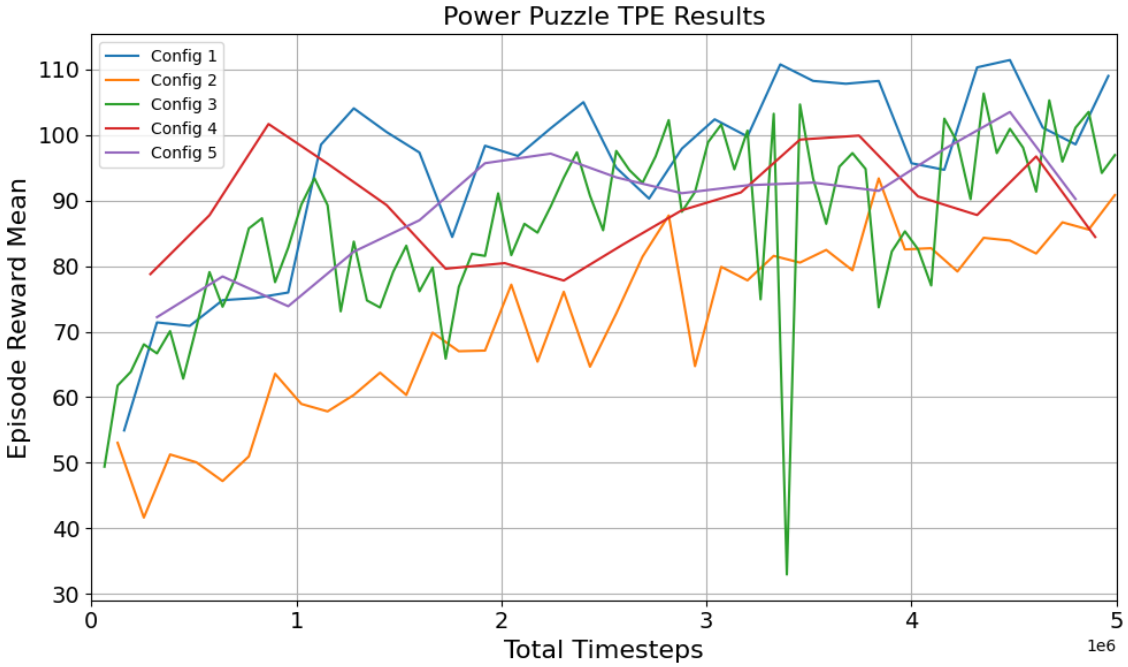


Figure 6: TPE results for the Power Puzzle agent using the A2C algorithm

As can be seen, configuration 1 does perform overall slightly better than other runs and displays greater stability, with no major losses in performance as indicated by i.e. configuration 3. Furthermore we can see the configurations tested here perform better than the random search runs from figure 5, with configurations 1 and 4 reaching values over 100 within the first 2 million timesteps. The values used for configuration 1 can be seen below:

Hyperparameter	**RS**	**TPE**
Number of Steps	20	45
Gamma	0.96	0.92
Value Loss Coefficient	0.5	1.0
Entropy Coefficient	0.01	0.0036
Max Grad Norm	0.5	1.0
Learning Rate	1e-4	2e-4
Lambda	0.98	0.91

Table 1: Comparison of hyperparameter values for best-performing configurations from HPO

4.2.4 Results

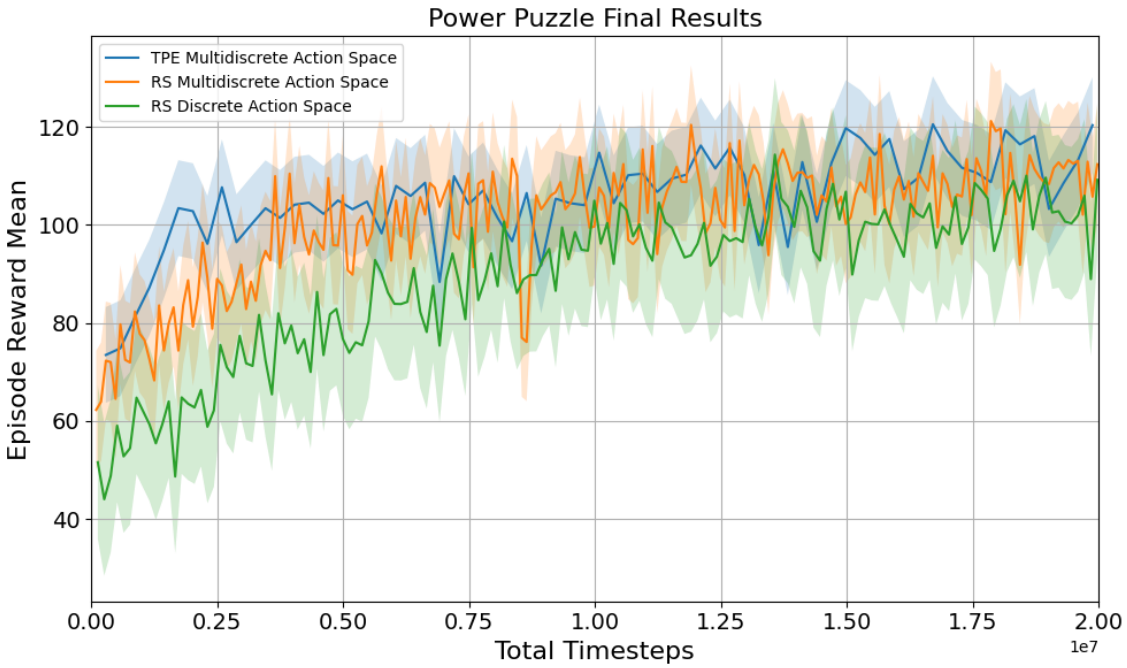
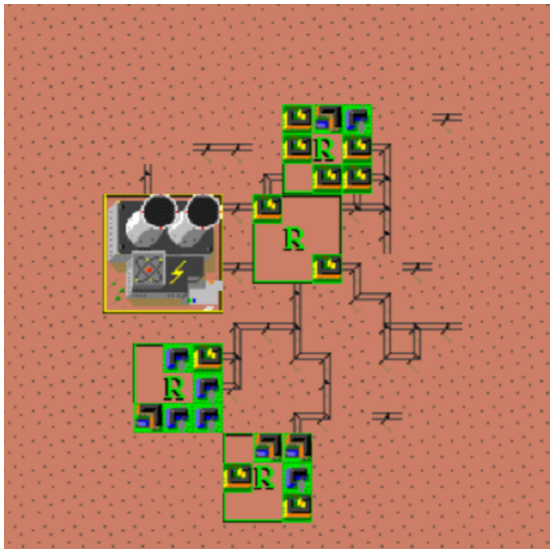
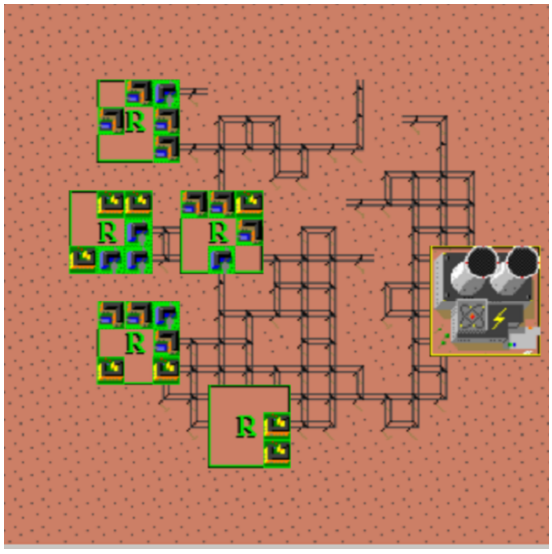


Figure 7: Final results for the Power Puzzle agent using the A2C algorithm. The lighter shaded areas of the same colour display the distributions of the runs, calculated as the data point value ± the dataset's standard deviation.

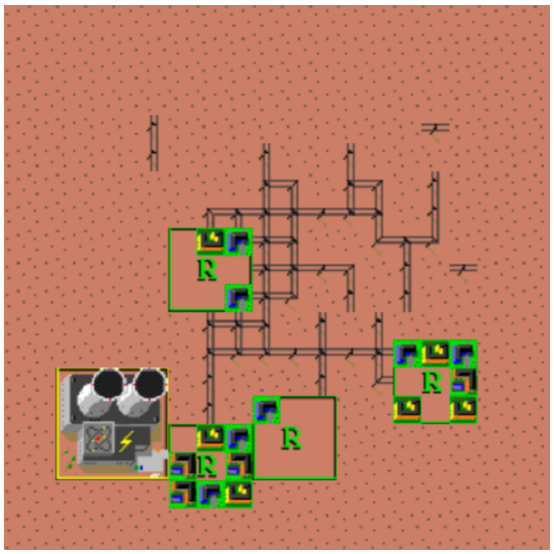
All runs were completed using 64 simultaneous environments, with the final results for each run being averaged over all environments. Figure 7 shows that the multidiscrete action space does perform better early on in the run and generally overall. However, it is not possible to conclude this definitively as the distribution zones often heavily overlap between all three configurations. Closer to the end of training at 2e7 frames all runs seem to plateau to around similar values, indicating a possible local optimum. The TPE run only seems to perform minimally better than the Multidiscrete random search run, which may be attributed to a variety of reasons, such as not having long enough training times, the average episode length/reward not being a suitable enough indicator for loss, or the difference in hyperparameter values not really influencing the actual functionality of the agent by a lot.



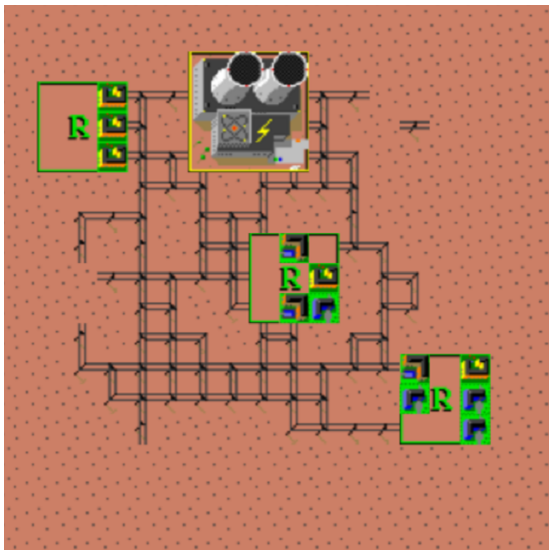
(a) A run with 4 residential zones



(b) A run with 5 residential zones



(c) Another run with 4 residential zones



(d) A run with 3 residential zones

Figure 8: Power Puzzle mini-game results with various map setups

The strategy adopted by our agent in figure 8 can be analysed to a certain extent. As can be seen in fig 8a when the residential zones are placed close to the power plant the agent quickly and efficiently finds a solution, placing minimal random wires to connect all zones. However, it is interesting to note that the lower left zone is not connected directly to the plant (which would only take 1 wire), but rather that it is connected via the center residential zone. This could indicate that the agent has a preference for connecting from one zone to another once it has established an initial connection, rather than starting a new power line from the plant.

Figures 8b and 8d indicate that the agent has more difficulty establishing connections when zones are far away from the power plants, leading to much more building overall. The agent also does not usually build singular paths but rather arrays of connected networks. It is possible that this kind of approach is more reliable overall depending on what kind of setup the agent is faced with, but especially in cases like these it is not the most efficient minimum spanning tree that the agent could take.

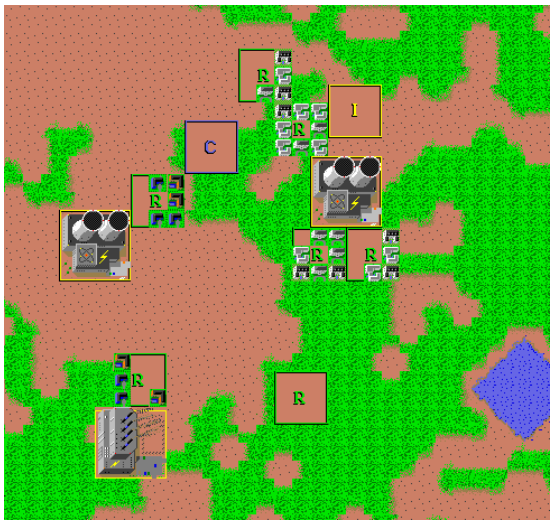
4.3 StableBaselines3 Experiments

4.3.1 Setup

From previous work we retain the *Micropolis* Python environment, originally written by Don Hopkins. Some adjustments have been made to the Action and Observation spaces, with the Action space being converted from Discrete to Multidiscrete and the Observation space being checked for bugs and fixed. With SB3, attempts were made to use both the A2C and PPO algorithms, supplying custom hyperparameter configurations based on testing. The primary objective was to program the agent to build a small village with an action space consisting of zoning, power, wires and roads.

We provide the agent with a limited toolset consisting of just zoning, power plants, roads and wires, as we are just interested in the ability of the agent to build coherent neighbourhoods. The other utilities and services available would have little use on the map size we have chosen for the agent. The agent is rewarded at each step by a function that calculates the population from each zone type: residential, commercial and industrial, alongside receiving an extra bonus for further zone variety. Furthermore, the agent is also rewarded for building roads that are adjacent to a zone or a building.

4.3.2 Development Process



(a) Agent learns that placing zoning adjacent to other zones and more importantly to power plants allows for initial population establishment.

(b) Inter-connectivity between zones leads to higher development and population size, but truly structured neighbourhoods do not exist yet and the agent is unable to establish a consistent road network

Figure 9: Early results of agent training.

The simplest reward function used in previous code measured total population, obtained directly from the engine and comprising all three types of zoning, not just residential. A more complex version of this reward function, and the most-used implementation in both our and Earle's code, took into account zoning variety on top of pure population. This enhanced version incentivised the agent not only to increase population but also to strategically diversify by developing all three types of zoning.

Initially, the agent showed no capacity to learn until it was discovered that the environment's configuration apparently necessitated the reward function to be based on positive or negative changes rather than absolute values. Adjusting the reward to measure the change in population finally yielded some initial results. However, eventually we shifted back to using absolute values as the change in results from the initial adjustment likely stemmed from setup inaccuracies, which were rectified as competency in using the code grew.

The process of refining the reward function was iterative and crucial for improving the agent's learning performance. Initially, focusing solely on population change helped the agent begin to understand the impact of its actions. However, it soon became apparent that this approach was limited. It became clear that more nuanced reward signals were necessary to guide the agent towards building a functional and sustainable city. This realization prompted experimentation with additional reward metrics, such as penalizing the agent for unconnected zones or rewarding it for creating infrastructure that supported population growth, like power plants and roads.

One significant breakthrough occurred when rewarding the agent for building zoning areas. Zoning is essential for population growth, and this change resulted in the first successful outcomes. Further improvements were observed when rewarding only powered zones, ensuring that population could grow there. However, the agent still struggled to build roads to connect these zones, highlighting a critical oversight.

To address this challenge, a more sophisticated reward mechanism was incorporated to evaluate the placement of roads. By obtaining a 2D array of the game map at each step, indicating the type of building in each tile, the agent was rewarded if a road was adjacent to a zone or another building. This encouraged the construction of roads around zones, although they often remained isolated without forming larger networks. Further tweaks were made based on this insight, resulting in an agent capable of constructing slightly more coherent and connected road systems.

Progress in this area stalled until the Power Puzzle environment was resolved, as explained earlier in this paper (see section 4.2.2). With a newly configured environment that was confidently working correctly, motivation was renewed to address this issue.

One solution implemented was to compel the agent to build adjacent roads whenever it constructed a zone or power plant, which initially facilitated connections between different areas. However, this approach was deemed unsatisfactory, prompting further exploration of reward engineering and other possible approaches to encourage the agent to construct roads. Satisfactory results were finally achieved with the implementation of a grid layout to help agent with building roads. With a reward function based on population, zoning variety and road adjacency our agents were able to generate relatively stable neighbourhoods with traffic flowing between different zones and buildings. The results are shown below in section 4.3.4.

4.3.3 HPO

As mentioned in section 4.2.3, while some initial hyperparameter configurations were available, we decided to conduct independent HPO to obtain optimal configurations, but also to ensure the code was running correctly and the agent had the capacity to learn. Grid search was chosen as the method, as it was relatively easy to implement and still yielded decent results. All outcomes were averaged over 4 vectorized environments with seeds of 1, 2, 3, and 4 to enhance result reliability and mitigate randomness. The grid search HPO was done during an

earlier phase of experimentation using a slightly simpler reward function, but we believe the environment and network architecture are similar enough to our final setup to warrant being displayed here and used for further experimentation.

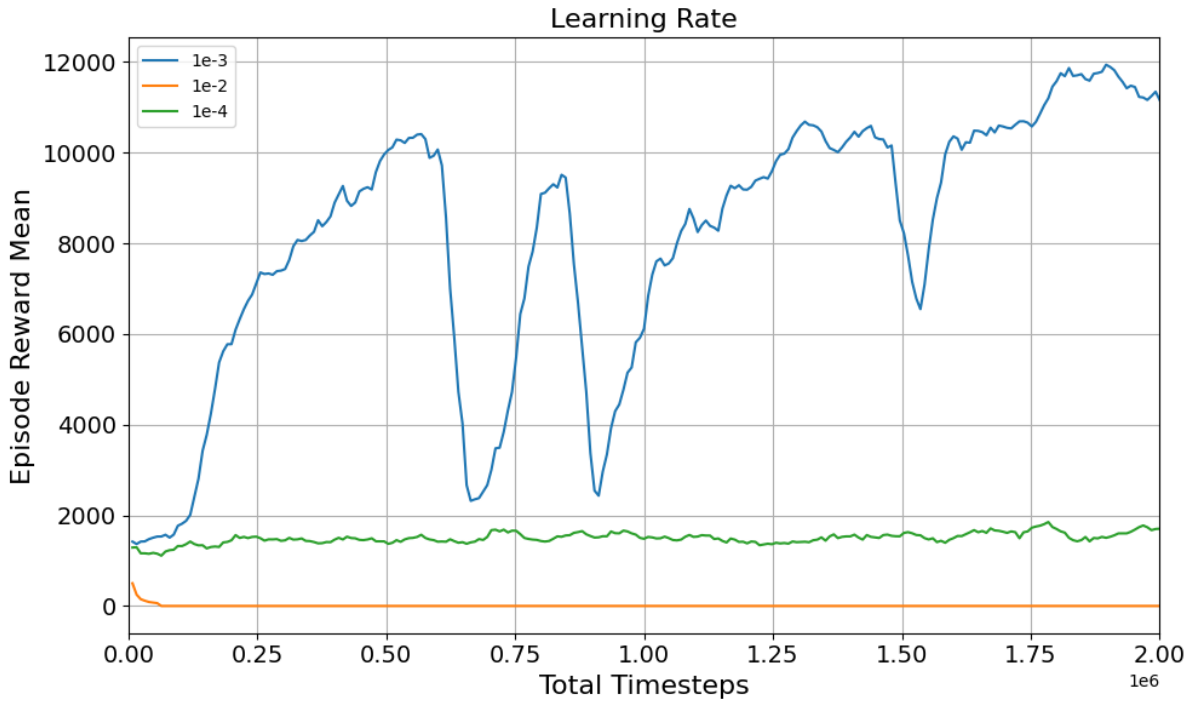


Figure 10: A2C Grid Search Learning Rate Results.

Figure 10 provides an example plot of the ones seen in the following pages, specifically here shown are the grid search results for Learning Rate variations with the A2C algorithm. Explanations for the plot are similar as when first explained in the Power Puzzle HPO section (4.2.3). Based on the results obtained in figures 10 and 11, we propose hyperparameter values for the A2C algorithm which can be found in table 2 at the end of this section.

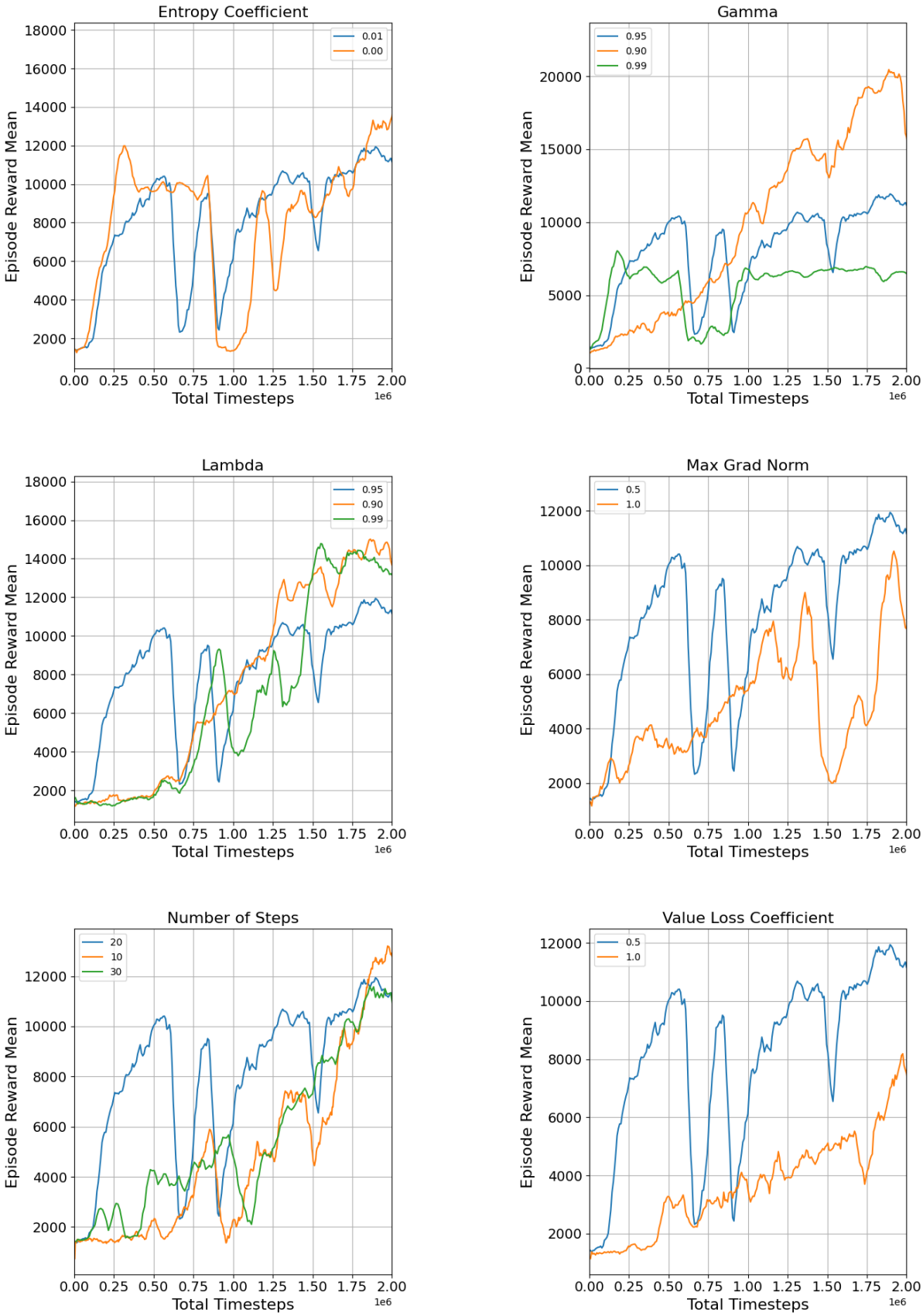


Figure 11: A2C Grid Search Results.

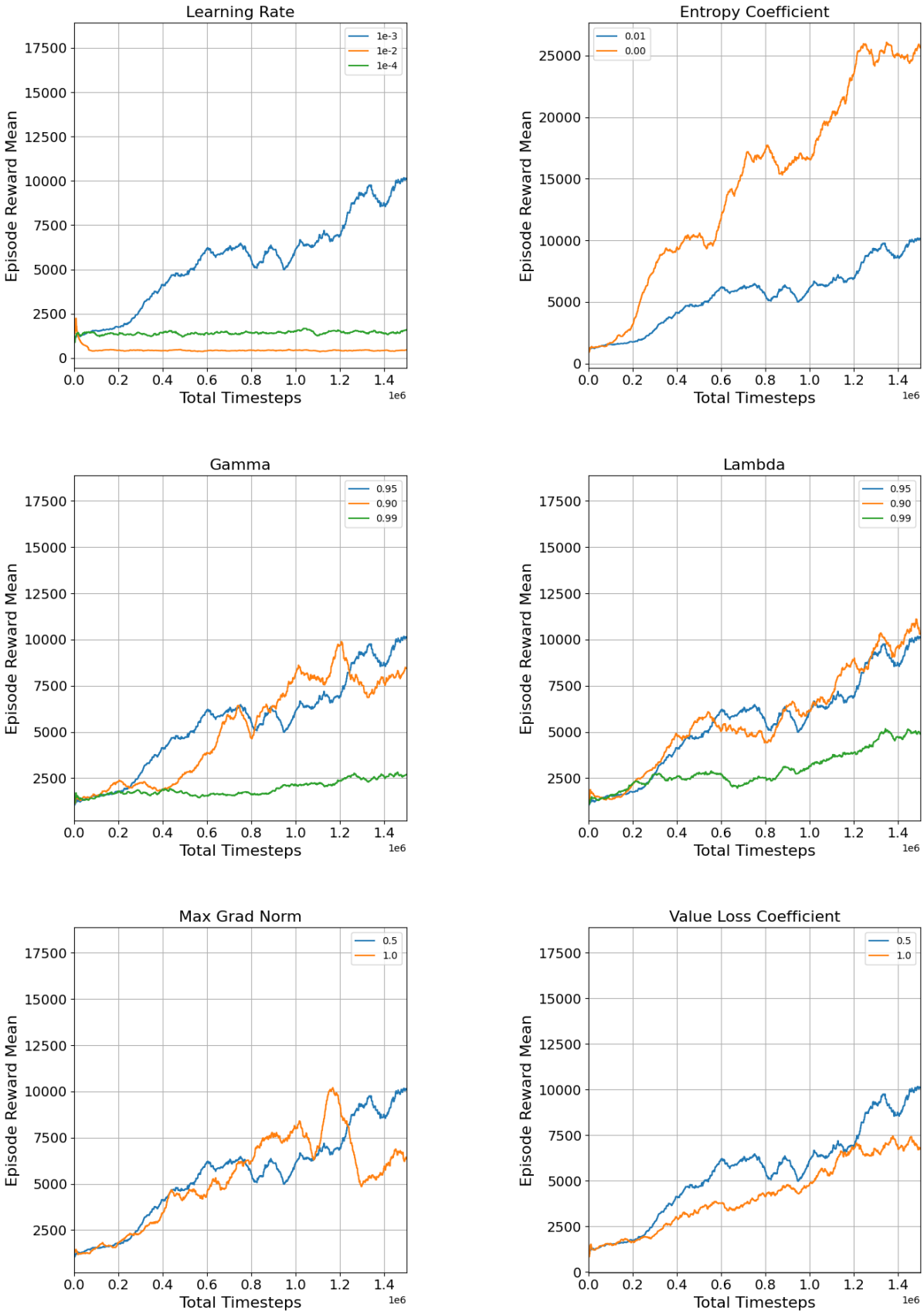


Figure 12: PPO Grid Search Results.

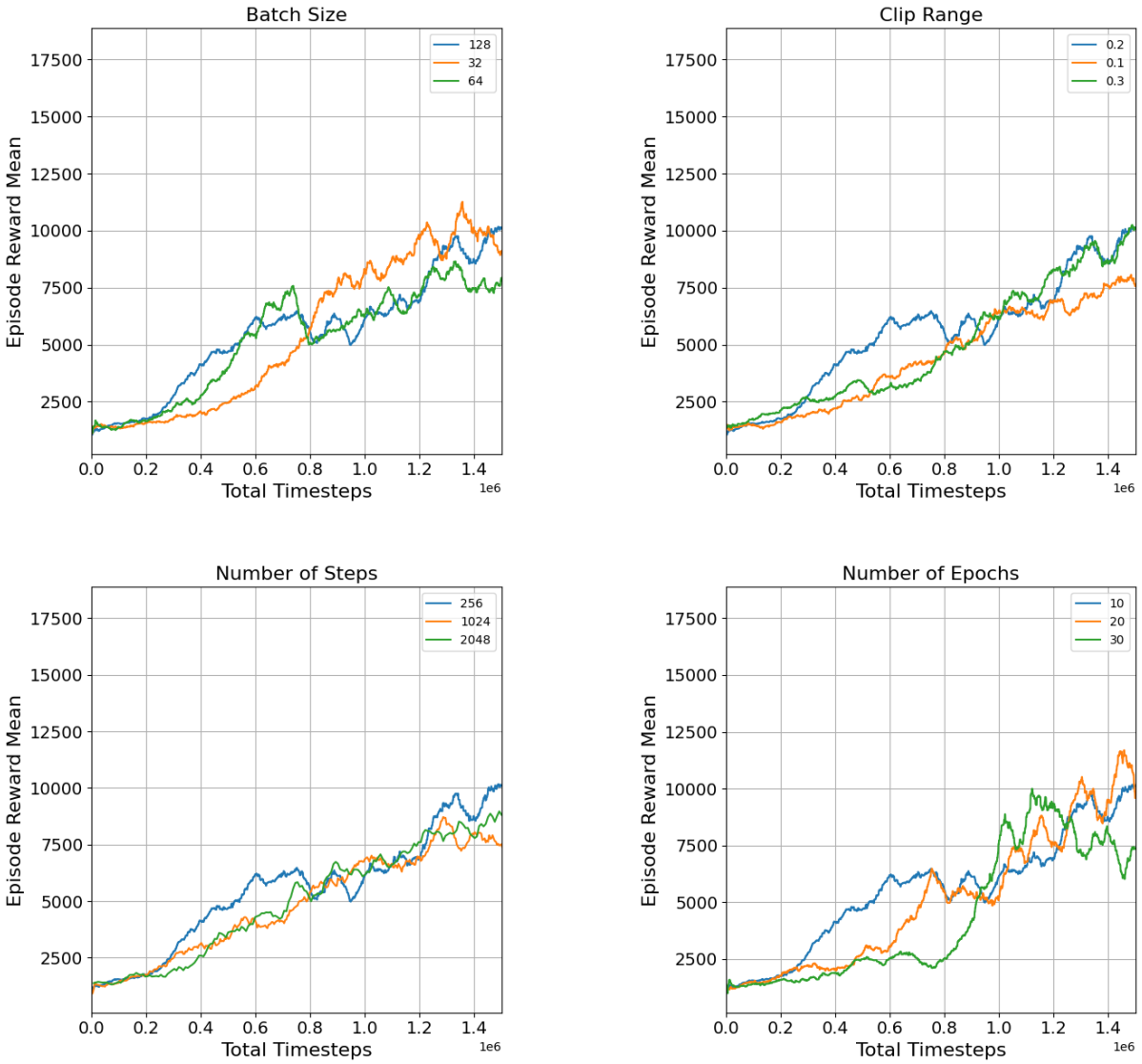
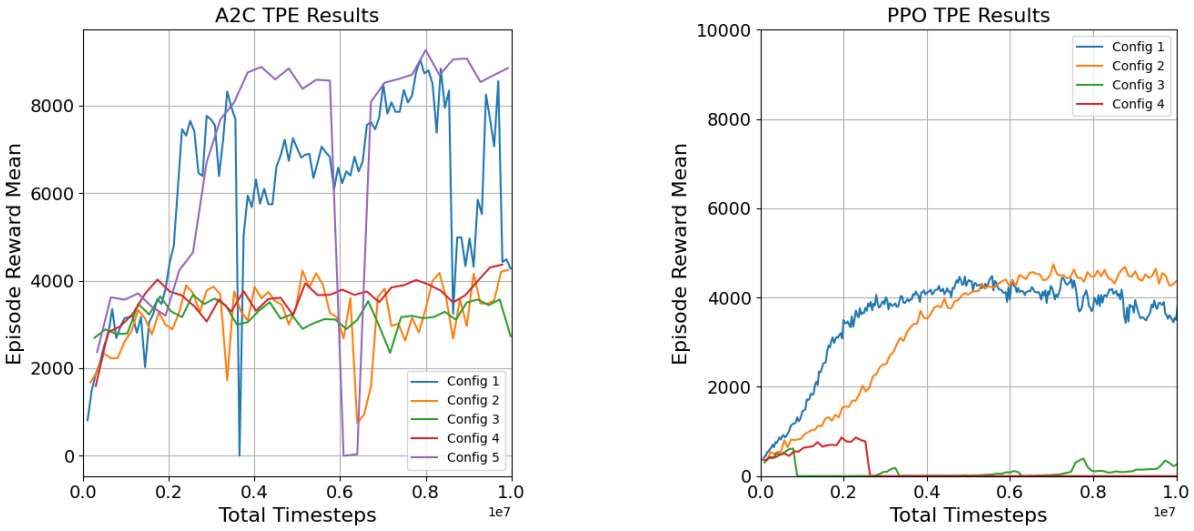


Figure 13: PPO Grid Search Results cont.

Based on the results obtain in figures 12 and 13, we propose hyperparameter values which can also be found in table 2.

Furthermore, we also utilised TPE to automatically optimise our hyperparameters for both algorithms, the results are shown here:



(a) A2C TPE results

(b) PPO TPE Results

Figure 14: TPE results for the A2C and PPO algorithms.

For A2C we ran 20 trials and sampled 5 here including the highest performing one, as was done for the Power Puzzle mini-game. Due to computational resource limitations we were unable to run the full TPE HPO on the PPO algorithm, thus only 4 runs can be displayed here. This also may point to why figure 14 indicates such surprising results. We expected PPO to outperform A2C but were unable to reach even close to the same performance. As stated, we believe this is due to computational issues, one run distributed over 64 vectorized environments for A2C would take less than 6 hours while for PPO it could take between 20-30 hours. One characteristic we can attest to is the stability of the runs, as seen for configurations 1 and 3 in figure 14b. There is far less variation and instability as compared to runs in figure 14a, which we attribute to the clipped objective function and increased stability of the PPO algorithm.

For A2C Configuration 5 was the best performing configuration measured by our metric. However, we do note the instability for this configuration as seen in 14a at around 6 million timesteps.

Hyperparameter	**A2C GS**	**A2C TPE**	**PPO GS**	**PPO TPE**
Number of Steps	20/10/30	50	256	1280
Gamma	0.90	0.95	0.90/0.95	0.97
Value Loss Coefficient	0.5	0.5	0.5	1.0
Entropy Coefficient	0.01	0.003	0	0.0009
Max Grad Norm	0.5	1.0	0.5	1.0
Learning Rate	1e-3	6e-4	1e-3	5e-4
Lambda	0.90	0.99	0.90/0.95	0.91
Batch Size	-	-	128	96
Clip Range	-	-	0.2/0.3	0.2
Number of Epochs	-	-	20	20

Table 2: Comparison of A2C and PPO hyperparameter configurations for best-performing configurations from HPO. Multiple values in a cell indicate that any of them may perform similarly well.

While final TPE HPO results are somewhat disappointing, especially for PPO, we do believe it is still good practice to include such approaches and thus integrate them below.

4.3.4 Results

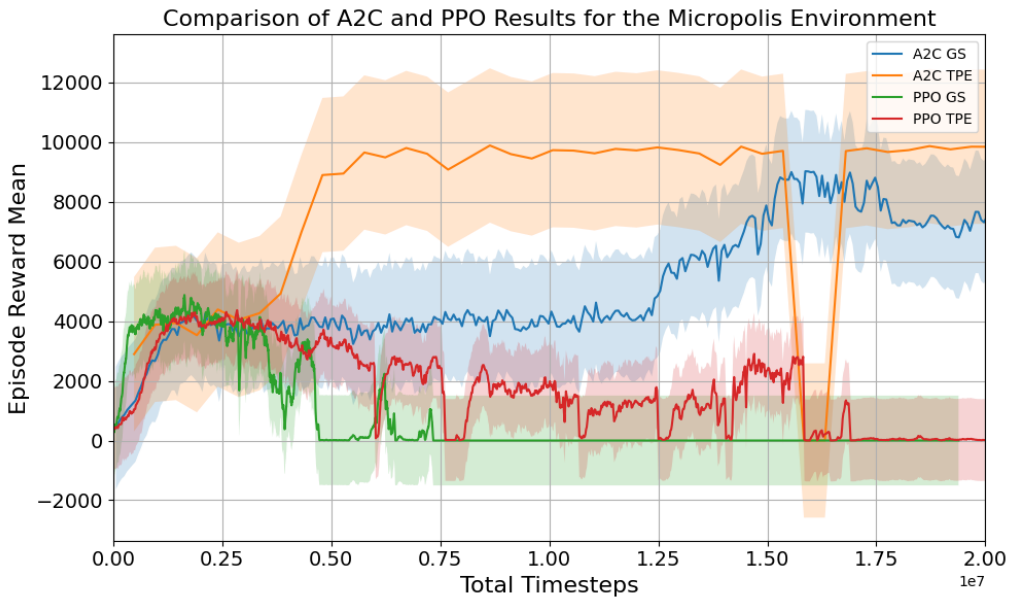
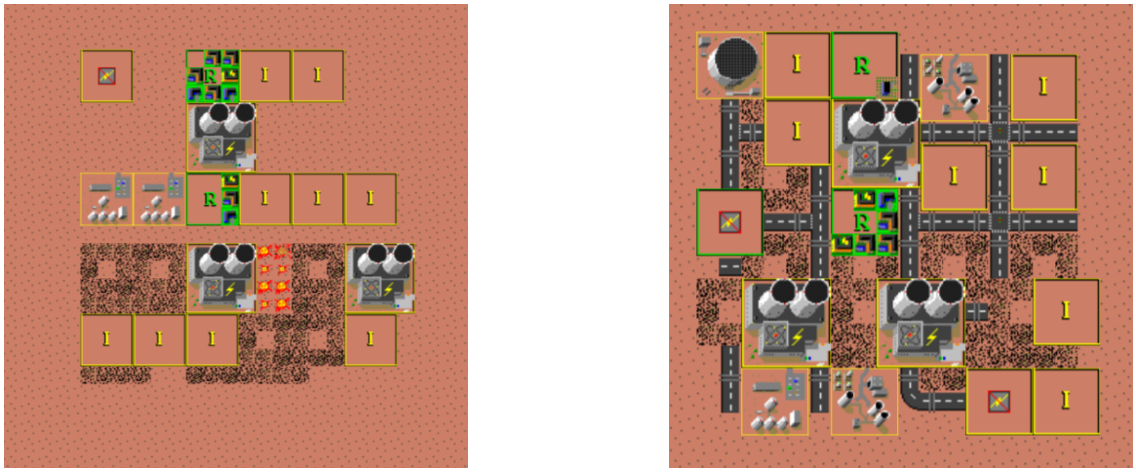


Figure 15: Final results for *Micropolis* Environment over 20M timesteps. The lighter shaded areas of the same colour display the distributions of the runs, calculated as the data point value ± the dataset's standard deviation.

For the graphs in this section we include distributions as done in section 4.2.4. Figure 15 indicates initial final training results of our two agents. While the A2C agent performs as we expect it to, with stable training, the performance of both PPO runs drops off over time. This was unexpected, but could also not be predicted during HPO as that was only run for less than 2 million timesteps for PPO. For the A2C TPE run we see a similar drop in performance as in figure 14a. Overall the A2C algorithms seem more flexible and capable of learning than their PPO counterparts, which is surprising but can be attributed to the computational resources available and the variety of situations exposed to the agent via vectorized environment running. Due to memory issues we were able to run the PPO algorithm with only 12 simultaneous environments, while we were able to run A2C with up to 96.



(a) No grid layout

(b) Grid Layout

Figure 16: A2C City-building results on a 16x16 Build area with pure population + zoning based reward.

Figure 16 shows results with a reward function dependent on just total population and zoning variety. Without a grid layout, as seen in figure 16a, the agent fully focuses on building zones in a row-like order, with plants placed in between to provide power to the connected zones. There does seem to be adequate spacing for roads in the center row and around certain zones, but the agent makes no attempt to try and fill these spaces with such infrastructure.

Figure 16b shows the results of this run when provided with a grid layout. Little of the original grid is preserved during agent building, but the placement of power plants still allows for power to be fed to most of the zones. The agent does not seem to try and respect any kind of pre-existing infrastructure provided by the environment, and also places too many power plants which can be possibly attributed to it being trained without a road grid.

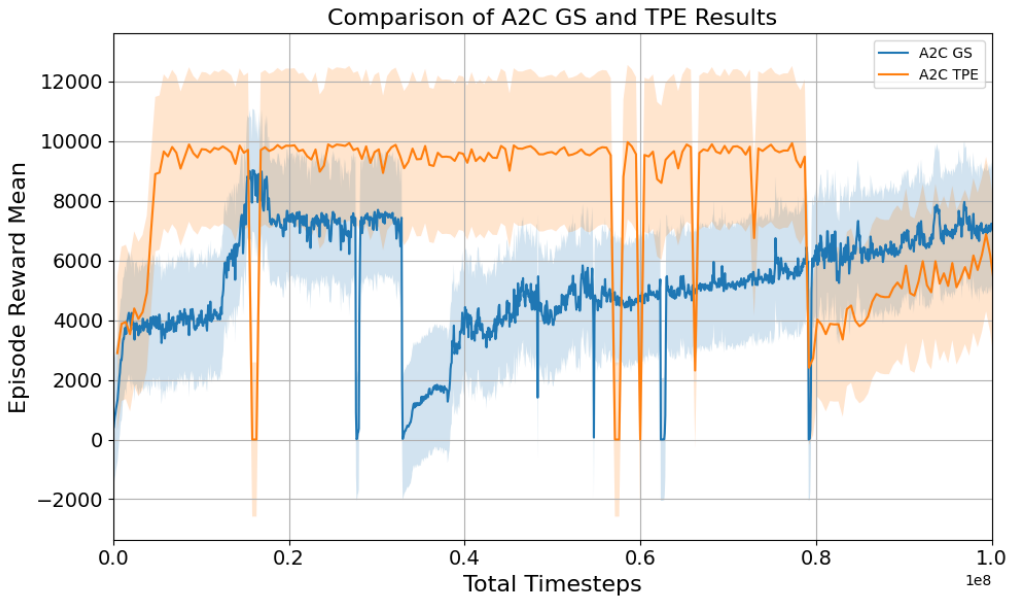


Figure 17: Final results for the A2C Agents over 100M timesteps.The lighter shaded areas of the same colour display the distributions of the runs, calculated as the data point value ± the dataset's standard deviation.

Figure 17 shows the training of our A2C agents over a longer timeframe. Interestingly enough, after about 80M timesteps the TPE algorithm experiences a sharp drop in performance and the GS algorithm overtakes it's counterpart in terms of average reward. While neither of these agents provide adequate visualisations at any point in their training process, they do still indicate that the network is learning and exploring in different directions during training, trying to overcome local optima.

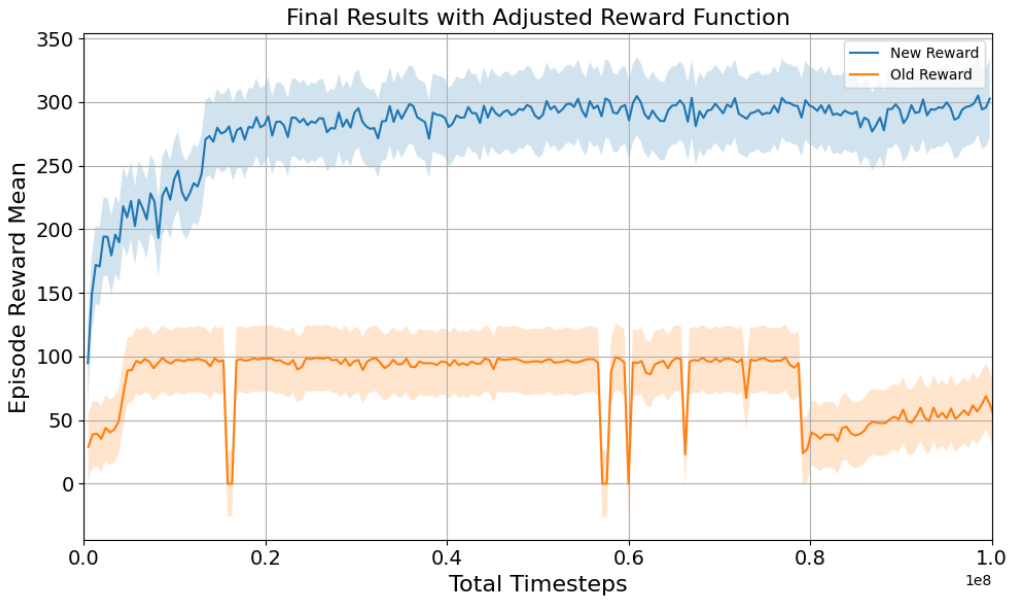


Figure 18: Comparison between final old and new reward functions over 100M timesteps. The old reward function has been normalised in the same way as the new one for comparability. The lighter shaded areas of the same colour display the distributions of the runs, calculated as the data point value ± the dataset's standard deviation.

To obtain the results as seen in figure 19 we had to make one final adjustment to the reward function with which we had experimented with before, which was providing a road adjacency score to the agent. We also normalised the population-based reward function we had been utilising until now by dividing it by 100. At each timestep, the agent would update a virtual map of the gameboard in it's memory and select all of the roads present. For each road present, the agent would be provided with a score for +1 if the given road was adjacent to at least one zone or building, and with a score of -1 if the road was not adjacent to any of these. This would encourage the agent to build roads next to zones, or at least respect existing road grid layouts if provided with them. The old reward function was normalised so that the new road adjacency score addition would be significant enough to have an impact on the behaviour of the agent. The effects of these changes are displayed in displayed in figure 18, with the old reward function coloured in orange and the new one coloured in blue. We can see that the new reward function is more stable than the old one, without any major oscillations. Furthermore overall reward is higher, but this was also to be expected with the addition of a new component.

(a) Example 1 (b) Example 2

Figure 19: A2C City-building results on a 16x16 Build area when supplied with a grid layout and reward function including road adjacency score.

The results shown in figure 19 indicate the extent of development of our algorithms, with a reward based on population, zoning and road adjacency. While we were unable to influence either the A2C or PPO algorithms to build functional road grids throughout the engineering of the reward function itself, when supplying the agents with a layout of the road grid at the start of each episode the agents would adapt to this and build power plants and zoning while performing minimal changes to the infrastructure itself. This does not always work perfectly, as we can see in 19a where in the lower right quadrant the agent has built a single, unconnected commercial zone and left a lot of empty space where a far better configuration of zoning is possible.

5 Discussion and Conclusion

This research focused on designing and improving RL agents from the SB3 library for the complex and dynamic environment of *Micropolis*, a city-building simulation. The iterative process of developing these agents highlights the importance of exploratory analysis embedded in a robust RL framework as to achieve effective agent performance in such dynamic environments. The results obtained in this study continue to underscore the potential of RL to handle the multifaceted challenges presented by dynamic simulation environments such as *Micropolis*. Furthermore, the implementation of these RL algorithms via the SB3 library promotes reproducibility in an effort to help standardise the research area of RL so that future researchers no longer need to completely build agents and environments from scratch.

The selection of the A2C and PPO algorithms was due to their robustness and versatility in handling complex action and state spaces. A2C provided a strong baseline due to its combination of policy gradient and value-based methods, facilitating stable and efficient learning. Furthermore, the lightweight nature of the algorithm allowed for efficient distributed execution on the CPU sections of ALICE. PPO, while it was supposed to further enhance performance through better handling of policy updates and exploration-exploitation trade-offs, could not perform as well as A2C due to computational resource limitations.

Hyperparameter optimisation was an integral part of refining both agents. Key hyperparameters

such as learning rate, gamma and lambda alongside PPO-specific hyperparameters were optimised via by random search, grid search and TPE. This optimization process was crucial in balancing the agent's ability to explore the environment effectively while learning stable policies that generalize well across different scenarios.

5.1 Process of Designing and Refining the Agent

The initial experimentation phase involved setting up the *Micropolis* environment and updating our fork of the GitHub repository of Earle to function with the ALICE environment. Despite having access to prior research and Python code, recreating a satisfactory baseline proved extremely challenging. This difficulty first brought forth the necessity of a stable and well-documented RL framework, eventually leading to the adoption of SB3. This library provided a modular and flexible foundation with well-tested policies, allowing for more streamlined experimentation and a suitable solution to the initial challenges faced.

5.1.1 Observation and Action Space Design

One of the foundational steps in developing the RL agent was defining the observation and action spaces. The observation space was adapted from Earle's original code and captured essential aspects such as population density, traffic flow and power availability [Earle, 2020]. This detailed representation ensured that the agent had access to all relevant information required to make informed decisions. Some changes had to be made to the observation space to normalise the values so that they would be compatible with the SB3 framework. While some other variations of the observation space were tested, we found that the original construction worked best.

The formulation of the action space was initially defined as in the original code but was later redefined to reduce computational complexity. The re-implementation of the action space as a multidiscrete space allowed the agent to provide separate predictions for the action, x coordinate and y coordinate. By enabling the agent to focus on specific subsets of actions (e.g. by either varying only the structure or the location being built), the multidiscrete action space facilitated more precise control and learning.

5.1.2 Power Puzzle Environment

The Power Puzzle experiment, while also initially fraught with challenges, ultimately provided valuable insights into the adaptability and learning capacity of agents within the environment. Despite encountering obstacles in aligning the original code with SB3 algorithms, the iterative process of troubleshooting and refinement led to significant advancements in both the environment's functionality and agent performance.

Initially, difficulties arose in modifying the reward function to incentivize effective gameplay. Despite incorporating population and zone connectivity as reward parameters, the agent failed to exhibit substantial improvement. Attention then shifted towards testing various iterations of the neural network architecture, but no significant breakthroughs were achieved.

Efforts then shifted towards simplifying the environment itself. Streamlining the Power Puzzle mini-game (and thus by extension the overall *Micropolis* environment) by removing unnecessary variables and functions enabled a clearer understanding of its functionality, facilitating

targeted troubleshooting. Subsequent adjustments to the observation space, particularly in normalizing values and addressing computation errors, ensured compliance with SB3 guidelines. Adjustments to the action space, as mentioned above, were also performed during this period of experimentation.

A pivotal breakthrough occurred with the realization that the reward function could be simplified to align more closely with the agent's task: connecting sources to sinks efficiently. By structuring the reward function to incentivize shorter episode lengths, the agent was presented with a clearer objective, leading to optimised performance.

5.1.3 Reward Engineering

Developing an effective reward function for *Micropolis* was perhaps the most challenging aspect of the project. The initial reward function was too simplistic, focusing solely on population growth. This approach led to suboptimal behaviors, such as the agent building zones and power plants without connecting them with roads. This highlighted the necessity of a more nuanced reward structure that could guide the agent towards not only immediate goals but also long-term city planning objectives.

The iterative refinement of the reward function involved incorporating criteria such as rewarding only powered zones and evaluating the placement of roads. This iterative process significantly improved the agent's performance. For instance, rewarding the agent for building roads adjacent to zones or other buildings encouraged the development of more structured and connected urban layouts. This process underscores the importance of reward engineering in RL, as it is shown that the reward function directly influences the learning trajectory of the agent. Furthermore, penalizing the agent for negative actions, such as creating unpowered zones or isolated roads, helped in developing more sustainable city planning strategies.

As stated, subsequent iterations introduced rewards for zoning activities and ensuring zones were powered, which significantly improved general agent performance. However, the agent's inability to construct coherent road networks presented a new challenge. The solution involved a sophisticated reward mechanism that evaluated the placement of roads, encouraging the agent to build interconnected road systems around zones. This approach led to more coherent city development, although challenges remained in creating structured neighborhoods. This was finally solved by providing a road grid before each episode to the agent, which then was no longer due to reward engineering but rather through modifications made to the environment.

5.1.4 Challenges Faced

The design and refinement of our RL agents for *Micropolis* faced several challenges:

- Complexity of the Environment: *Micropolis*, with its multifaceted dynamics and interdependent systems, presented a highly complex environment for the RL agents. Our agents had to manage various aspects of city planning simultaneously, such as zoning, road development, and power distribution. This complexity required a comprehensive observation space and sophisticated reward structures.
- Computational Efficiency: Training the RL agents in such a complex environment demanded significant computational resources. The use of SB3 and its support for parallelized training was crucial in addressing this challenge. By leveraging multiple CPU cores and

running parallel environments, the training process was significantly accelerated, allowing the agents to experience a wider variety of scenarios within a shorter time-frame. Furthermore, running experiments on the ALICE server allowed for far more computational resources to be allocated to training.

- Reward Function Design: Designing a reward function that effectively balanced short-term and long-term objectives was particularly challenging. Initial iterations focusing solely on population growth led to unintended behaviors. Through refinement and extensive testing, a more balanced reward structure was developed that guided the agents toward more suitable city planning strategies.

5.2 Future Work and Research Directions

While this paper has made significant progress in standardising and reducing the complexity of utilising RL algorithms with the *Micropolis* environment, several interesting avenues for future research remain. Further exploration of even more advanced reward shaping techniques, such as conditional or hierarchical rewards, could enhance agent performance and fix challenges that remain unsolved in this study. Further investigating the application of transfer learning to enable agents to use knowledge from similar tasks and environments, which is more in the vein of what was done in [Earle, 2020], could improve learning efficiency and generalisation. Increasing the difficulty and complexity of the *Micropolis* environment by allowing the agent to adjust taxation and service rates could bring us another step closer to real-world systems and increase the versatility of our approach.

We also recognise the low performance of our PPO algorithm in this paper, and future work with better infrastructure and more powerful GPUs available could provide further research in this direction to obtain better results with GPU-based RL approaches. We also suggest that applying the methodologies we have developed to real-world urban planning and tasks could validate the practical utility of our research and highlight potentially unforeseen areas for improvement. Finally, combining our approach with other AI techniques, such as evolutionary algorithms or neural architecture search, could ultimately lead to more robust and adept solutions.

5.2.1 Applicability to Other Tasks

The methodologies developed and insights gained from this study potentially have broader implications for other complex and dynamic environments beyond *Micropolis*. A core aspect of this research, utilising a well-established RL framework such as SB3 can significantly streamline the overall development process, providing stability and reproducibility with no significant drawbacks. Developing effective reward functions via an iterative approach is critical to any complex environment, and taking into account agent behaviour during this process can lead to more meaningful and effective improvement. Careful tuning of hyperparameters is essential for achieving optimal performance, and automated HPO techniques such as TPE can perform this process much more effectively than most humans. The ability to adapt to changing conditions and objectives is crucial in dynamic RL, and the usage of techniques such as parallelised training to reduce wall-clock times and expose the agents to a broader range of scenarios, can easily be applied to other dynamic and complex tasks if they can be parallelised.

5.2.2 Guidelines for Designing Similar Agents

Based on the experiences gathered in this research, several ideas can be proposed for designing and developing RL agents in similar environments. These include choosing a good RL framework, ensuring the environment is accurately modelled, and iteratively refining reward structures based on observed agent behaviour. Furthermore, choosing RL algorithms that are suited for the task complexity and action space, and considering alternative approaches are both important steps. We also found that performing systematic hyperparameter tuning, manual and automatic, can help improve agent performance greatly. Regularly evaluating agent performance using informative metrics and utilising visualization tools to monitor training progress is also core to ensuring effective development. We have compiled the lessons learned below into a step-by-step guideline, that we hope can be generalized across different environments and problems, to aid future researchers in their endeavours when working with RL agents performing similar tasks.

1. **Choose a Robust RL Framework** - Selecting the correct RL framework is fundamental. Robust frameworks such as SB3 provide a variety of modular tools that are essential for standardized RL training. These frameworks offer stable training, well-implemented algorithms, utilities for monitoring and evaluation, and support for custom environment integration.
2. **Accurately Model and Integrate the Environment** - Ensure that the environment is functioning properly within the framework. This can range from running simple test problems to check if a given agent can interact with the environment to implementing 'mini-games', such as the Power Puzzle task, to test higher level learning without requiring the complexity or resources of the main task.
3. **Construct Efficient Action and Observation spaces** - If working with discrete action spaces, reformulating them as Multidiscrete or Dictionary spaces can help with programmer legibility and reduce complexity. Ensure that the action space covers all possible decisions the agent might want to make, but avoid adding unnecessary complexity that could hinder learning. The observation space should provide all pertinent information for decision-making without overwhelming the agent with redundant metrics. Test different iterations of both spaces to see what works best for you, and ensure that the formulation of the spaces are in accordance with the framework being used.
4. **Develop Initial Reward Function** - Start with a basic reward function that aligns with the overall objectives of the task. The initial function can be extremely simplistic, but it should still be able to influence the agent during it's learning process enough to provide meaningful growth.
5. **Hyperparameter Tuning** - Begin tuning the hyperparameters of your algorithms to facilitate better learning and balance of the exploration/exploitation trade-off. Manual methods such as random and grid search provide decent initial results, but automated techniques such as TPE should be used in later stages if possible. Key hyperparameters for learning algorithms such as A2C and PPO include learning rate, discount factor, and the coefficients for value and entropy loss.
6. **Leverage Parallelized Training** - Accelerate the learning process by leveraging parallelized training techniques such as Open MPI. Utilize multiple environments and distributed

computing frameworks to enhance scalability, especially if access to high-performance computing systems is possible. This approach not only often drastically speeds up the training process but also helps in better generalizing the agent's policies across different scenarios.

7. **Continuous Monitoring and Evaluation** - RL tasks such as these often require long training times over millions of episodes, with there often being no clear indication of improvement for long stretches. Therefore, regularly evaluate agent performance using diverse informative metrics. Implement visualization tools to monitor learning progress, diagnose issues and observe trained behaviour. This step involves analyzing learning curves, policy behaviors, and other performance indicators to ensure the agent is improving and adapting as expected.
8. **Iterative Refinement and Experimentation** - Adopt an iterative approach, making step-by-step improvements based on feedback and performance analysis. Be prepared to experiment with a variety of different architectures, reward functions, and optimization techniques. Refine the reward function based on agent feedback and performance. Analyse agent behaviour to identify areas where the reward function might be functioning sub-optimally. Positive rewards should be assigned for achieving desired outcomes, while penalties should discourage undesirable actions. Adjust the rewards to align more with long-term objectives and encourage increasingly complex behaviour.
9. **Incorporate Advanced Techniques** - Explore the possible integration of more advanced techniques such as multi-objective reward functions and transfer learning. These methods can further enhance agent performance and versatility. Additionally, consider the integration of multi-agent systems where different RL agents can play with or against each other in a single environment, adding further flexibility and versatility to the approach.
10. **Real-World Application and Validation** - If possible, apply developed strategies to real-world tasks to validate their practical utility. This step involves coordinating with relevant policy-makers and domain experts to transfer learned RL policies to real-world scenarios and assessing their effectiveness. This highlights potential areas for improvements and ensures that the research done is applicable beyond just a simulated environment.

5.3 Conclusion

The development of RL agents for intricate and highly complex environments like *Micropolis* involved an extensive design and experimentation process. Even with access to prior work and code, many adjustments to the reward function, environment configuration and overall ML pipeline were still required to take place before satisfactory results could be obtained. Utilising SB3 as the RL framework in this study is a crucial aspect of our research that helps standardise approaches to RL and enables future researchers to more easily build upon and extend previous studies.

Ultimately, we found that we were unable to have our agent automatically generate satisfactory city structures through pure reward engineering. Even though we attempted a variety of different functions that measured different metrics, we were unable to find a balance, especially between road-building and zone-building. However, when we provided our agent with a template of a

grid during training and inference it learnt to respect these boundaries and started creating more lifelike neighbourhoods.

The A2C and PPO algorithms were both powerful RL approaches that we applied to our environment. A2C was our main focus as it was more successful than it's counterpart, training in far less wall-clock time due its optimisation for running on CPUs and the computational resources available at ALICE Leiden. PPO should perform better and with far more stability if correctly configured and enough resources are allocated to it, making it a viable approach for future researchers.

The lessons learned from this research can be applied beyond the specific case of *Micropolis* and may find use in a wide range of RL problems. By emphasizing the critical importance of leveraging modular RL libraries, performing independent HPO using advanced methods, and adopting parallelised training techniques, researchers and RL practitioners can develop more effective solutions for a variety of systems modelling complex, real-world scenarios. Ultimately, this work contributes to the rapidly growing body of knowledge in dynamic RL, and highlights the potential of this subdomain of ML as a powerful approach for addressing complex challenges in various research areas.

References

[Auer, 2002] Auer, P. (2002). Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3:397–422.

[Beckenbach et al., 2022] Beckenbach, L., Osinenko, P., and Streif, S. (2022). A stabilizing reinforcement learning approach for sampled systems with partially unknown models.

[Bellinger et al., 2024] Bellinger, C., Crowley, M., and Tamblyn, I. (2024). Dynamic observation policies in observation cost-sensitive reinforcement learning.

[Bergstra et al., 2011] Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc.

[Bergstra and Bengio, 2012] Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305.

[Earle, 2020] Earle, S. (2020). Using fractal neural networks to play simcity 1 and conway's game of life at variable scales.

[Feng et al., 2012] Feng, X., Butler-Purry, K., and Zourntos, T. (2012). Multi-agent system-based real-time load management for all-electric ship power systems in dc zone level. *Power Systems, IEEE Transactions on*, 27:1719–1728.

[Friedman, 1999] Friedman, T. (1999). The semiotics of simcity. *First Monday*, 4.

[Green et al., 2021] Green, M. C., Yen, V., Earle, S., Rajesh, D., Edwards, M., and Soros, L. B. (2021). Exploring open-ended gameplay features with micro rollercoaster tycoon.

[Haarnoja et al., 2018] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.

[Hafner et al., 2024] Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. (2024). Mastering diverse domains through world models.

[Kaiser et al., 2024] Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Mohiuddin, A., Sepassi, R., Tucker, G., and Michalewski, H. (2024). Model-based reinforcement learning for atari.

[Kayid, 2019] Kayid, A. (2019). Distributed deep reinforcement learning for large scale robotic simulations.

[Li et al., 2022] Li, J., Wu, X., and Fan, J. (2022). Speed planning for connected and automated vehicles in urban scenarios using deep reinforcement learning. In *2022 IEEE Vehicle Power and Propulsion Conference (VPPC)*, pages 1–6.

[Li et al., 2023] Li, Z., Yang, Z., and Wang, M. (2023). Reinforcement learning with human feedback: Learning dynamic choices via pessimism.

[Lindauer et al., 2022] Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhopf, T., Sass, R., and Hutter, F. (2022). Smac3: A versatile bayesian optimization package for hyperparameter optimization.

[Lu et al., 2023] Lu, X., Fan, F. X., and Wang, T. (2023). Action and trajectory planning for urban autonomous driving with hierarchical reinforcement learning.

[Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning.

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[Mukherjee and Vu, 2021] Mukherjee, S. and Vu, T. L. (2021). On distributed model-free reinforcement learning control with stability guarantee. *IEEE Control Systems Letters*, 5(5):1615–1620.

[O'Donoghue et al., 2017] O'Donoghue, B., Osband, I., Munos, R., and Mnih, V. (2017). The uncertainty bellman equation and exploration.

[OpenAI et al., 2019a] OpenAI, :, Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., d. O. Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. (2019a). Dota 2 with large scale deep reinforcement learning.

[OpenAI et al., 2019b] OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., and Zhang, L. (2019b). Solving rubik's cube with a robot hand.

[Osband et al., 2016] Osband, I., Blundell, C., Pritzel, A., and Roy, B. V. (2016). Deep exploration via bootstrapped dqn.

[Osinenko et al., 2022] Osinenko, P., Yaremenko, G., and Osokin, I. (2022). A note on stabilizing reinforcement learning.

[Pilario et al., 2020] Pilario, K. E., Cao, Y., and Shafiee, M. (2020). A kernel design approach to improve kernel subspace identification. *IEEE Transactions on Industrial Electronics*, PP:1–1.

[Raffin et al., 2021] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: reliable reinforcement learning implementations. *J. Mach. Learn. Res.*, 22(1).

[Schrittwieser et al., 2020] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., and Silver, D. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609.

[Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

[Shah et al., 2020] Shah, D., Xie, Q., and Xu, Z. (2020). Stable reinforcement learning with unbounded state space.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489.

[Silver et al., 2017] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm.

[Taylor and Stone, 2009] Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.*, 10:1633–1685.

[Tessler et al., 2016] Tessler, C., Givony, S., Zahavy, T., Mankowitz, D. J., and Mannor, S. (2016). A deep hierarchical approach to lifelong learning in minecraft.

[Thompson, 1933] Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294.

[Torrado et al., 2018] Torrado, R. R., Bontrager, P., Togelius, J., Liu, J., and Perez-Liebana, D. (2018). Deep reinforcement learning for general video game ai.

[Vázquez-Canteli and Nagy, 2019] Vázquez-Canteli, J. and Nagy, Z. (2019). Reinforcement learning for demand response: A review of algorithms and modeling techniques. *Applied Energy*, 235:1072–89.

[Wang et al., 2023] Wang, H., Liang, D., and Xi, Y. (2023). Urban path planning based on improved model-based reinforcement learning algorithm. In *Proceedings of the 4th*

International Conference on Advanced Information Science and System, AISS '22, New York, NY, USA. Association for Computing Machinery.

[Wang et al., 2018] Wang, J., Kurth-Nelson, Z., Kumaran, D., Tirumala, D., Soyer, H., Leibo, J., Hassabis, D., and Botvinick, M. (2018). Prefrontal cortex as a meta-reinforcement learning system. *Nature Neuroscience*, 21.

[Wang et al., 2017] Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. (2017). Learning to reinforcement learn.

[Watanabe, 2023] Watanabe, S. (2023). Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance.

[Watkins, 1989] Watkins, C. (1989). Learning from delayed rewards.

[Wei et al., 2018] Wei, H., Zheng, G., Yao, H., and Li, Z. (2018). Intellilight: A reinforcement learning approach for intelligent traffic light control. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 2496–2505, New York, NY, USA. Association for Computing Machinery.

[Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256.

[Wright and Joffe, 1989] Wright, W. and Joffe, B. (1989). Simcity: thematic mapping+city management simulation=an entertaining, interactive gaming tool. In *GIS/LIS '89 Proceedings. Annual Conference*, volume 2, pages 591–600, Orlando, FL, USA. American Congress on Surveying and Mapping; American Society for Photogrammetry and Remote Sensing.

[Zhang et al., 2023] Zhang, Z., He, F., and Oki, E. (2023). Dynamic VNF Scheduling: A Deep Reinforcement Learning Approach. *IEICE Transactions on Communications*, 106(7):557–570.

Disclaimer

This thesis was conducted with the assistance of ChatGPT in accordance with the regulations stipulated by Leiden University. Specifically, ChatGPT was utilized as a study aid during the initial stages of research, assisting in brainstorming ideas and searching for information. Additionally, it was employed to help structure the text of this thesis. However, ChatGPT was not used to generate or rewrite any text contained in the final draft. All information provided by ChatGPT was carefully verified for accuracy, and no fabricated references or sources generated by ChatGPT were used in this research. For further details on the regulations, please refer to the Leiden University guidelines[9].

[9] `https://www.library.universiteitleiden.nl/students/chatgpt#i-am-a-student-dos-and-don-ts`