Enhancing Deep Reinforcement Learning for Scale Flexibility in Real-Time Strategy Games

Marcelo Luiz Harry Diniz Lemos[a], Ronaldo e Silva Vieira[a], Anderson Rocha Tavares[b], Leandro Soriano Marcolino[c], Luiz Chaimowicz[a]

[a] *Universidade Federal de Minas Gerais, Belo Horizonte, Brazil*
[b] *Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil*
[c] *Lancaster University, Lancaster, United Kingdom*

Abstract

Real-time strategy (RTS) games present a unique challenge for AI agents due to the combination of several fundamental AI problems. While Deep Reinforcement Learning (DRL) has shown promise in the development of autonomous agents for the genre, existing architectures often struggle with games featuring maps of varying dimensions. This limitation hinders the agent's ability to generalize its learned strategies across different scenarios. This paper proposes a novel approach that overcomes this problem by incorporating Spatial Pyramid Pooling (SPP) within a DRL framework. We leverage the GridNet architecture's encoder-decoder structure and integrate an SPP layer into the critic network of the Proximal Policy Optimization (PPO) algorithm. This SPP layer dynamically generates a standardized representation of the game state, regardless of the initial observation size. This allows the agent to effectively adapt its decision-making process to any map configuration. Our evaluations demonstrate that the proposed method significantly enhances the model's flexibility and efficiency in training agents for various RTS game scenarios, albeit with some discernible limitations when applied to very small maps. This approach paves the way for more robust and adaptable AI agents capable of excelling in sequential decision problems with variable-size observations.

Keywords: Deep Learning, Reinforcement Learning, Real-Time Strategy Games, Game-Playing AI

1. Introduction

Advancements in Deep Reinforcement Learning (DRL) have significantly influenced the landscape of Artificial Intelligence (AI) in recent years. This domain has not only achieved remarkable breakthroughs but has also showcased its prowess by outperforming human experts in various environments. These advancements highlight the potential of DRL to excel in problem-solving tasks across diverse domains, ranging from robotics to autonomous vehicles, and beyond. The ability of DRL algorithms to learn from experience and make decisions based on trial and error has revolutionized AI research, offering new avenues for tackling real-world challenges with unprecedented efficiency and adaptability.

The strategic utilization of games as platforms for DRL research offers a unique advantage in this development process. Titles such as Atari (Mnih et al., 2013), Go (Silver et al., 2016), StarCraft II (Vinyals et al., 2019), and Dota 2 (Berner et al., 2019) have emerged as crucial platforms for DRL research. By providing controlled yet complex environments, these games enable researchers to develop and refine DRL algorithms in a flexible and accessible setting. The successful autonomous agents developed for these games highlight the technique's capacity to master complex tasks that were once deemed exclusive to human intelligence. This approach not only propels the field of DRL forward but also enhances our understanding of AI's adaptability and potential for broader applications.

Despite these successes, DRL systems often face challenges related to handling varying input sizes, requiring retraining even if the underlying environment has the same mechanics (e.g. adapting to various map sizes in the same game). This limitation arises because DRL architectures are typically designed to process fixed-size inputs and often include layers that need a specific input size, such as fully connected layers. Attempting to feed data of different sizes into these architectures would result in incompatible data shapes, preventing the model from functioning. Although input resizing methods such as cropping or warping offer a workaround, they introduce a preprocessing step that can cause loss of information or distortions that can hinder the agent's learning.

To address this constraint, our study introduces an innovative architecture incorporating Spatial Pyramid Pooling (SPP) (He et al., 2015) to forge a scale-flexible DRL framework applicable across diverse DRL scenarios. SPP, a method initially devised for scale-invariant tasks in image recognition, is

adapted here to enhance DRL architectures, allowing a neural network to adapt to inputs of varying sizes efficiently. This adaptation not only streamlines and accelerates the training process of DRL agents but also facilitates transfer learning, empowering agents to manage inputs of previously unseen sizes. Additionally, by incorporating multiple map sizes in a unified training regimen, we can improve the agent's learning process, enhancing its capacity for generalization.

We evaluated our approach in two distinct environments: initially on Gym-μRTS (Huang et al., 2021) and subsequently on Frozen Lake (Brockman et al., 2016). The Gym-μRTS environment is a RTS game developed for research, encompassing complex challenges such as expansive state and action spaces, and intricate resource management, unit control, infrastructure development, and combat strategies. Its grid-based map system aligns well with our grid-oriented control strategy, enabling seamless distribution of commands across the entirety of the map. The Frozen Lake environment, provided by OpenAI Gym, is a simulated scenario where an agent navigates a frozen lake to reach a goal while avoiding holes. It provides a simple setting where we can more easily investigate the adaptability and robustness of our proposed DRL architecture while remaining in a task related to RTS games.

Our findings suggest that our novel framework enhances performance across both single and multi-environment tasks, outperforming existing state-of-the-art agents in μRTS. While our agent demonstrated remarkable proficiency in large maps, successfully learning and extrapolating acquired knowledge to analogous scenarios, we also observed instances of learning instability and generalization issues on smaller maps, a phenomenon we analyze within the Frozen Lake environment. As our experiment analysis will reveal, our agent exhibits greater suitability for larger maps rather than smaller ones. Moreover, our μRTS study highlights the benefits of training agents in a diverse range of environments, fostering adaptability and proficiency across various scenarios rather than specialization in narrow contexts.

This paper is a comprehensive and self-contained extension of our previous work (Lemos et al., 2024), integrating all its foundational results while introducing new experiments and discussions that expand on the original findings. The latest experiments were designed to address unresolved questions identified in the prior study, providing a more robust understanding of the potential of a scale-invariant DRL model. Furthermore, discussions are enriched with findings from these new experiments, offering a deeper and more comprehensive analysis. This approach reaffirms the validity of the

original results and extends its implications, providing valuable directions for future research and practical applications.

2. Related Work

In the domain of Real-Time Strategy games, notable efforts have been made to create autonomous agents capable of rivaling human players. A prominent example is DeepMind's AlphaStar (Vinyals et al., 2019), which achieved a remarkable performance, surpassing 99.8% of human players. This result was mainly possible due to the employment of advanced Deep Reinforcement Learning techniques combined with a robust infrastructure and long training sessions. Similar approaches have been pursued by other research teams, often involving the utilization of thousands of CPUs and GPUs (Wang et al., 2021).

Recognizing the resource-intensive nature of RTS games, many researchers have explored alternative platforms with reduced hardware requirements to study various aspects of RTS games. For instance, μRTS (Huang et al., 2021) offers a simplified grid-based framework in Java, while miniRTS (Tian et al., 2017) presents a lightweight RTS environment with a Reinforcement Learning backend. Deep RTS (Andersen et al., 2018) provides a highly configurable platform tailored for DRL research, emphasizing speed and complexity. Additionally, initiatives such as the StarCraft II Learning Environment (SC2LE) mini-games (Vinyals et al., 2017) isolate specific gameplay elements to target distinct challenges individually.

Managing the fluctuating number of units in real-time strategy (RTS) games poses a significant challenge for players. With no reliable method to predict the exact quantity of units at any given moment, players must possess the skill to efficiently micromanage an indefinite number of units throughout a game. Han et al. (2019) devised a solution to this problem by partitioning the game map into a grid and assigning actions to individual cells. By taking inspiration from the Encoder-Decoder architecture typically utilized in image segmentation tasks, they implemented a Grid-Wise Control mechanism that operates independently of the number of units present on the battlefield.

Proximal Policy Optimization (PPO) (Schulman et al., 2017) and Deep Q-Network (DQN) (Mnih et al., 2013) algorithms have demonstrated strong performance across various domains, and the former has been successfully applied in μRTS as well. Huang et al. (2021) leveraged PPO alongside Grid-Wise Control and Invalid Action Masking (Huang and Ontañón, 2022), re-

sulting in the development of an agent capable of efficiently outperforming state-of-the-art opponents. Despite its effectiveness, this model exhibits exhibits the usual lack of flexibility of DRL approaches: changes in map size require alterations to the network architecture. While the encoder-decoder component of the actor can accommodate any map, the critic relies on a fixed observation space due to its fully connected architecture. Consequently, the agent requires predefined layer sizes and must undergo training from scratch whenever a different map dimension is chosen.

Incorporating attention mechanisms into multi-agent control offers a promising avenue for managing diverse observation spaces. An instance of this is evident in the utilization of a Transformer-based approach (Vaswani et al., 2017) for multi-agent credit assignment and joint action evaluation within the context of the Starcraft Multi-Agent Challenge (Samvelyan et al., 2019). Despite their notable successes across various domains, Transformers are susceptible to vanishing and exploding gradients (Pascanu et al., 2013), and strong dependencies on residual branches (Liu et al., 2020).

An alternative approach to addressing the challenge of varying input sizes involves adapting the observation model. Utilizing fixed-size representations may result in inefficient memory usage and processing resources in certain contexts. Given that much of the environment information is associated with entities, entity-based representations can offer notable efficiency gains, especially in sparsely populated domains. Graph Attention Networks (GATs) (Brody et al., 2021) have been applied in single-agent control scenarios using an entity-based methodology within environments like the Arcade Learning Environment (ALE) (Bellemare et al., 2013) and Simple Playgrounds (Jankovics et al., 2022). Moreover, GATs have been employed in Multi-Agent Reinforcement Learning settings to tackle Starcraft mini-games (Yun et al., 2021) using a decentralized approach. However, it's important to note that these applications were primarily tested on specific tasks rather than entire Real-Time Strategy (RTS) matches.

In our novel approach, we integrate Grid-Wise Control with Spatial Pyramid Pooling, creating a scale-invariant architecture tailored for sequential control problems. This innovation yields a versatile and efficient agent capable of seamlessly managing a varied number of units across distinct input sizes without requiring structural modifications. Leveraging insights from the work of Huang et al. (2021), particularly techniques like Invalid Action Masking and Action Composition, our approach is grounded on established methodologies. Additionally, we incorporate Reward Shaping (Mataric, 1994),

accelerating the Reinforcement Learning process by introducing a series of small rewards tailored to guide the agent swiftly toward its ultimate objective.

Our proposed model introduces two significant enhancements over the literature. Firstly, we introduce a novel network architecture featuring Spatial Pyramid Pooling layers, producing a flexible scale-invariant network. Secondly, we implement a revamped training methodology that capitalizes on experiences garnered from maps with varying dimensions and representation sizes. This strategic approach generates a more comprehensive and robust learning process, enriching the agent's ability to tackle complex environments effectively.

3. Background

3.1. Grid-Wise Control

Grid-wise Control (Han et al., 2019) introduces a method for managing entities within environments divisible into a grid structure. This approach, implemented through the GridNet architecture, employs an encoder-decoder mechanism akin to those utilized in image segmentation tasks. Given a grid with dimensions (h, w, n_f), where (h, w) represents the grid's scale and n_f denotes the number of feature planes, GridNet takes as input a state $s \in \mathbb{R}^{h \times w \times n_f}$ and generates an action a_{ij} for each grid position (i, j), with $1 \leq i \leq h$, $1 \leq j \leq w$. The resulting output, defined as an action map a, has dimensions (h, w, c_a), where c_a denotes the action dimensionality.

In Real-Time Strategy (RTS) games, actions are typically characterized by a combination of parameters, such as directing a unit to construct a base at coordinates (x, y). Here, the action type may be designated as *build*, with the structure type specified as *base*, and the target location determined by the coordinates x and y. Consequently, the dimension c_a must encompass all parameters essential for predicting and composing any given action along with its specifications.

3.2. Spatial Pyramid Pooling

Convolutional Neural Networks (CNNs) have profoundly transformed the landscape of Deep Learning, particularly within the field of computer vision. However, traditional CNN architectures typically require fixed input

sizes, primarily due to the presence of fully connected layers. Yet, real-world datasets often comprise images of varying dimensions, posing a compatibility challenge with CNNs. To address this issue, many models resort to preprocessing techniques like cropping (Krizhevsky et al., 2017) or warping (Girshick et al., 2014) to conform images to the required input dimensions. However, these methods are suboptimal, potentially leading to content loss or geometric distortions, thereby diminishing model accuracy and impairing recognition capabilities in certain scenarios.

A more robust solution comes in the form of Spatial Pyramid Pooling (SPP) (He et al., 2015), a computer vision technique designed to generate fixed-length representations irrespective of input dimensions. SPP achieves this by employing multi-level pooling across spatial bins, ensuring output consistency without introducing distortions. Unlike conventional pooling methods, which utilize fixed-size filters, SPP adapts its filter size dynamically according to the input dimensions. Moreover, a predefined number of bins yield a constant-shape representation. As its name suggests, SPP also incorporates multiple filters, creating a pyramid-like structure. Each SPP layer produces an output vector of consistent size, regardless of the input dimensions, as illustrated in Figure 1.

3.3. Gym-μRTS

μRTS is a streamlined version of a RTS game specifically designed to facilitate Artificial Intelligence (AI) research. It operates within a simplified framework that replicates the strategic complexity and decision-making challenges characteristic of the RTS genre but in a more controlled and manageable environment. Set on a rectangular grid, each cell of μRTS can contain only one entity – be it a unit, a building, or a resource – at any given time. This constraint simplifies the spatial component of RTS games, making it easier for researchers to focus on the development and testing of AI algorithms. The ability to customize the size and layout of the map before each game allows for a wide range of scenarios, enabling researchers to systematically evaluate the performance of AI agents across different conditions. Despite its simplicity compared to full-fledged commercial RTS games, μRTS retains the core strategic elements that make the RTS genre a rich domain for AI research, including resource management, unit production, and tactical combat.

To help in the accessibility and flexibility of μRTS for diverse research needs, the game supports various unit types and strategies, mirroring the

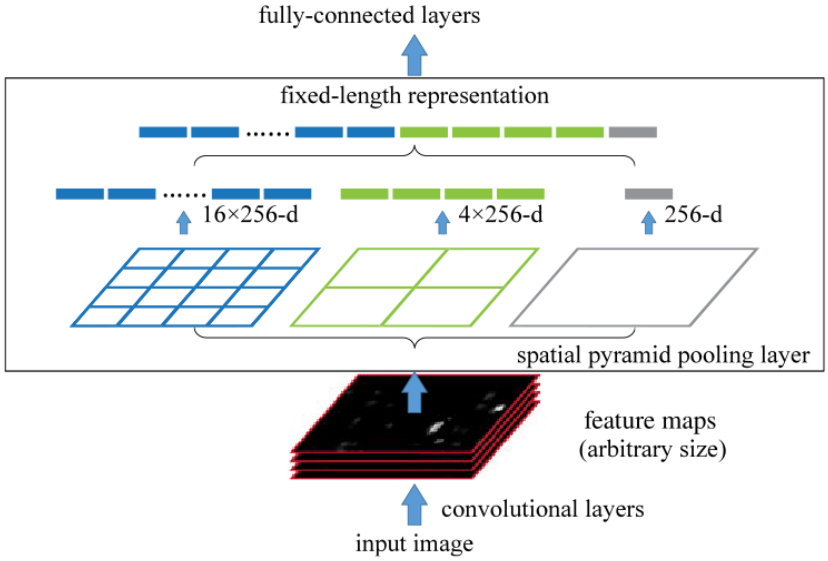

Figure 1: An example of Spatial Pyramid Pooling, assuming that the latest convolution layer yields 256 feature maps. The first SPP level has 16 bins of dimension 256, the second level has 4 bins and the third level has 1 bin. The final representation given by SPP has length $(16 + 4 + 1) \times 256$ regardless of the input. Source: He et al. (2015).

complexity of larger-scale RTS games. Each unit type in μRTS – ranging from workers that gather resources and construct buildings to combat units with distinct offensive and defensive capabilities – requires unique strategic considerations. This setup ensures that while the game complexity is reduced, the strategic depth is not. μRTS also includes configurable AI opponents with different levels of sophistication, from basic scripted agents to more advanced adversaries. This variability allows researchers to challenge their AI models against progressively harder opponents, facilitating the study of incremental learning and adaptation strategies in AI systems. Through these features, μRTS serves as a testbed for algorithmic innovation and a dynamic platform for studying complex interaction patterns and competitive behaviors in AI agents. In Figure 2, we can see an example of what a μRTS match looks like.

OpenAI Gym (Brockman et al., 2016) is a widely used open-source toolkit that provides a diverse collection of environments designed to benchmark and facilitate the development of RL algorithms. By offering a standardized interface and a broad range of tasks, including classic control problems, Atari games, and simulated robotics, OpenAI Gym enables researchers to system-

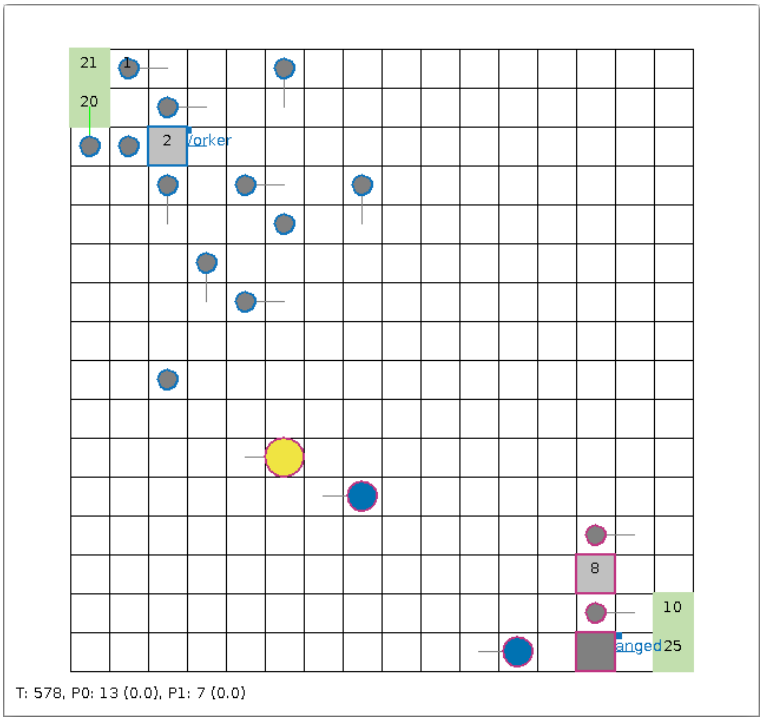

Figure 2: Screenshot of a μRTS match in a 16×16 map that showcases distinct units and structures. Circles represent units, their type delineated by size and color, while squares denote various structures, each color-coded for easy identification.

atically prototype, test, and compare the performance of their algorithms. Its ability to streamline the experimental process allows easy reproducibility and results sharing within the RL community. Moreover, OpenAI Gym's extensibility supports the creation of custom environments, further fostering innovation and the exploration of new RL applications. This toolkit has become an important tool to help advancing the state of the art in RL, as it provides the necessary infrastructure for rigorous experimentation and the development of more robust and generalizable RL solutions.

The integration of μRTS with OpenAI Gym through Gym-μRTS further enhances its utility for AI research by providing a standard interface for μRTS experiments. Gym-μRTS abstracts the complexities of μRTS into a format that is compatible with popular machine learning libraries, simplifying the process of developing and training AI models. The observation space of Gym-μRTS, represented as a tensor (h, w, n_f) that reflects the game's grid

dimensions and the types of entities present, requires AI agents to adapt to variable map sizes and configurations. This flexibility is crucial for developing versatile AI systems capable of generalizing across different environments. By offering a simplified yet challenging platform for RL, Gym-μRTS plays a significant role in advancing the field of AI, providing researchers with a valuable tool for exploring strategic decision-making, learning dynamics, and the scalability of AI algorithms in complex, ever-changing scenarios.

3.4. Frozen Lake Environment

The Frozen Lake environment is a classic grid world simulation in OpenAI Gym, designed to evaluate reinforcement learning algorithms. It represents a grid of frozen tiles, with the agent starting at a designated position and aiming to reach a goal tile without stepping on hazardous tiles. However, the catch is that the ice can be slippery, causing the agent to slide in unintended directions. Due to its inherent unpredictability, it forces algorithms to account for the probabilistic nature of state transitions, making it an effective testbed for evaluating robustness and adaptability. The environment comes with varying sizes and complexities, offering different levels of challenge for reinforcement learning agents. Additionally, the simplicity of the environment allows for clear visualization and interpretation of the learning process.

Each tile in the Frozen Lake environment is represented by one of four types: a start tile (S), a frozen tile (F), a hole (H), or the goal (G). The agent navigates through the grid by taking actions such as moving up, down, left, or right. The goal is for the agent to learn an optimal policy to navigate from the starting position to the goal while avoiding the hazardous holes, making it a popular benchmark for testing reinforcement learning algorithms' ability to handle stochastic environments and long-term planning. Overall, the Frozen Lake environment's blend of simplicity and complexity makes it a versatile and challenging setting for advancing the understanding and development of intelligent agents. In Figure 3 we can see a graphical representation of the Frozen Lake environment.

4. Methodology

To address the limitations associated with fixed input sizes in current RL models, we propose a novel solution that integrates techniques from DQN, PPO, Grid-Wise Control, and SPP. This approach aims to create

Figure 3: Screenshot of the Frozen Lake environment.

scale-invariant Reinforcement Learning agents, offering a more versatile and adaptable framework. This research culminated in the development of two distinct agents. The initial agent was designed specifically for Real-Time Strategy (RTS) Games, where we conducted most of our experiments. Building on the insights gained, a second agent was tailored for Frozen Lake, a single-agent RL environment. This second agent not only served as tool to verify and evaluate the efficacy and potential of our proposed architecture, but also allowed us to investigate and better understand the limitations observed in the first agent.

4.1. µRTS Agent

The first model developed was tailored for µRTS. As discussed previously, a significant challenge inherent to RTS games lies in managing a dynamic number of units and structures. However, conventional RL algorithms are primarily structured to control a single agent. To overcome this obstacle, we

adopted the approach outlined by Huang et al. (2021), integrating Grid-Wise Control with PPO, resulting in an agent capable of managing a dynamic number of units by segmenting the environment into a grid and assigning actions to each grid cell. Notably, since the μRTS environment is inherently grid-based, there was no necessity to devise additional mechanisms to partition the environment, streamlining the implementation process. Subsequently, we augmented the architecture with SPP to achieve a scale-invariant model. In the remainder of this section, we will detail how the model architecture is structured and how we leverage its capabilities to devise a more robust training routine for the agent.

4.1.1. Model Architecture

As shown in Figure 4, the model architecture is divided into three parts: one encoder and two decoders. The encoder consists of convolutional layers followed by pooling layers, designed to process the agent's observation inputs and generate a concise feature map. This feature map is instrumental for learning the optimal policy for the agent and assessing the value of the given environment state. While the baseline model encoder encompasses four pairs of convolutional and pooling layers, we found that reducing it to two pairs of layers retained the agent's performance while significantly decreasing the number of parameters in the network.

The encoder output feeds into the first decoder – the actor's decoder – which is responsible for generating action probabilities for each grid cell. This is done by combining deconvolutional layers with pooling layers, resulting in an output tensor with the same height and width as the initial observation but with a different number of channels. Each output channel is responsible for a parameter of the possible actions and will be used to compose the action the agent will perform. This first path of the network behaves in the same way as the Grid-Wise Control described previously, taking a grid observation as input and outputting an action for each grid cell. Once again, we reduced the reference GridNet actor by removing the last two deconvolutional and pooling layers.

The second decoder – the critic's decoder – uses the encoder output to predict the state's value function. The reference architecture is composed only of fully connected layers, and it flattens the encoder output to pass it onto the critic's decoder. Instead of flattening the encoder's output, we added an SPP layer before the fully connected layers. The SPP layer generates a standardized representation of any input passed to the critic, which allows

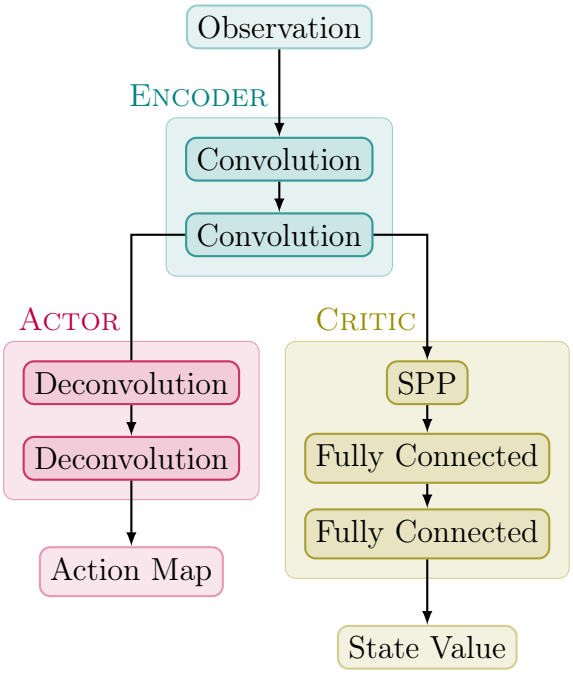

Figure 4: Network Architecture.

the agent to work properly in any environment regardless of the observation dimensions. Despite being a simple innovation, this causes great changes to the actor's behavior and, as shown in our experimental results, improves the agent's effectiveness in most scenarios.

Overall, the revised architecture with an optimized encoder and actor's decoder, as well as an enhanced critic's decoder via the SPP layer, not only reduces the number of parameters in the network but also boosts the model's scalability and adaptability in various environment states, demonstrating robust improvements in the agent's performance as we will see on our evaluation section.

Most of the hyperparameters used by our model follow the guidelines set by Huang et al. (2021), with a few exceptions. Specifically, we reduced the number of convolutional and pooling layers from four to two each, which decreased the model's parameter count while maintaining it's performance. Furthermore, we conducted experiments to determine the best hyperparameters for the SPP layer we introduced, as detailed in Section 5. Comprehensive information about the architecture of our μRTS agent, including all layers

Algorithm 1 PPO with environment swap

for iteration $= 1, 2, \ldots$ **do**
  Select environment E
  for actor $= 1, 2, \ldots$ **do**
    Run policy $\pi_{\theta_{old}}$ in environment E for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size M
  $\theta_{old} \leftarrow \theta$
end for

and their sizes, can be found on our GitHub repository[1].

4.1.2. Expansion of Training Scenarios

The new model's capabilities allow a distinct training approach by leveraging the agent's expertise acquired in one scenario to accelerate learning in other similar settings. Building on this concept, we devise a training routine that utilizes multiple scenarios within a single session, thereby developing a more general agent capable of performing well in distinct environments.

This enhanced training methodology is consolidated in our modified version of PPO – shown in Algorithm 1 – that allows for a more diverse training approach. Before each algorithm iteration, a new environment is selected for the training. The agent then collects experience by interacting with the environment, and the policy is updated as usual. This adjustment generates a diverse training procedure while keeping the algorithm's implementation simple when using widely adopted Deep Learning libraries.

As we will see below, the strategy used to select the environment may also affect the agent's behavior. Different approaches may lead to more consistent learning throughout all environments or prioritize specific environments without forsaking others. For example, let us consider a set of three maps $\{A, B, C\}$, with dimensions $(a \times a)$, $(b \times b)$, and $(c \times c)$, respectively. We could sequentially cycle through $\{A, B, C\}$, ensuring the agent would experience all three distinct representations for the same amount of steps. Another option would be to randomly select the maps following a uniform distribution, creating an unpredictable experience for the player.

[1] Available at `https://github.com/marcelo-lemos/MicroRTS-Py`

Moreover, we can assign different weights to the performance in different scenarios. For example, if we evaluate the agent by its weighted average performance, with weights $(0.6, 0.2, 0.2)$ for (A, B, C), we would prioritize experience on map A. We could tailor the selection strategy to meet this specific need by utilizing a weighted random selection with the same $(0.6, 0.2, 0.2)$ weights, ensuring map A would be prioritized. The strategy selection could also be used as a form of Curriculum Learning (Bengio et al., 2009), where the agent starts playing only in simple maps and, as time passes, more complex maps are added to the selection pool.

This approach to training using multiple environment dimensions enhances the agent's generalization capabilities and allows for tailored learning strategies. By adjusting the environment selection process and incorporating weighted performance evaluations, we can guide the agent's development more effectively, ensuring robust performance across a wide range of scenarios. This method provides a powerful framework for developing adaptable and resilient agents capable of thriving in heterogeneous environments.

4.2. Frozen Lake Agent

Our second model was tailored to an agent tasked with navigating a grid-based environment, specifically the Frozen Lake environment. Within this setting, we made the assumption that the agent possessed complete visibility of the grid, encompassing its dimensions, obstacles, goals, and its own position. In an environment characterized by these features and fixed-size inputs, a conventional Q-Learning could be employed to address the navigation challenge. However, the inherent limitation arises when dealing with varying input sizes. Traditional Q-Learning relies on a table of Q-values tied to specific states, thereby becoming constrained by input dimensions. Alternatively, a DRL approach could be employed, with DQN emerging as the most straightforward option rooted in Q-Learning principles. However, it would also encounter difficulties with diverse input sizes, as the convolutional layers in DQN would generate multiple output sizes, which is incompatible with the expected dimensions of the fully connected layers responsible for outputting the Q-values.

In the progression of our research, the initial application of the PPO algorithm within a RTS game environment served as a robust framework for exploring complex decision-making processes. However, upon encountering specific learning challenges within this context (detailed on the next sections), our investigation took a pivotal turn towards a more granular analysis using

the Frozen Lake environment. This transition prompted us to adopt the DQN algorithm, a choice driven by several key considerations. Firstly, the Frozen Lake environment is typically solvable by employing a traditional Q-Learning agent, so DQN's relative simplicity and direct lineage to Q-learning principles offer a streamlined approach for dissecting and addressing the learning challenges we encontered. Furthermore, DQN's value-based approach streamlines the agent's architecture, facilitating a clearer understanding of how our modifications influence its behavior. By leveraging DQN's more straightforward framework, we could iterate rapidly and refine our architectural innovations, ensuring they are not only effective in a simplified context like Frozen Lake but also scalable to the complex dynamics of an RTS environment when integrated with PPO. This strategic choice of algorithmic transition thereby allows for a more focused and profound exploration of our proposed architectural modifications, ensuring their applicability and effectiveness across a spectrum of learning environments and challenges.

In the Frozen Lake environment, we implemented an adaptation of the conventional CNN architecture, integrating a Spatial Pyramid Pooling (SPP) layer situated between the final convolutional layer and the network's first fully connected layer. This design mirrors the approach adopted for both the encoder and critic components of the μRTS model. The SPP layer is designed to process inputs of varying dimensions, outputting a uniform-sized representation irrespective of the input size. This combination of convolutional layers and SPP layers yields a versatile and powerful encoder, capable of producing standardized representations across diverse environments that we can then use to infer the Q-values of the pairs (s, a). It is important to highlight that this approach does not employ any form of input preprocessing, thereby preserving the entirety of the state's information without introducing any distortions.

In the classic Frozen Lake challenge, the agent's objective is to learn navigation across a singular map, traditionally encoding the state as a single integer that represents the agent's location. However, to accommodate navigation across varying map sizes and different maps of the same size, we revised the state representation strategy. Instead of relying on a singular integer, our approach utilizes three binary feature planes for the observation model. Each plane adheres to dimensions $(h \times w)$ of the map, where h and w signify the height and width, respectively. The first plane marks the agent's current location; the second delineates the positions of the holes; while the third highlights the goal's location. This adjustment ensures our agent is

equipped to intelligently navigate any map on the Frozen Lake environment.

Given the inherent simplicity of the Frozen Lake environment, which is typically solvable through tabular methods, we were mindful to avoid overcomplicating our model. Consequently, in our initial approach we adopted a basic CNN design for the DQN. This design comprises a single convolutional layer, succeeded by a pooling layer. These layers are then connected to a hidden fully connected layer, culminating in a final output layer responsible for generating the Q-values associated with each possible action.

In designing this network, we intentionally configured the initial layers to act as an encoder, simplifying the detailed observation of the map into an abstracted representation. With this goal in mind, we initially matched the dimensionality of the hidden fully connected layer to the total count of possible states within the map. For instance, in a setting with an 8×8 map, there are 64 unique states, prompting us to set the hidden layer's size to 64 units. This configuration allows our agent to undertake dual responsibilities: first, to learn the encoding of observations into their respective states efficiently, and second, to learn the most promising action for each state based on the expected outcomes.

Upon verifying that our model demonstrated the capacity to learn effectively on smaller maps with the initial architecture, we introduced a Spatial Pyramid Pooling (SPP) layer situated between the pooling and the fully connected layers. The primary role of the SPP layer is to normalize the output from the pooling layer across varying map sizes, thereby equipping our agent with the ability to apply its learned insights universally, regardless of the map dimensions. This enhancement ensures that our model's adaptability and learning efficiency are maintained even when transitioning to environments of different scales.

To further enhance the model's learning ability, we incorporated a reward function tailored to the specifics of the Frozen Lake environment. This reward function was designed to provide more granular feedback to the agent, rewarding not only the successful navigation to the goal but also intermediate milestones such as avoiding holes and progressing towards the goal. Specifically, we introduced small positive rewards for each step that brought the agent closer to the goal and small penalties for steps that either moved the agent away from the goal or brought it dangerously close to holes. Additionally, we incorporated a negative reward for falling into a hole and a significant positive reward for successfully reaching the goal. This reward system aimed to mimic real-world navigation challenges, encouraging the agent to develop

a more sophisticated understanding of the environment and fostering a more strategic approach to decision-making. By providing continuous and context-sensitive feedback, we ensured that the agent's learning process was not only focused on end goals but also on optimizing the path taken to achieve these goals, thereby enhancing the agent's overall performance and resilience in varying and complex scenarios.

Details about the architecture of our Frozen Lake agent, including all layers and their sizes can be found on our GitHub repository[2].

5. μRTS Results

In a series of experiments conducted within the Gym-μRTS framework, we thoroughly assessed the performance of our model, systematically benchmarking it against state-of-the-art agents. Following the experimental protocol outlined by Huang et al. (2021), we designated CoacAI[3] as the primary adversary in our trials, and used diverse opponents in training to grant a more complete experience. We trained our model against CoacAI, RandomBiasedAI, LightRushAI, and WorkerRushAI. CoacAI is the winner of the 2020 IEEE CoG MicroRTS Competition, and the other bots are part of the μRTS framework and are used as baselines for the competitions. While the other opponents are not as relevant as CoacAI, playing against them ensures our agent will acquire knowledge of many strategies and will not easily lose to different opponents.

Furthermore, we use the best model[4] developed by Huang et al. (2021) as a baseline to compare and evaluate our approach and the impact of the proposed modifications. Unless stated otherwise, our proposed model used a single-layer SPP with 4×4 bins and trained with a sequential map selection where maps were swapped every 100,000 steps. As we show below, this was the best configuration we have found.

All agents were trained for 300 million steps, with the resulting policy being tested in 100 games against CoacAI. Three map configurations were used, with sizes 8×8, 16×16, and 24×24. Since the baseline model cannot play in multiple map dimensions without changes to its architecture, we have

[2]Available at `https://github.com/marcelo-lemos/deep-frozen-lake`

[3]Available at https://github.com/Coac/coac-ai-microrts

[4]The model did not receive an official name but was referred to as the combination of GridNet + PPO + invalid action masking + diverse bots + encoder-decoder.

used three versions, one for each map, adapting the first fully connected layer of their critic to accommodate the encoder's output. All the hyperparameters used on the experiments can be found at our GitHub repository[5].

The game results during test time were the primary metrics used to evaluate the agents. For easier visualization, we opted to simplify it, and instead of using the raw results, we employ a Score metric where a player gets 1 point for each victory and 0.5 points for each draw.

In RL, agents receive rewards for completing certain actions or reaching specific states, generally associated with their final goal. Since we employ reward shaping, our agent receives rewards from several small actions, such as collecting resources or attacking enemy units. One of the main rewards we use is a win/loss reward, which is always received at the end of a game: 1 in case of a win, 0 in case of a draw, or -1 in case of a loss. The sum of all rewards received across a game (or episode) is called the episodic return, which does not have an upper bound in our case but has a lower bound of -1 in case the only reward received was of a loss. We use the win/loss rewards and episodic returns received during training as additional metrics to analyze learning progression. Since our agent plays thousands of games during training, we apply a moving average to better visualize our graphs. Values closer to -1 indicate the agent is losing more games than winning, while values closer to 1 indicate the opposite. Values close to zero indicate the agent is winning almost as much as it is losing. Our experiments are divided into four categories as follows.

5.1. Proposed vs Baseline Model

In this first experiment, we compare our proposed model with the best version developed by Huang et al. (2021) playing against CoacAI in two different scenarios: specialized and generalist.

5.1.1. Specialized Scenario

For each of the three maps, an agent of each model was trained and tested exclusively on it. As shown in Figure 5, the proposed model outperformed the baseline in all three maps. The greatest difference occurred on the 8×8 map, where the original lost all 100 games against CoacAI, while the proposed version achieved a score of 70 points. This disparity highlights

[5]Available at `https://github.com/marcelo-lemos/MicroRTS-Py`

a critical limitation of the baseline model: its architecture and hyperparameters were tailored for the 16×16 map, which likely caused suboptimal performance in environments requiring different strategies. The 16x16 and 24x24 maps are relatively large, causing the games to last longer and requiring strategies that focus on developing more military structures and combat units. On these maps, long-term planning and gradual build-up are essential for success. In contrast, the 8x8 map is significantly smaller, favoring quicker strategies. Here, efficiency in creating a maximum number of simple workers and deploying them for immediate attacks becomes crucial. The baseline model, designed for the larger 16x16 environment, struggles with the rapid, aggressive tactics needed for the 8x8 map, whereas the proposed model adapts better to these distinct scale demands.

Figures 6 and 7 show that the win/loss reward and the episodic return are very close for the two models tested, except for the 8×8 map once again. On this smaller map, the baseline model failed to learn effective gameplay strategies, as evidenced by its declining performance after 100 million steps, from which it could not recover. This suggests a critical deficiency in the baseline model's adaptability and robustness when confronted with different environmental constraints.

Our novel architecture, designed with greater flexibility, demonstrates consistent performance across all tested environments. It does not compromise effectiveness when focusing on a single environment and shows improved results. We also notice that the episodic returns differ from one map to another, which can be attributed to our use of reward shaping. In larger maps, units must traverse greater distances, leading to longer game durations and allowing the agent to accumulate more rewards from a series of smaller actions. This accumulation contrasts with the more immediate, direct rewards available in the shorter games of smaller maps.

5.1.2. Generalist Scenario

Since only the proposed model can play any map without structural changes, we compare the baseline agent results in single map training (as above) with the proposed model in a generalist setup, where the training occurs over multiple maps.

Figure 8 shows that the specialized agents of the baseline model outperformed our generalist agent in two of the three maps. On 16×16, the baseline received 36.5 more points than the proposed model. However, on the 24×24, the performance gap was much narrower, with the baseline only scoring 3.5

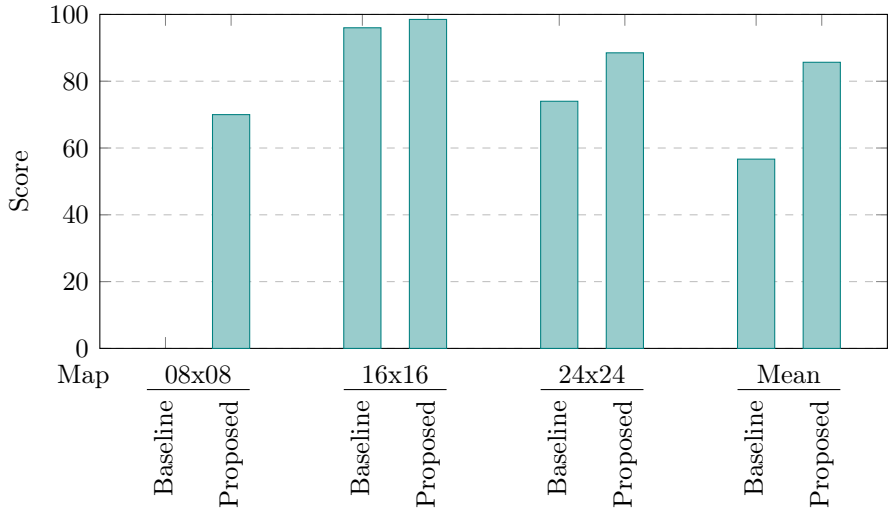

Figure 5: Comparison of proposed and baseline models' scores across three map dimensions in the specialized setting. The proposed model outperforms the baseline on all tested dimensions.

points higher. Lastly, on the 8×8 map, the proposed model achieved 97 points against zero of the baseline. When considering all maps, our proposed model achieved a better mean score, almost 20 points higher than the baseline.

An important detail to note is that the proposed model had a third of the total training budget of the baseline model in this setting. The baseline had to be trained in each map individually for 300 million steps, totaling 900 million. In contrast, the proposed model trained a single time for 300 million total. Despite this significantly reduced training budget, the proposed model demonstrated a notable performance improvement.

Figure 9 shows that the win/loss reward received by the generalist agent is very similar to the baseline on maps 16×16 and 24×24. This suggests that our proposed generalist model is nearly as effective as the baseline when considering these larger maps. Figure 10, however, shows that the generalist agent's episodic returns are biased towards the 8×8 baseline. Since the generalist agent is trained on all three maps for the same number of steps, and because the 8×8 map is smaller and games are shorter, the agent completes more episodes on this map, skewing the curve towards the 8×8 baseline.

In conclusion, while the baseline model's specialized agents perform slightly

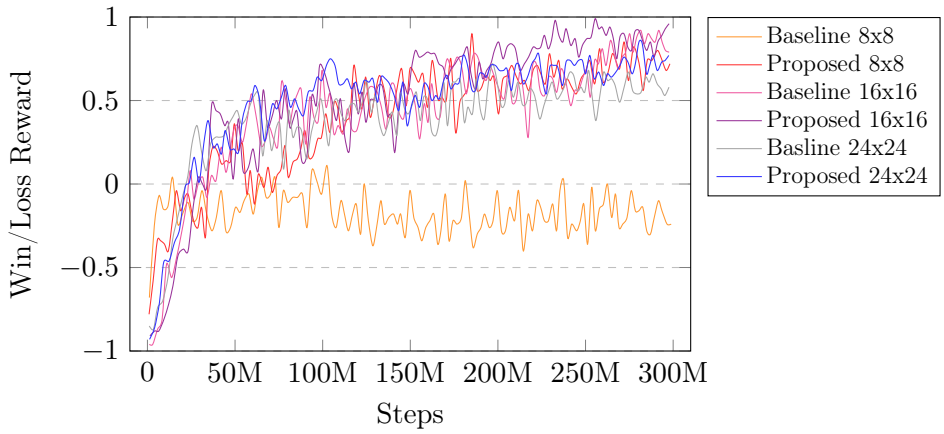

Figure 6: Comparison of win/loss rewards between proposed and baseline models across three map dimensions in the specialized setting. Both models exhibit similar performance trends, except on the 8×8 map, where the baseline model struggled with learning optimal winning strategies over time.

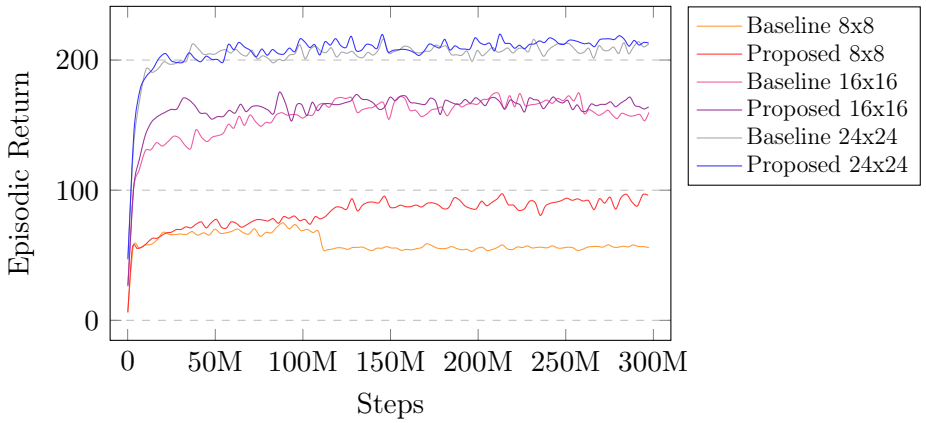

Figure 7: Comparison of episodic returns between proposed and baseline models across three map dimensions in the specialized setting. Both models exhibit similar performance trends, except on the 8×8 map, where the baseline model struggled with learning optimal winning strategies over time. Larger maps result in longer games, leading to greater cumulative rewards per game for the agents.

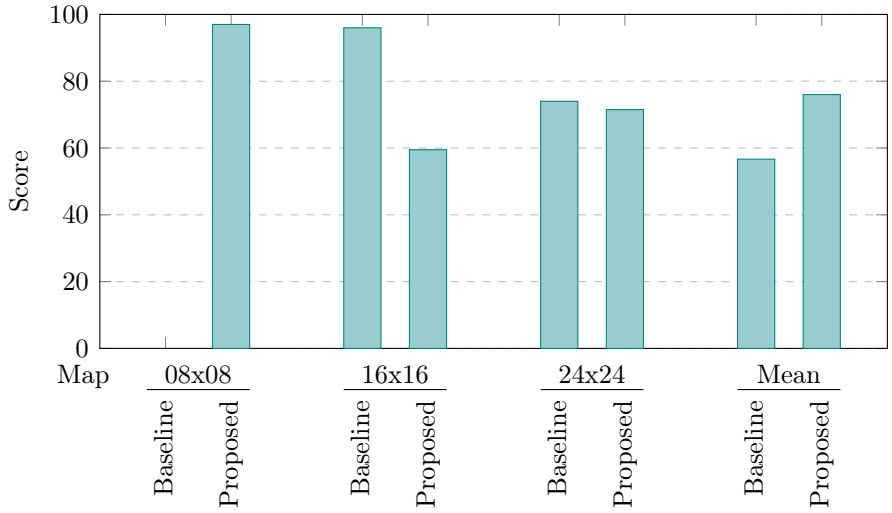

Figure 8: Comparison of proposed and baseline models' scores across three map dimensions in the generalist setting. The baseline agent outperforms the proposed model in two of the three maps, but the proposed model achieves a greater mean score.

better on individual maps, the proposed generalist approach offers superior overall performance and efficiency across diverse environments. The ability to generalize across different map sizes and strategies demonstrates a significant advantage in scenarios where agents must perform well in diverse settings. However, the training routine must be carefully crafted to prevent biases towards some maps or strategies.

5.2. Specialized vs General Training

To consolidate whether training in various scenarios instead of focusing on a single environment is advantageous for the agent, we tested four agents of our proposed model with the same architecture and configuration, each trained in different map settings: (i) 8×8 map only, (ii) 16×16 map only, (iii) 24×24 map only, and (iv) all three maps. For clarity, from now onward, we will refer to each agent as S-08x08, S-16x16, and S-24x24 for those trained solely on maps 8×8, 16×16, and 24×24, respectively. The agent trained on all maps will be referred to as *generalist*. Despite the different training, they were all evaluated on all three maps and we also included a 32×32 map in the evaluation that was not included in any agent's training.

Figure 11 shows that the specialized agents outperformed the generalist agent in two of the three maps. On 16×16, the S-16x16 received 39 more

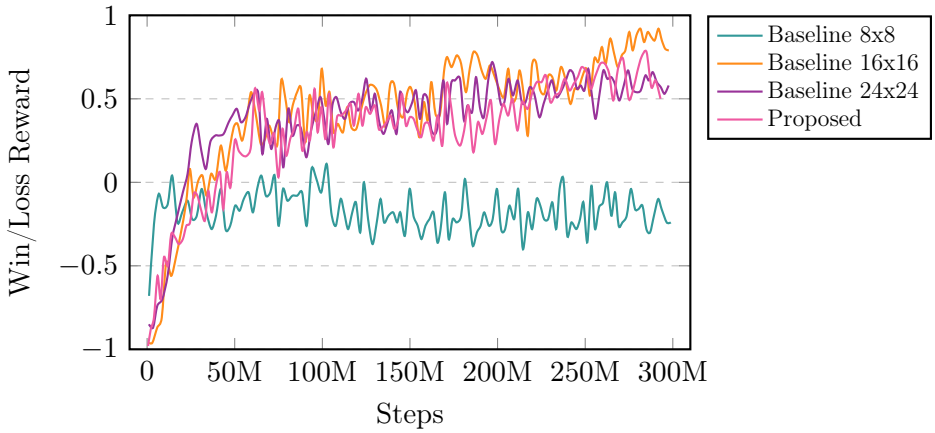

Figure 9: Comparison of win/loss rewards between proposed and baseline models across three map dimensions in the generalist setting. Both models exhibit similar performance trends, except on the 8×8 map, where the baseline model struggled with learning optimal winning strategies over time.

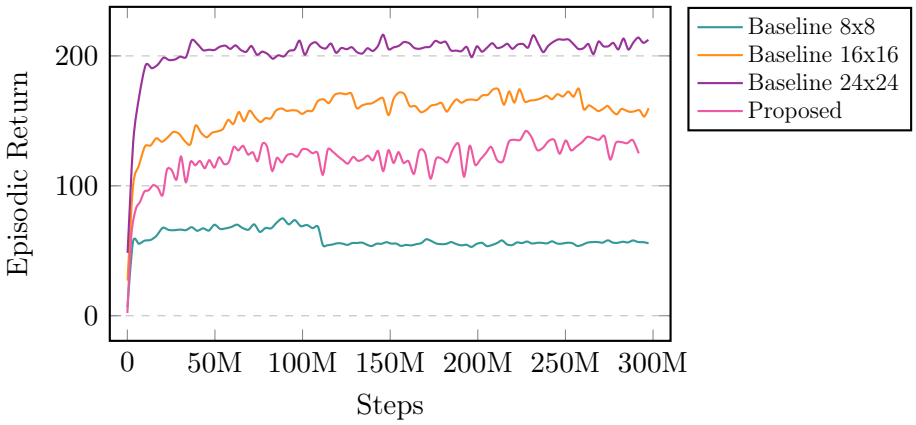

Figure 10: Comparison of episodic returns between proposed and baseline models across three map dimensions in the specialized setting. The proposed model's performance cannot be directly compared to the baselines' due to its training on multiple map dimensions, each offering distinct maximum achievable cumulative rewards.

points than the generalist, but on the 24×24, the difference was a lot smaller, with the S-24x24 being only 17 points better. However, on the 8×8 map, the generalist outperformed the S-08x08, achieving 97 points compared to the 70 points of the specialized agent. Lastly, on the 32×32 map – which was not seen during training – the generalist achieved the second-best performance, with 53 points against the 65 points of the S-24x24. All four maps present the same structure and units, the only difference being the map's scale. This factor gave an edge to S-24x24 due to the similarity in the scale of the training map and the 32×32 map. Considering all maps, the generalist agent achieved a better mean score, 17 points higher than the second place.

Figures 12 and 13, illustrate the win/loss reward and episodic return, respectively, where we note patterns similar to those observed in the previous experiment. A major difference was the performance of the S-08x08 agent, which, unlike the baseline model, managed to attain good results on map 8×8. The flexibility of our architecture allowed the S-08x08 to adapt better to the specific demands of the smaller map.

While specialized agents may have an edge on their respective training maps, the generalist agent demonstrates an overall superior performance. The generalist's ability to adapt and perform well across different environments, including the previously unseen 32×32 map, highlights the advantages of training on multiple scenarios. This adaptability suggests that a more generalized training approach can lead to more robust performance in varied settings, compared to agents trained exclusively on single environments. This approach ensures adaptability and robustness but also prepares the agent for unforeseen challenges, making it a more versatile and efficient solution for diverse environments.

5.3. Environment Selection

As discussed before, our training method involves swapping environments mid-training. To investigate the impacts of the strategy used to select the new environment, we examined both the method and the frequency of environment selection.

5.3.1. Selection Method

Two methods were verified, random and sequential. In the sequential method, we cycle through a predefined sequence of maps from smallest to largest.

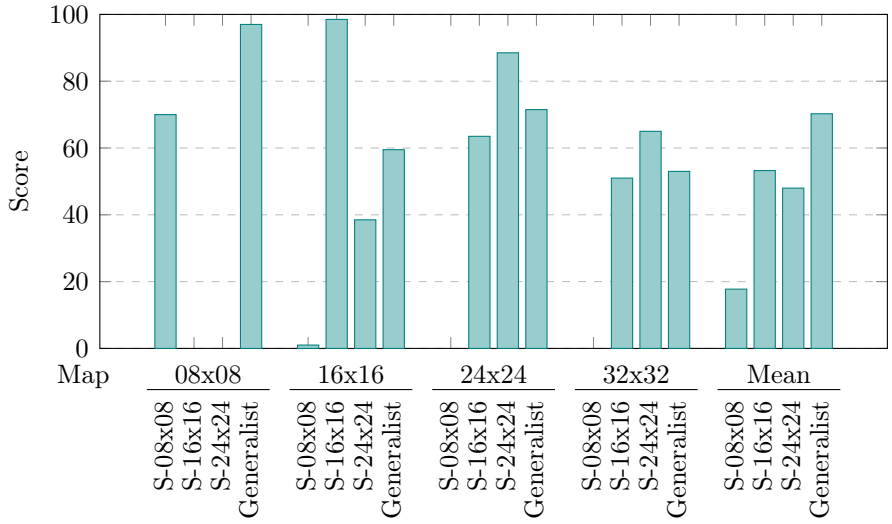

Figure 11: Comparison of specialist and generalist models' scores across four map dimensions. The generalist model exhibits the best or second-best performance on all maps and achieves the best performance overall.

As seen in Figure 14, the sequential selection method outperformed the random method in all three maps tested. This difference was especially pronounced on the 8x8 map, where the random method lost all 100 games, whereas the sequential method won 97 games. Examining the episodic return of both methods, depicted in Figure 16, reveals a significant learning instability for the random method compared to the sequential method. This instability directly correlates to the reduction of the win/loss reward in Figure 15, ultimately affecting the resulting policy. Notably, sequential selection promotes a more consistent and smoother learning process.

The sequential method's superiority can be attributed to its structured progression, which likely allows the agent to develop and reinforce strategies incrementally. By starting on smaller maps and gradually moving to larger ones, the agent can build on its previous experiences, leading to more stable and effective learning. In contrast, the random method's lack of structure can cause the agent to randomly encounter vastly different environments, disrupting the learning process and leading to instability.

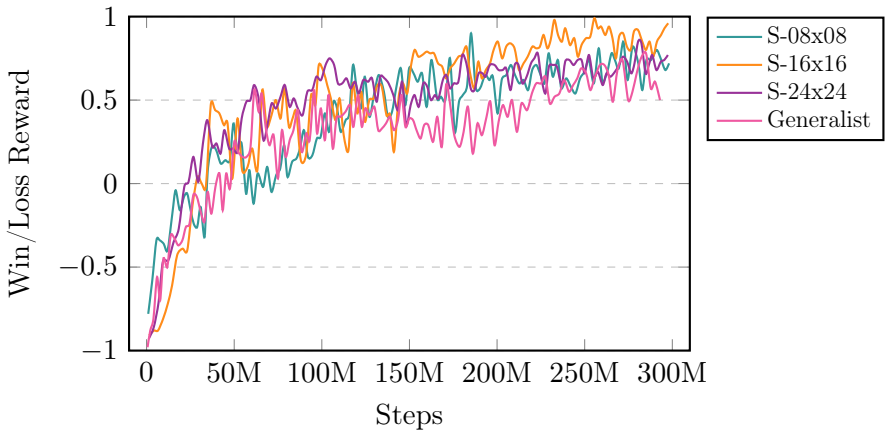

Figure 12: Comparison of win/loss rewards between specialized and generalist models across three map dimensions during training. All models exhibit similar performance trends.

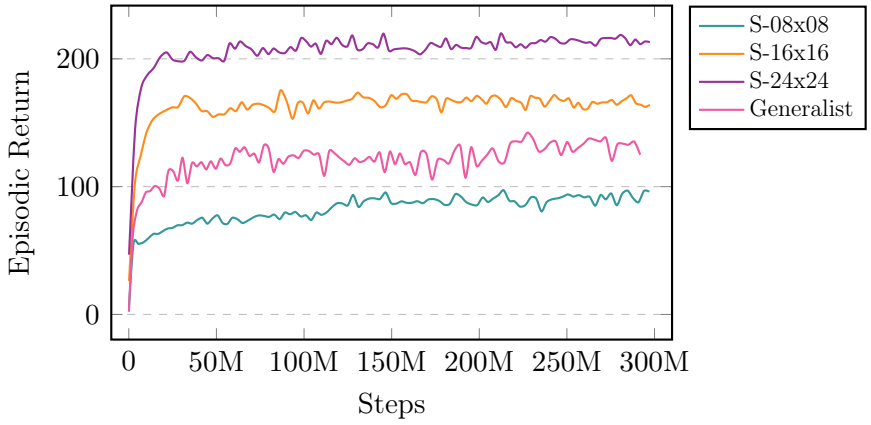

Figure 13: Comparison of episodic returns between specialized and generalist models across three map dimensions during training. The episodic returns of the generalist model fall between those of the S-08x08 and S-16x16 models, reflecting intermediate cumulative rewards due to its training on multiple map dimensions.

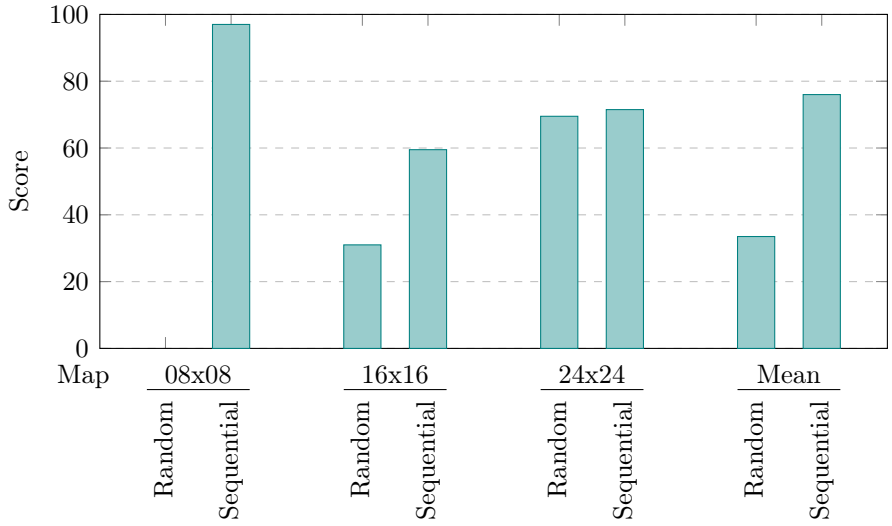

Figure 14: Comparison of models' scores using random and sequential environment swap across three map dimensions. The sequential method consistently outperforms the random method in all maps.

5.3.2. Change Frequency

To evaluate the change frequency's impact, we utilized only the sequential method and ensured our agent experienced each environment for the same total steps. We tested two different frequencies, one changing every 100 million steps – causing each environment to be seen a single time – and one changing every 100,000 steps. We refer to them as A-100M and B-100K, respectively.

This experiment aims to understand the effects of long-term versus short-term environmental exposure on the agent's learning and adaptability. The 100 million steps frequency (A-100M) allows the agent to thoroughly explore and learn each environment in isolation, potentially leading to stronger, more environment-specific skills and strategies. This setup ensures each environment will be seen only once during training and tests the agent's ability to maximize performance in a stable setting before encountering a new environment. Conversely, the 100,000-step frequency (B-100K) introduces more frequent environmental changes, challenging the agent to adapt quickly to new conditions and develop more generalizable skills.

As seen in Figure 17, the agent B-100K, trained with more frequent swaps, achieved better results on the test games on most maps, except for the 24×24

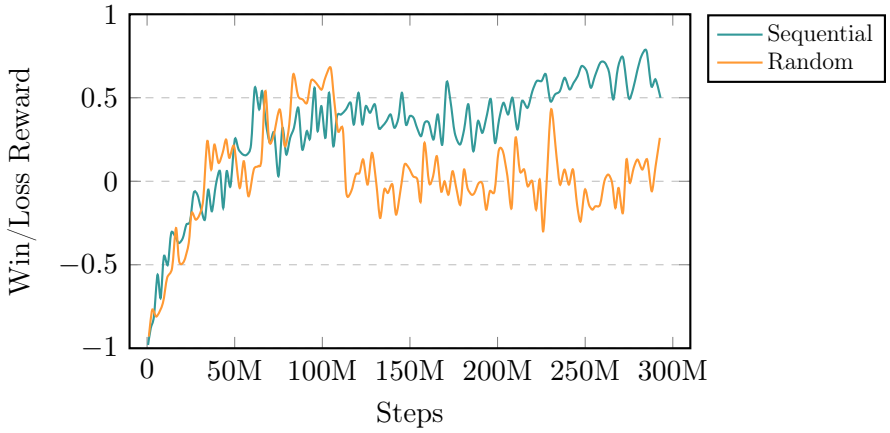

Figure 15: Comparison of win/loss rewards between models using random and sequential environment swap. The sequential method demonstrates a consistent improvement trend over time, whereas the random method experiences significant performance drops.

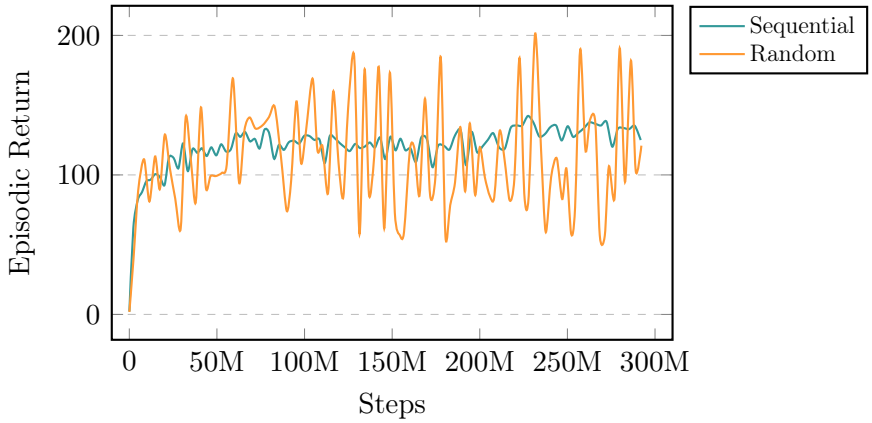

Figure 16: Comparison of episodic returns between models using random and sequential environment swap. The random method exhibits unstable returns compared to the sequential method.

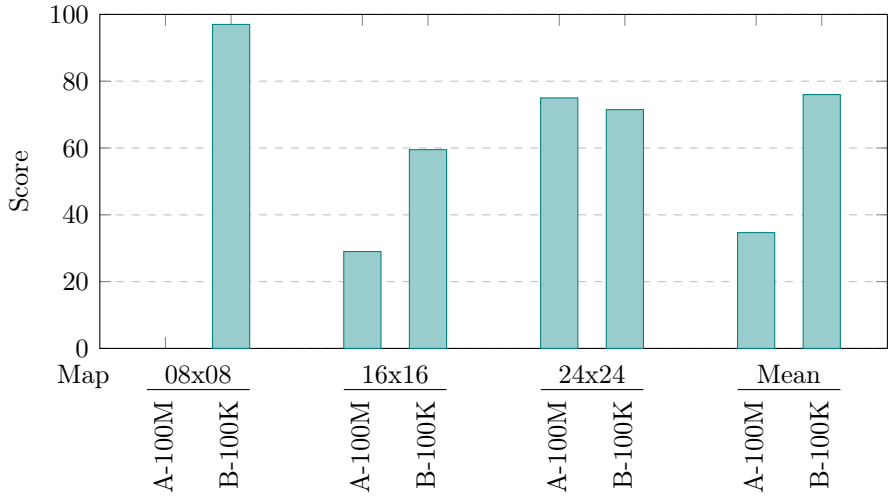

Figure 17: Comparison of models' scores when the environment is swapped every 100K or 100M steps across three map dimensions. Swapping every 100K steps generally results in better performance, except on the 24×24 map, where swapping every 100M steps yields superior results by a small margin.

map, where the score was 3.5 points below A-100M. Figure 18 shows that the agent that trained for longer periods before swapping environments presented a big drop in performance during the change of context. This suggests that the agent specialized in a single map after training on it for an extended period, but struggled to adapt when the environment changed, taking time to adjust its strategies to the new situations.

Figure 19 shows the episodic return, where we can clearly see when the map changes occurred for the A-100M agent, marked by noticeable increases on the episodic return. In contrast, the more frequent map changes for agent B-100K led to smoother and more consistent learning. This continuous exposure to different environments helped the agent maintain flexibility and prevented overfitting to any single map.

The agent trained with more frequent swaps (B-100K) achieved better overall results and demonstrated smoother and more consistent learning. This suggests that maintaining a dynamic and varied training regimen, with frequent exposure to different environments, is crucial for developing agents capable of adapting to multiple scenarios encompassing different scales.

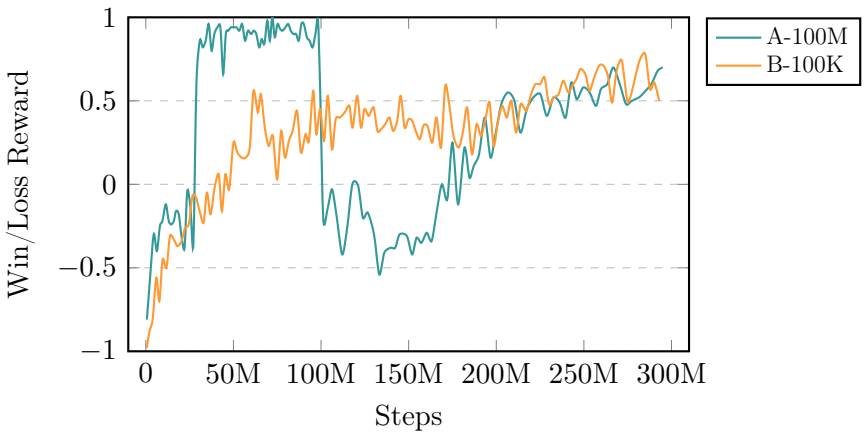

Figure 18: Comparison of win/loss rewards when the environment is swapped every 100K or 100M steps. The model A-100M stays for longer periods on the same map and exhibits significant performance drops when the environment is swapped. Meanwhile the model B-100K exhibits more consistent improvements.

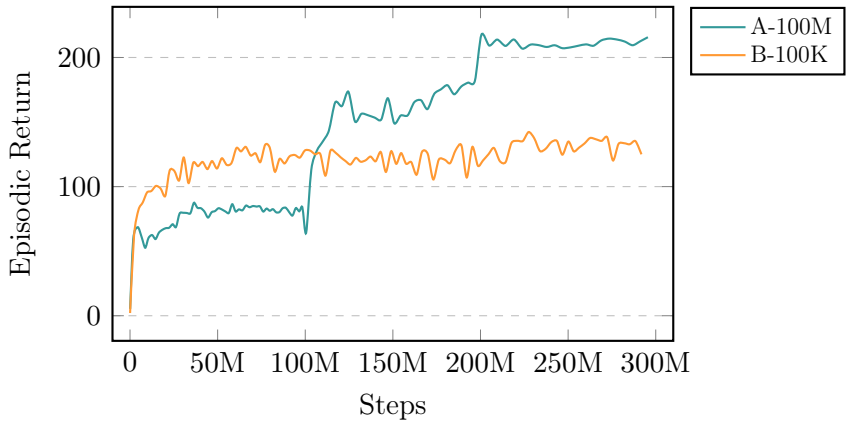

Figure 19: Comparison of episodic returns when the environment is swapped every 100K or 100M steps. Model B-100K, which experiences more frequent swaps, displays a smoother curve. In contrast, Model A-100M shows distinct steps in episodic returns corresponding to each environment swap.

5.4. SPP Layer Size

We study the impact of different sizes of the SPP layer in our agent by comparing four different compositions. The layers tested are (i) a single layer with 2×2 bins, (ii) a single layer with 4×4 bins, (iii) a single layer with 8×8 bins, and (iv) three layers with 2×2, 4×4, and 8×8 bins. All four architectures were trained and tested on all three map dimensions.

The 2×2 bins capture coarse spatial information, summarizing large regions of the input, which can help the model understand broader spatial patterns and context. This configuration can be useful where the overall structure is more important than fine details. The 4×4 bins provide a mid-level spatial resolution, balancing between capturing overall patterns and finer details. This intermediate resolution can help the model detect features that might be missed by either too coarse or too fine resolutions. The 8×8 bins offer a fine-grained representation, preserving detailed spatial information. This high resolution is important for tasks that require precise localization and recognition of small features within the input data. By combining all three sizes in the fourth configuration (2×2, 4×4, and 8×8 bins), we aim to leverage the strengths of all three previous configurations. This multi-layer approach allows the model to capture spatial information at different granularities, potentially improving its ability to generalize across various tasks and environments. This is particularly important in scenarios where the agent must simultaneously understand the global context and local details.

As shown in Figure 20, single-layer architectures with 2×2 and 4×4 bins attained the best results, with mean scores of 73.5 and 76, respectively. In contrast, the bigger architectures exhibit worse performances, especially on map 16×16. The small single layers displayed better generalization than bigger or multiple ones. Figures 21 and 22 show that all four configurations performed closely during training. The biggest single layer, with 8×8 bins, deviated more from the others. Its win/loss reward received dipped around 80M and 260M steps. It eventually recovered from the first dip but not the second one, which surely impacted the final policy.

The superior performance of the 2x2 and 4x4 bin configurations suggests that smaller bin sizes in the SPP layer facilitate more effective feature extraction and generalization. These configurations likely strike a balance between capturing sufficient spatial information and maintaining computational efficiency, leading to better adaptability across different map sizes. In contrast, the poorer performance of the larger 8x8 bin configuration can be attributed

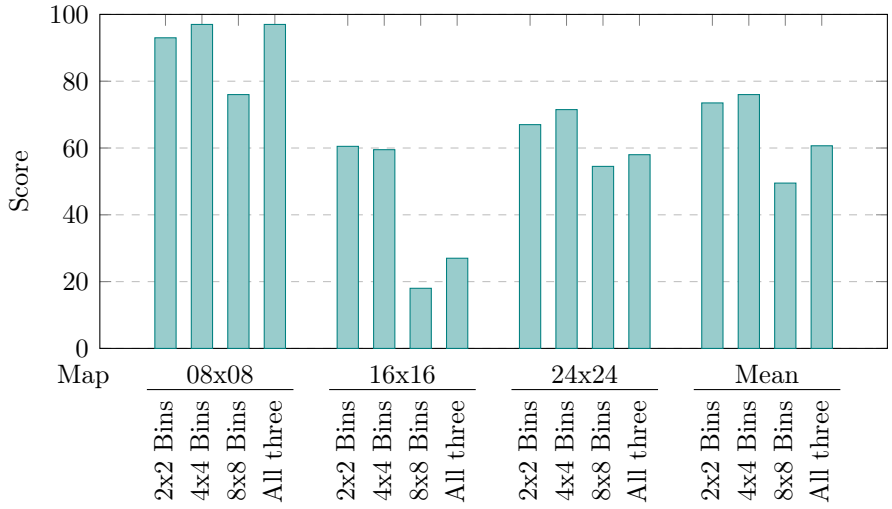

Figure 20: Comparison of models' scores using different SPP layers across three map dimensions. The model with 4×4 bins exhibits the best performance overall and is closely followed by the model with 2×2 bins. The model with 8×8 shows the lowest performance among all configurations.

to its tendency to overfit to specific spatial patterns within the training data, resulting in less robust generalization to new environments. The combination of multiple SPP layers (2x2, 4x4, and 8x8 bins) also did not perform as well, possibly due to the increased complexity and potential redundancy in feature extraction, which might have hindered the agent's ability to develop a coherent strategy.

6. Frozen Lake Analysis

In the previous section, we observed a learning instability on smaller maps, notably on the 8×8 size. The agents tested exhibited either exceptionally high or exceedingly low score ratings on the smallest map. Building upon this observation, we devised experiments for the Frozen Lake environment that would allow us to investigate this issue in more detail. This section presents and discusses the experiments conducted to evaluate the effectiveness and scalability of our proposed architecture in the Frozen Lake environment. The primary focus was to examine the model's capability to generalize its learning from smaller to larger maps. This aspect is crucial for reinforcing the notion that a truly adaptive model should not only excel

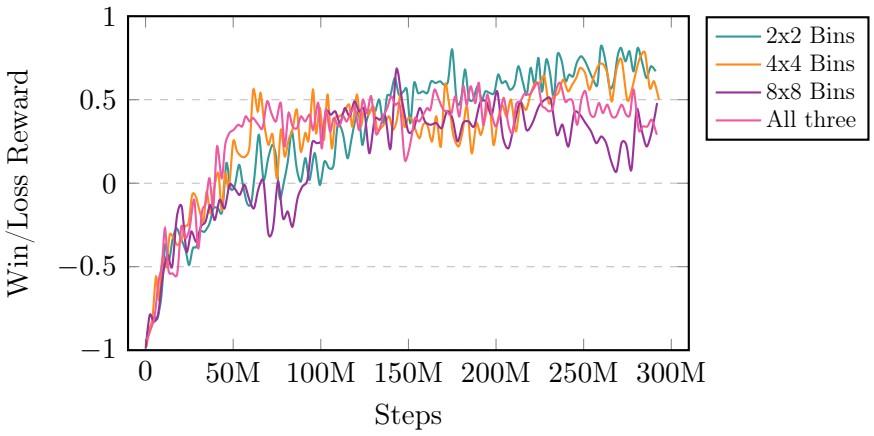

Figure 21: Comparison of win/loss rewards between models using different SPP layers. All models exhibit similar performance trends.

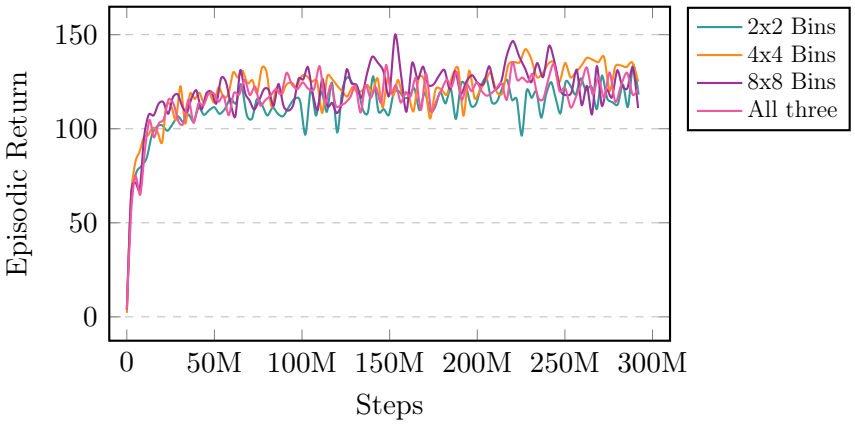

Figure 22: Comparison of episodic returns between models using different SPP layers. All models exhibit similar performance trends.

in environments it has been directly trained on but also demonstrate a high degree of transferability and performance across scenarios of different scales. To this end, we selected a range of map sizes similar to those used on our previous experiments – 8×8, 16×16, 24×24, 32×32, 48×48, and 64×64 – to thoroughly explore how our model navigates and adapts to varying levels of complexity. Given our emphasis on scalability, we ensured consistency in the features and obstacles present across all maps, differing only in their spatial dimensions, to isolate the effects of scale on the model's effectiveness. In all maps our agent starts at the upper-left corner of the map and must traverse an U-shaped path to reach the goal on the upper-right corner. In Figure 23, we can see the 8x8 map used on our experiments. All other maps present the same path, but in a different scale.

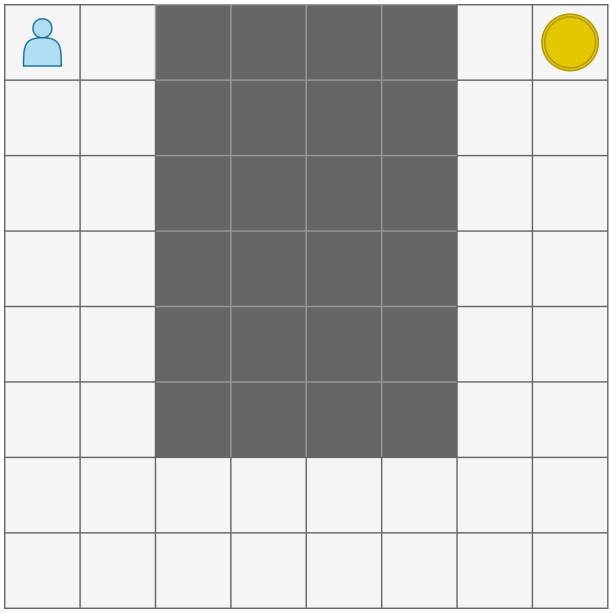

Figure 23: Illustration of the 8x8 Grid Utilized in Our Frozen Lake Experimentation: Starting from the upper-left corner, the objective of the agent is to navigate to the goal located at the upper-right corner. The grid features cells depicted in white, representing frozen terrain that is safe for the agent to traverse, contrasted with the dark cells, which represent holes to be avoided.

6.1. Encoder Generalization

In this exploration, our primary focus is on assessing the consistency of spatial feature representations generated by the SPP layer within our network

across varying map scales. To this end, we initially trained our model on an 8×8 map for 30,000 steps, then guided the agent along a predetermined route across maps of different sizes. During this process, we captured and stored the SPP layer outputs for each state encountered along the trajectory in an array A, with each element a_i representing the SPP output at the i-th step. Our goal is to analyze and quantify the differences in representations for analogous states across varying map sizes.

Given that larger maps naturally lead to longer trajectories for the identical route, we implemented a normalization process to facilitate a more equitable comparison. This process involves adjusting for map dimensions, using the 8×8 dimension as a standard reference. Specifically, utilizing dimension 8 as a reference, where each map expands by a factor of n, we aggregated the representations at intervals of n steps along the trajectory, computing the mean representation for these grouped states. This adjustment yields arrays of uniform length for each map, enabling us to then compute the Euclidean distance between the representation arrays of each map pair. The resulting distances can be seen in Table 1.

Map	**8x8**	**16x16**	**24x24**	**32x32**	**48x48**	**64x64**
8x8	0.00	51.67	71.00	69.59	68.08	65.82
16x16	51.67	0.00	19.32	17.91	16.41	14.15
24x24	71.00	19.32	0.00	1.40	2.91	5.17
32x32	69.59	17.91	1.40	0.00	1.51	3.76
48x48	68.08	16.41	2.91	1.51	0.00	2.25
64x64	65.82	14.15	5.17	3.76	2.25	0.00

Table 1: Euclidean distance between the representation of different map sizes. The table displays the pairwise Euclidean distances among all maps, with rows and columns representing the map sizes (8×8, 16×16, 24×24, 32×32, 48×48, and 64×64). This comparison highlights the similarity between representations of bigger maps and dissimilarity of smaller maps.

The results from our analysis reveal an intriguing pattern: the representations in larger maps demonstrated minimal variance, suggesting a high degree of similarity. The distance between the representation of maps with sizes 24×24 or larger are all less than 5.2, with some of them as low as 1.4. Conversely, representations derived from smaller maps exhibited a greater deviation when compared to others. The greatest deviation registered was between the maps 8×8 and 24×24, with a value of 71. All the distances

between the 8×8 and the other maps were bigger than 51, almost 10 times larger than what we observed between the larger maps. This indicates that as the map size increases, the model's ability to generate consistent representations improves, showcasing a robustness in handling spatially complex environments. On the other hand, the increased divergence seen in smaller maps underscores potential challenges in generalizing from limited spatial contexts to more expansive scenarios. This suggests that the spatial features learned from smaller maps may not encapsulate enough variability, leading to less robust representations when scaled to larger environments. To address this issue, enhanced adaptation strategies or modifications in model architecture may be necessary.

6.2. Policy Transferability

In this subsection, we investigate the transferability of our model's policy across different map sizes. We focused on comparing how well our model could transfer a policy learned on one scale to another. Initially, we trained the agent on the 8×8 map for 30,000 steps and evaluated its generated policy on every map. The results, shown in Figure 24, revealed a suboptimal transfer to the 16×16 map (which also happens in larger maps), which was unsurprising given the observed discrepancies in representations for smaller maps seen on the previous subsection. The policy learned on 8×8 allows the agent to reach the goal by traversing the shortest path every time, but when transfer to larger maps a lot was lost. Notably, the agent struggled to effectively apply a policy acquired in a small-scale environment to larger maps, resulting in seemingly random and ineffective behaviors.

To delve deeper into this phenomenon, we extended our investigation by conducting supplementary experiments. Here, we conducted additional experiments by training the agent on the 24×24 map and assessing its performance on larger maps. The resultant policies are illustrated in Figure 25. Remarkably, in this scenario, we observed a highly successful transfer, where most of the actions learned on the 24×24 map are transferred to the corresponding expected action on the the 48×48 map, indicating a robust ability of the model to generalize the policy from intermediate to larger spatial contexts.

These contrasting outcomes shed light on the nuanced dynamics of policy transfer across different scales, emphasizing the critical role of scale considerations in Reinforcement Learning. The failure of the 8x8-trained policy

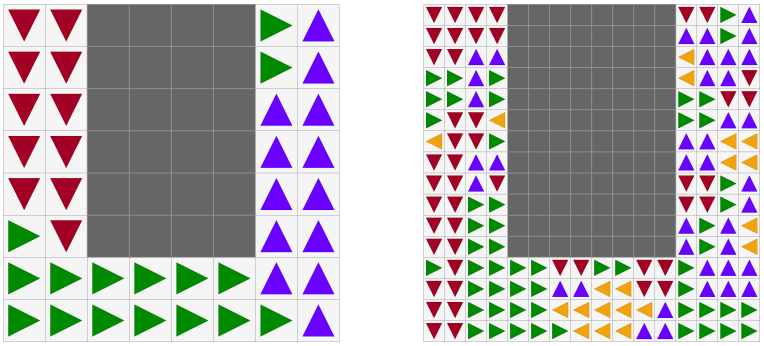

Figure 24: Agent policy transferability: on the left, the policy derived from training on the 8×8 map; on the right, the corresponding policy transferred to a 16×16 map. Significant discrepancies between the policies on each map are evident, indicating a suboptimal transfer.

to generalize to larger maps highlights the limitations of transferring knowledge acquired with small observation sizes to larger ones. On the other hand, the successful generalization from the 24x24 map to the 48x48 map underscores the importance of training on sufficiently complex environments. When trained on an intermediate-sized map, the agent encounters a diversity of spatial patterns and strategic challenges that better prepare it for even larger maps. This experience enables the model to develop more generalized and adaptable policies, capable of scaling to different sizes.

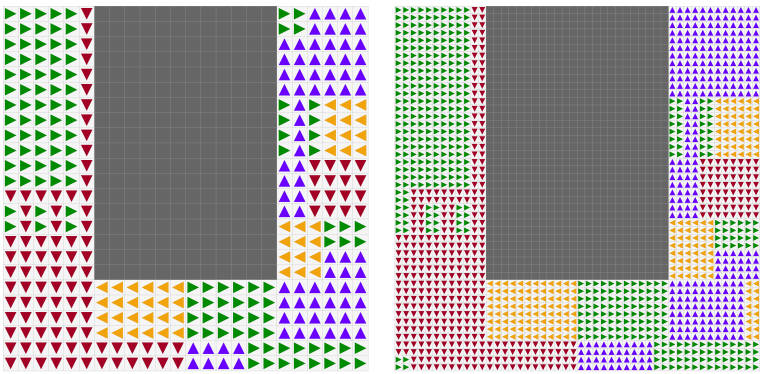

Figure 25: Agent policy transferability: on the left, the policy derived from training on the 24×24 map; on the right, the corresponding policy transferred for a 48×48 map. Despite the difference in map dimensions, the transferred policy closely matches the original policy, with only a few deviations.

6.3. Agent performance

It is important to note that our primary focus in this section's experiments is on the agent's ability to generalize its learning across different map scales rather than its specific performance metrics. Throughout our investigation, we observed a peculiar trend in our agent's performance. While it exhibited proficiency in learning the optimal policy within smaller maps (8×8 or smaller), it encountered significant challenges navigating larger maps to reach the goal. This issue was evident in the resulting policies from previous experiments: the agent trained on the 8x8 map successfully reached an optimal policy, whereas the agent trained on the 24x24 map struggled, failing to reach the goal consistently.

Interestingly, this issue isn't unique to our model; we conducted tests employing other architectures and algorithms from established libraries like Stable Baselines (Raffin et al., 2021), only to find even worse results. These models struggled to learn even in the context of smaller maps. This predicament appears to stem from an inherent incongruity between the simplicity of the environment and the design focus of Deep Reinforcement Learning models, which are typically geared towards addressing more complex tasks. This is particularly relevant for environments like Frozen Lake, where the simplicity of the task doesn't fully leverage the advanced capabilities of DRL algorithms.

Despite these performance challenges, the significance of our analysis in this section remains intact. Our primary objective was to develop a scale-invariant DRL model and evaluate its ability to generalize learning across different map sizes rather than evaluating DQN efficacy in such environments. The insights gained from these experiments are valuable, especially when considering more complex environments where our approach did not display the same performance issues.

7. Conclusion

In this study, we tackle the limitations of Reinforcement Learning (RL) frameworks that struggle when faced with varying sizes of state representations. We introduce a novel architecture that integrates Grid-wise Control with Spatial Pyramid Pooling, crafting a versatile model adept at learning from any grid-based setting without the need for structural modifications. This design allows for the smooth transfer of learned behaviors across similar environments. To enhance the model's adaptability, we've also devised a

novel training methodology that encompasses a variety of environments, each with unique state representation dimensions. This method offers the flexibility to adjust the focus on specific environments based on the requirements of the task at hand.

Our experiments conducted within the Gym-μRTS environment, demonstrate that our model not only outshines existing state-of-the-art solutions in terms of efficiency and generalization capabilities but also exhibits remarkable performance under constrained training conditions, outperforming baseline models in several instances. However, throughout our experiments, we encountered learning instabilities and generalization problems while working with small maps. Upon conducting an in-depth analysis within the Frozen Lake environment, we confirmed that our model struggles with very small maps, highlighting a significant area for improvement.

Looking ahead, our focus extends to addressing these shortcomings. We intend to explore alternative strategies to address the model's limitations in small maps, while simultaneously evaluating its effectiveness across a wider range of map sizes to ensure consistent performance. Furthermore, we are delving into alternative approaches for selecting environments within the adapted PPO algorithm, such as implementing a weighted random selection mechanism, to gain deeper insights into its impact on the model's learning trajectory.

In summary, our research has made significant strides in developing a scale-invariant RL model capable of generalizing across diverse scale settings. By addressing the identified limitations and refining our training methodologies, we aim to further enhance the model's robustness and applicability, paving the way for more versatile and effective RL solutions in complex and varied environments.

References

Andersen, P.A., Goodwin, M., Granmo, O.C., 2018. Deep RTS: a game environment for deep reinforcement learning in real-time strategy games, in: 2018 IEEE conference on computational intelligence and games (CIG), IEEE. pp. 1–8.

Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M., 2013. The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research 47, 253–279.

Bengio, Y., Louradour, J., Collobert, R., Weston, J., 2009. Curriculum learning, in: Proceedings of the 26th annual international conference on machine learning, pp. 41–48.

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al., 2019. Dota 2 with large scale deep reinforcement learning. arXiv preprint arXiv:1912.06680 .

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W., 2016. OpenAI Gym. arXiv preprint arXiv:1606.01540 .

Brody, S., Alon, U., Yahav, E., 2021. How attentive are graph attention networks? arXiv preprint arXiv:2105.14491 .

Girshick, R., Donahue, J., Darrell, T., Malik, J., 2014. Rich feature hierarchies for accurate object detection and semantic segmentation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 580–587.

Han, L., Sun, P., Du, Y., Xiong, J., Wang, Q., Sun, X., Liu, H., Zhang, T., 2019. Grid-wise control for multi-agent reinforcement learning in video game AI, in: International Conference on Machine Learning, PMLR. pp. 2576–2585.

He, K., Zhang, X., Ren, S., Sun, J., 2015. Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE transactions on pattern analysis and machine intelligence 37, 1904–1916.

Huang, S., Ontañón, S., 2022. A closer look at invalid action masking in policy gradient algorithms, in: Barták, R., Keshtkar, F., Franklin, M. (Eds.), Proceedings of the Thirty-Fifth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2022, Hutchinson Island, Jensen Beach, Florida, USA, May 15-18, 2022. doi:10.32473/flairs.v35i.130584.

Huang, S., Ontañón, S., Bamford, C., Grela, L., 2021. Gym-μrts: Toward affordable full game real-time strategy games research with deep reinforcement learning, in: 2021 IEEE Conference on Games (CoG), Copenhagen, Denmark, August 17-20, 2021, IEEE. pp. 1–8. doi:10.1109/CoG52621.2021.9619076.

Jankovics, V., Ortiz, M.G., Alonso, E., 2022. Efficient entity-based reinforcement learning. arXiv preprint arXiv:2206.02855 .

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2017. Imagenet classification with deep convolutional neural networks. Communications of the ACM 60, 84–90.

Lemos, M.L.H.D., Vieira, R.E.S., Tavares, A.R., Marcolino, L.S., Chaimowicz, L., 2024. Scale-invariant reinforcement learning in real-time strategy games, in: Proceedings of the 22nd Brazilian Symposium on Games and Digital Entertainment, Association for Computing Machinery, New York, NY, USA. p. 11–19. URL: https://doi.org/10.1145/3631085.3631337, doi:10.1145/3631085.3631337.

Liu, L., Liu, X., Gao, J., Chen, W., Han, J., 2020. Understanding the difficulty of training transformers. arXiv preprint arXiv:2004.08249 .

Mataric, M.J., 1994. Reward functions for accelerated learning, in: Machine Learning, Proceedings of the Eleventh International Conference.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 .

Pascanu, R., Mikolov, T., Bengio, Y., 2013. On the difficulty of training recurrent neural networks, in: International conference on machine learning, PMLR. pp. 1310–1318.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N., 2021. Stable-Baselines3: Reliable reinforcement learning implementations. J. Mach. Learn. Res. 22, 268:1–268:8.

Samvelyan, M., Rashid, T., De Witt, C.S., Farquhar, G., Nardelli, N., Rudner, T.G., Hung, C.M., Torr, P.H., Foerster, J., Whiteson, S., 2019. The starcraft multi-agent challenge. arXiv preprint arXiv:1902.04043 .

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. CoRR abs/1707.06347. URL: http://arxiv.org/abs/1707.06347, arXiv:1707.06347.

Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al., 2016. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489.

Tian, Y., Gong, Q., Shang, W., Wu, Y., Zitnick, C.L., 2017. Elf: An extensive, lightweight and flexible research platform for real-time strategy games. Advances in Neural Information Processing Systems 30.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. Advances in neural information processing systems 30.

Vinyals, O., Babuschkin, I., Czarnecki, W.M., Mathieu, M., Dudzik, A., Chung, J., Choi, D.H., Powell, R., Ewalds, T., Georgiev, P., et al., 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature 575, 350–354.

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A.S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., et al., 2017. Starcraft II: A new challenge for reinforcement learning. arXiv preprint arXiv:1708.04782 .

Wang, X., Song, J., Qi, P., Peng, P., Tang, Z., Zhang, W., Li, W., Pi, X., He, J., Gao, C., et al., 2021. SCC: an efficient deep reinforcement learning agent mastering the game of StarCraft II, in: International Conference on Machine Learning, PMLR. pp. 10905–10915.

Yun, W.J., Yi, S., Kim, J., 2021. Multi-agent deep reinforcement learning using attentive graph neural architectures for real-time strategy games, in: 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC), IEEE. pp. 2967–2972.