Scale-Invariant Reinforcement Learning in Real-Time Strategy Games

Marcelo Luiz Harry Diniz Lemos
Universidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais, Brazil
marcelolemos@dcc.ufmg.br

Ronaldo e Silva Vieira
Universidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais, Brazil
ronaldo.vieira@dcc.ufmg.br

Anderson Rocha Tavares
Universidade Federal do Rio Grande do Sul
Porto Alegre, Rio Grande do Sul
Brazil
artavares@inf.ufrgs.br

Leandro Soriano Marcolino
Lancaster University
Lancaster, United Kingdom
l.marcolino@lancaster.ac.uk

Luiz Chaimowicz
Universidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais, Brazil
chaimo@dcc.ufmg.br

ABSTRACT

Real-time strategy games present a significant challenge for artificial game-playing agents by combining several fundamental AI problems. Despite the difficulties, attempts to create autonomous agents using Deep Reinforcement Learning have been successful, with bots like AlphaStar beating even expert human players. Many RTS games include several distinct world maps with different dimensions, which may affect the agent's observation and the representation of game states. However, most current architectures suffer from fixed input sizes or require extensive and complex training. In this paper, we overcome these limitations by combining GridWise Control with Spatial Pyramid Pooling (SPP). Specifically, we employ the encoder-decoder framework provided by the GridNet architecture and enhance the critic component of PPO by adding an SPP layer to it. The new layer generates a standardized representation of any game state regardless of the initial observation dimensions, allowing the agent to act on any map. Our evaluation demonstrates that our proposed method improves the models' flexibility and provides a more effective and efficient solution for training autonomous agents in multiple RTS game scenarios.

CCS CONCEPTS

• **Computing methodologies** → **Sequential decision making**; **Neural networks**; • **Applied computing** → *Computer games*.

KEYWORDS

reinforcement learning, game-playing AI, RTS games

SBGames 2023, November 06–11, 2023, Brazil

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Marcelo Luiz Harry Diniz Lemos, Ronaldo e Silva Vieira, Anderson Rocha Tavares, Leandro Soriano Marcolino, and Luiz Chaimowicz. 2023. Scale-Invariant Reinforcement Learning in Real-Time Strategy Games. In *Proceedings of SBGames 2023.* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Games have become a popular testbed for Artificial Intelligence, providing a unique environment for researchers to develop and test new AI algorithms. Among these, Deep Reinforcement Learning (DRL) has shown remarkable advancements by using games as a research platform. Games like Atari titles [16], Go [20], StarCraft II [23], and Dota 2 [4] have been particularly effective in advancing DRL research, with RL-trained agents surpassing human experts in these games. The implications of these advancements go beyond the gaming world, with potential real-world applications in areas such as robotics, autonomous driving, and more.

However, DRL approaches often struggle with fixed input sizes and require retraining when working with different scales of input, even if the underlying environment has the same mechanics (e.g. training on the same game with different map sizes). Input resizing techniques alleviate the problem but require a preprocessing step and the definition of the neural network input size beforehand, which might not properly fit in unforeseen data (e.g. the predefined input is too small, and resizing a large input causes loss of information).

To address this challenge, we propose a new architecture that uses Spatial Pyramid Pooling (SPP) [9] to create a scale-invariant DRL architecture that can be applied to any DRL domain. SPP is a technique commonly used in scale-invariant image recognition tasks, and we adapt it to work with DRL environments. Our approach enables the same neural network to play a game with different input (map) sizes, greatly simplifying and speeding up the training of DRL agents. Additionally, our approach allows for quick transfer learning, as the agent can seamlessly handle inputs with unforeseen sizes. Moreover, using multiple map sizes in a single training session enriches the agent's training and improves its generalization ability.

We evaluate our approach in the Gym-μRTS framework [11]. It provides a research-friendly real-time strategy game platform while retaining the main challenges of the domain: a huge state space, a combinatorial action space, and the necessity of micromanaging resources, controlling unit production, building structures, and engaging in combat. The framework uses grids to represent the world map, favoring the application of grid-wise control [8], where actions are sent to every position on the map.

Our results indicate that our novel architecture improves performance in single and multi-environment tasks, surpassing other state-of-the-art agents. It also shows that using a more diverse training comprised of multiple environments can lead to agents performing well in various scenarios instead of being highly specific.

2 RELATED WORK

Up to this day, the most successful attempt to develop an autonomous agent for a commercial RTS was DeepMind's AlphaStar [23], which was able to beat top human players. This was a singular achievement, mainly possible due to the extensive computational infrastructure available combined with long training sessions. Other teams have tried similar approaches but also used thousands of CPUs and GPUs [25].

Many researchers have resorted to less hardware-intensive platforms, which downsize different aspects of RTS games. For example, μRTS [11] is a grid-based framework developed in Java with simpler dynamics than a commercial RTS; miniRTS [21] is a lightweight RTS that is part of the ELF project and has an RL backend; and Deep RTS [1] is a fast, highly configurable, and complex RTS platform developed for DRL research. Another approach is to isolate parts of the game to focus on specific challenges, as seen on the StarCraft II Learning Environment (SC2LE) mini-games [24].

One of the main difficulties most players face in RTS games is the ever-changing number of units in play. Since there is no way to reliably predict how many units a player will have at any given time, players must be capable of micromanaging an arbitrary number of units at all times. Han *et al.* [8] developed a technique that minimizes this problem by splitting the game's map into a grid and issuing actions per cell. They applied the Encoder-Decoder architecture used in image segmentation tasks to select actions for every grid cell, creating a grid-wise control that disregards the number of units in play.

Proximal Policy Optimization (PPO) algorithms [19] are actor-critic models that have performed well in several domains and were also used in μRTS to great avail. By combining PPO with grid-wise control and masking invalid actions [10], Huang *et al.* [11] were able to develop an agent capable of beating state-of-the-art opponents in a very efficient manner. However, the proposed model lacks flexibility: the network architecture must change if the map size varies. While the encoder-decoder used for the actor is capable of handling any map, the critic needs a fixed observation space as input because of its fully connected architecture. Consequently, the agent requires hard-coded layer sizes and must be trained from scratch whenever a different map dimension is selected.

Attention mechanisms can be used in multi-agent control with heterogeneous observation spaces. For example, a Transformer-based approach [22] was used in multi-agent credit assignment and joint action evaluation [13] on the Starcraft Multi-Agent Challenge [18]. While they have achieved significant success in several domains, Transformers suffer from vanishing and exploding gradients [17] and strong dependencies on residual branches [15].

Another option to deal with varying input sizes is to change the observation model. Fixed-size representations may lead to a waste of memory and processing in some settings. Since most environment information is tied to entities, entity-based representations can be very efficient in sparse domains. Graph Attention Networks (GATs) [6] were used in single agent control using an entity-based approach on the Arcade Learning Environment (ALE) [2] and Simple Playgrounds [12]. GATs were also used in Multi-Agent Reinforcement Learning to tackle Starcraft mini-games [26] on a decentralized approach. However, they were tested on specific tests rather than entire RTS matches.

In our approach, we combine Grid-Wise Control with Spatial Pyramid Pooling to create a novel scale-invariant architecture for control problems. The result is a flexible and efficient agent that can control an arbitrary number of units in distinct state representations without structural changes. The agent developed by Huang *et al.* [11] was used as a baseline for our work, and we take advantage of some of the techniques used by them, including Invalid Action Masking [10] and Action Composition. We also apply Reward Shaping, a technique that incorporates external knowledge into RL problems by designing a set of small rewards for actions that can help the agent quickly learn how to reach the final goal.

Our proposed model includes two major improvements over the baseline. First, we introduce a new network architecture that includes Spatial Pyramid Pooling layers, producing a flexible scale-invariant network. Second, we implement a new training procedure that collects experience from maps with different dimensions – and, therefore, different representation sizes – producing a more diverse and robust learning process.

3 BACKGROUND

3.1 Grid-Wise Control

Grid-wise Control [8] is a technique used to control an arbitrary number of entities in environments that can be divided into a grid. The architecture employed – named GridNet – applies an encoder-decoder combination very similar to other architectures seen in image segmentation tasks to select an action for each cell of the observation grid. Considering a grid with dimensions (h, w, n_f), where (h, w) represents the scale of the grid and n_f is the number of feature planes, GridNet takes a state $s \in \mathbb{R}^{h \times w \times n_f}$ as input and predicts an action a_{ij} for each grid position (i, j) with $1 \leq i \leq h$, $1 \leq j \leq w$. The resulting output is defined as an action map a with dimensions (h, w, c_a), where c_a indicates the action dimension of the agents.

In RTS games, actions are usually defined by a combination of parameters, such as ordering a unit to build a base at position (x, y). In this case, the action type would be *build*, the structure type would be *base*, and the target location would comprise coordinates x and y. As a result, the dimension c_a must contain all the different

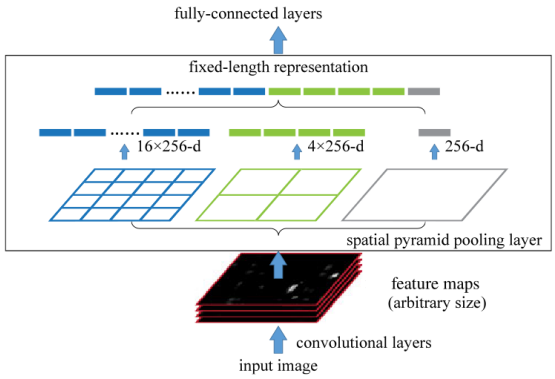

Figure 1: An example of Spatial Pyramid Pooling, assuming that the latest convolution layer yields 256 feature maps. The first SPP level has 16 bins of dimension 256, the second level has 4 bins and the third level has 1 bin. The final representation given by SPP has length $(16 + 4 + 1) \times 256$ **regardless of the input. Source: [9].**

parameters necessary to predict and compose any action and its specifications.

3.2 Spatial Pyramid Pooling

Convolutional Neural Networks (CNNs) have revolutionized Deep Learning, especially in the computer vision field. Still, most popular CNN architectures require fixed input sizes due to the fully connected layers. Nonetheless, real-world applications tend to encompass vastly heterogeneous data with images of varying sizes, which may be incompatible with CNNs by default. To circumvent this problem, many models use preprocessing routines – such as cropping [14] or warping [7] – to fit the images to the input size. In practice, neither cropping nor warping are ideal solutions as they may result in loss of content or geometrical distortion, reducing model accuracy and compromising recognition in some cases.

A more effective alternative is Spatial Pyramid Pooling (SPP) [9], a computer vision method that solves the problem by generating a fixed-length representation regardless of input dimensions. SPP uses multi-level pooling over spatial bins without causing distortions in the output. Like most pooling techniques, it uses filters to generate a summarized representation from a given feature map. However, unlike most pooling, the filter size is variable and dependent on the input size, while a predefined number of bins generate a constant-shape representation. As the name implies, SPP can also combine several filters, creating a pyramid-like structure. The result of each layer of the SPP is combined into a single vector that always has the same size, regardless of the input size. An example can be seen in Figure 1.

3.3 Gym-μRTS

μRTS is a simplified RTS game for two players designed for AI research. Games occur in a rectangular $h \times w$ grid where each cell may contain up to one unit, building or resource at any given time. Map size and configuration can be easily customized before the game starts, providing many scenarios to test AI agents. Despite not being as complex as most commercial games of the genre, μRTS still retains RTS's core features and challenges.

Gym-μRTS is a Reinforcement Learning interface for μRTS developed with OpenAI Gym [5] that facilitates the integration of μRTS with many popular machine learning libraries. Its observation space is represented by a tensor (h, w, n_f), where h and w are the map's dimensions, and n_f is the number of binary feature planes. Since the observation shape is coupled to the map's dimensions, agents developed for Gym-μRTS must work with an arbitrary space shape to play any possible scenario.

4 METHODOLOGY

4.1 Model Architecture

As shown in Figure 2, the model architecture is divided into three parts: one encoder and two decoders. The encoder is composed of convolutional layers followed by pooling layers; It takes the agent's observation as input and generates a concise feature map that can be better used to predict the policy the agent should follow in the given environment state, as well as the state's value. While the baseline model encoder encompasses four pairs of convolutional and pooling layers, we noticed that downsizing it to two pairs of layers maintained the agent's performance but significantly reduced the number of parameters on the network.

The encoder output is used by the first decoder – the actor's decoder – to generate the probabilities of each action for each cell of the grid. 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. This new layer generates a standardized representation of any input passed to the critic, which allows 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.

Details about the architecture, including all layers and their sizes can be found on our GitHub repository[1].

4.2 Expansion of Training Scenarios

The new model's capabilities allow a distinct training approach. We can leverage the agent's expertise in one scenario to accelerate its learning process in other similar settings. We take this idea one step further and utilize multiple scenarios in a single training set to

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

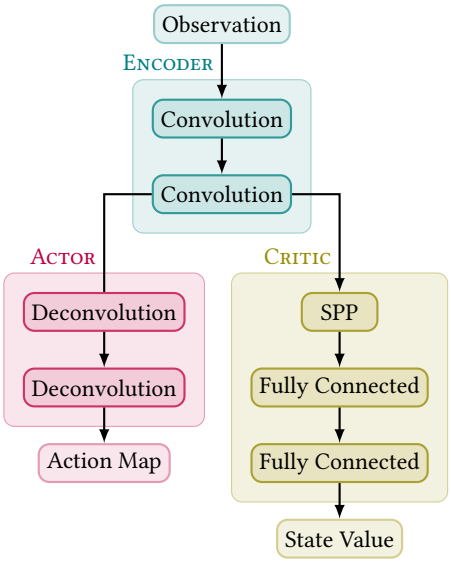

Figure 2: Network Architecture.

develop a more general agent that learns to perform well in distinct environments.

We modified PPO – shown in Algorithm 1 – to enable this 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.

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 [3], where the agent starts playing only in simple maps and, as time passes, more complex maps are added to the selection pool.

5 EXPERIMENTAL RESULTS

We verified our model's efficacy in several experiments on Gym-μRTS, validating its performance against state-of-the-art agents. Once again, we follow the setup employed by Huang *et al.* [11],

Algorithm 1 PPO with environment swap

```
for iteration = 1, 2, ... do
    Select environment E
    for actor = 1, 2, ... do
        Run policy π_θold in environment E for T timesteps
        Compute advantage estimates Â_1, ..., Â_T
    end for
    Optimize surrogate L wrt θ, with K epochs and minibatch size M
    θ_old ← θ
end for
```

designating CoacAI[2] as the main opponent in our experiments and using 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 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[3] developed in [11] 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 4x4 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 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[4].

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

[2]Available at https://github.com/Coac/coac-ai-microrts
[3]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.
[4]Available at https://github.com/marcelo-lemos/MicroRTS-Py

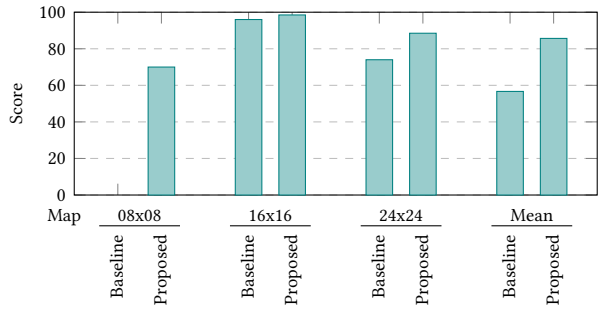

Figure 3: Proposed vs. Reference model - Score - Specialized setting.

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.* [11] 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 3, 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. The baseline model architecture and hyperparameters were designed for the 16×16 map, which may cause it to function improperly on settings that require different strategies. Both the 16×16 and 24×24 maps are big, and the strategies that work best on them involve developing more military structures and combat units. Meanwhile, the 8×8 map is smaller than the other two and favors quicker strategies, such as creating the maximum amount of simple workers and sending them to attack the enemy as soon as they are ready.

Figures 4 and 5 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, where the original model had trouble learning how to play the game effectively. After 100 million steps, its performance dropped, and it could not recover. Our novel architecture does not impair performance when focusing on a single environment and can even improve the results. We also notice that the episodic returns differ from one map to another mainly because of the reward shaping we use. In larger maps, units must travel greater distances, causing games to last longer and allowing the agent to get more rewards from small actions.

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.

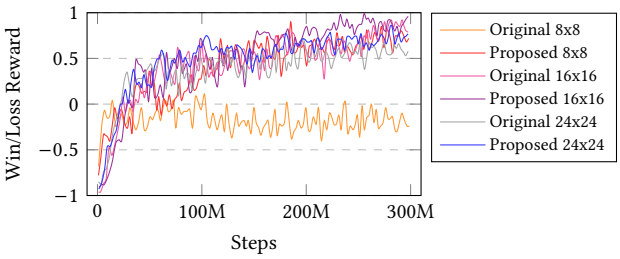

Figure 4: Proposed vs. Reference model - Win/Loss Reward - Specialized setting.

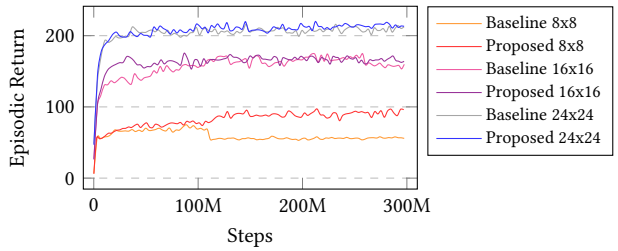

Figure 5: Proposed vs. Reference model - Episodic Return - Specialized setting.

Figure 6 shows that the specialized agents of the baseline model performed better than our generalist agent in two of the three maps. On 16×16, the baseline received 36.5 more points than the proposed model, but on the 24×24, the difference was a lot smaller, with the baseline being only 3.5 points better. 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.

One 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. Even with the heavily reduced budget, we observe a significant performance improvement.

Figure 7 shows that the win/loss reward received by the generalist agent is very similar to the baseline on maps 16×16 and 24×24, while Figure 8 shows that the generalist agent's episodic returns are biased towards the 8×8 baseline. The generalist is trained on all three maps for the same number of steps. Since the 8×8 map is small and games are short, the agent completes more episodes on it, skewing the curve towards the 8×8 baseline.

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. Despite the different training, they were all evaluated on all

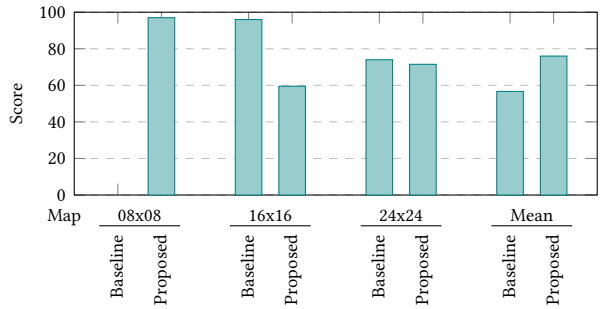

Figure 6: Proposed vs. Reference Model - Score - Generalist setting.

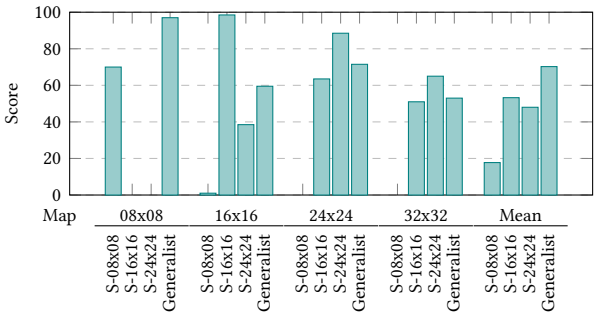

Figure 9: Specialized vs Generalist - Score.

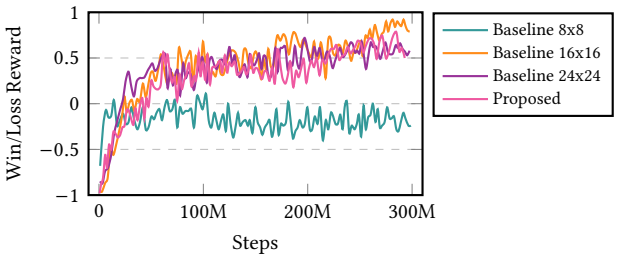

Figure 7: Proposed vs. Reference model - Episodic returns - Generalist setting.

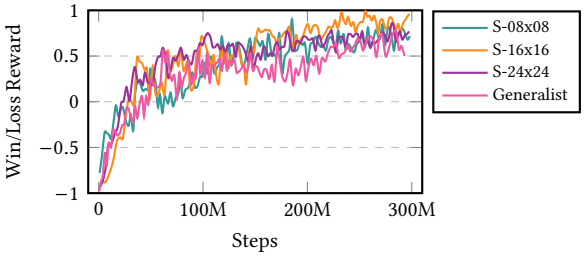

Figure 10: Specialized vs Generalist - Win/Loss reward.

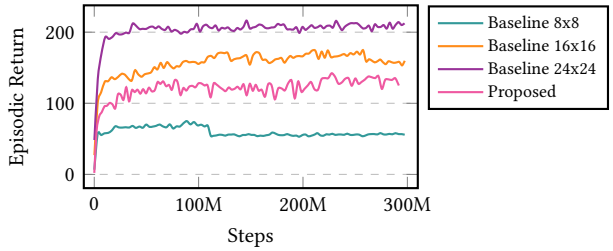

Figure 8: Proposed vs. Reference model - Episodic returns - Generalist setting.

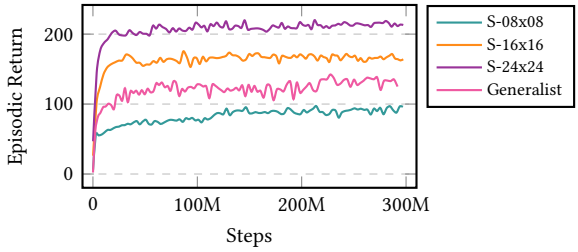

Figure 11: Specialized vs Generalist - Episodic Return.

three maps and we also included a 32×32 map in the evaluation that was not included in any agent's training.

From now onward, we refer to each agent as S-08x08, S-16x16, and S-24x24 for the agents trained solely on maps 8×8, 16×16, and 24×24, respectively, and *generalist* for the agent trained on all maps.

Figure 9 shows that the specialized training performed better than the generalist agent in two of the three maps. On 16×16, the S-16x16 received 39 more points than the generalist, but on the 24×24, the difference was a lot smaller, with the S-24x24 being only 17 points better. On the 8×8 map, the generalist achieved 97 points against 70 of the S-08x08. 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.

Both the win/loss reward and the episodic return – seen in Figures 10 and 11, respectively – were similar to the ones of the previous experiment. The only major difference was the S-08x08 agent, which, unlike the baseline model, managed to attain good results on map 8×8 due to the flexibility of our proposed model.

5.3 Environment Selection

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

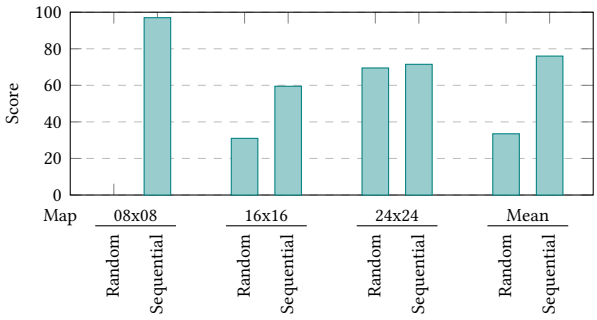

Figure 12: Random vs sequential selection - Score.

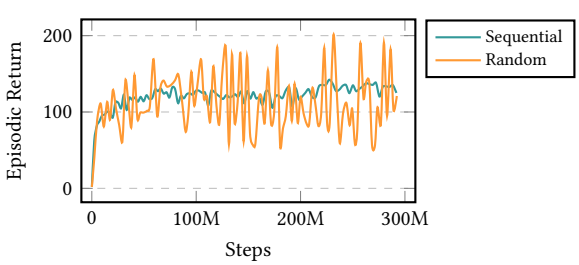

Figure 13: Random vs Sequential selection - Episodic Return.

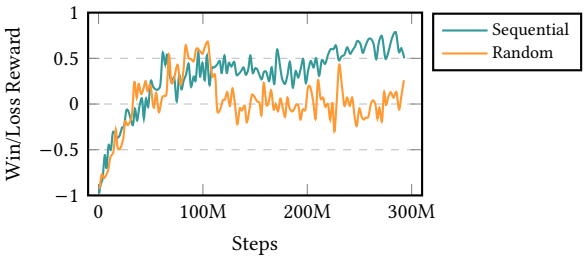

Figure 14: Random vs sequential selection - Win/Loss reward.

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.

As seen in Figure 12, the sequential selection outperformed random in all three maps tested, particularly on map 8×8, where the random method lost all 100 games, whereas the sequential won 97. The random method's episodic return, shown in Figure 13, reveals that the random selection is much less stable than the sequential one, which is directly related to the reduction of the win/loss reward – seen in Figure 14 – and the resulting policy. Sequential selection results in better and smoother learning processes.

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

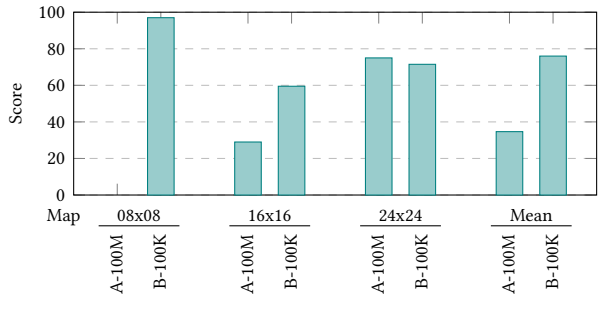

Figure 15: Environment swap frequency - agents' score.

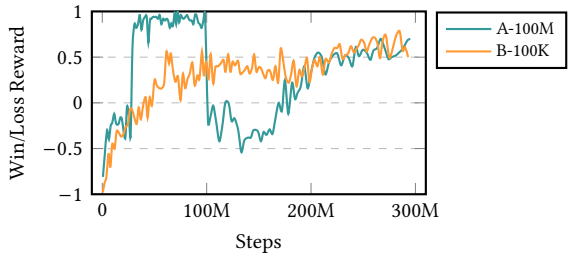

Figure 16: Environment swap frequency - win/loss reward.

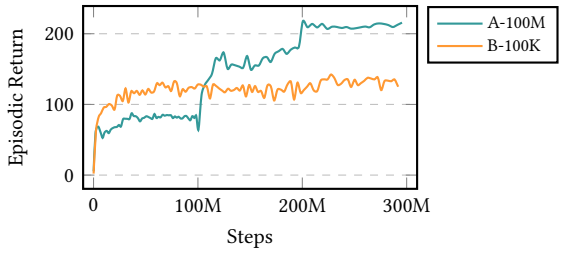

Figure 17: Environment swap frequency - episodic return.

changing every 100,000 steps. We refer to them as A-100M and B-100K, respectively.

As seen in Figure 15, the agent B-100K, trained with more frequent swaps, achieved better results on the test games on most maps, except for the 24×24 map, where the score was 3.5 points below A-100M. Figure 16 shows that the agent that trained for longer periods before swapping environments presented a big drop in performance during the change of context. The agent specialized in a single map after training for a long period on it, but when the environment changed, it took some time to adapt the knowledge acquired to the new situations. Figure 17 shows the episodic return, where we can see when the map changes occur for the A-100M agent by the steps created. Meanwhile, changing maps more frequently led to smooth and consistent learning.

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)

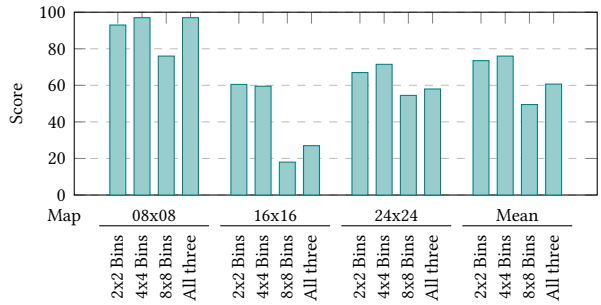

Figure 18: SPP Layer Size - Score.

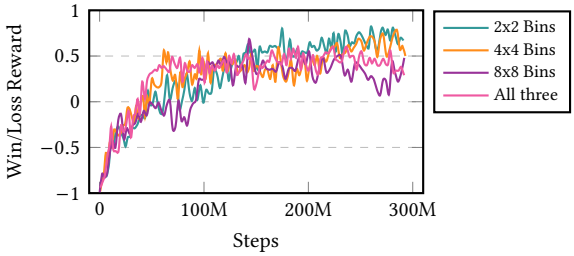

Figure 19: SPP Layer Size - Win/Loss reward.

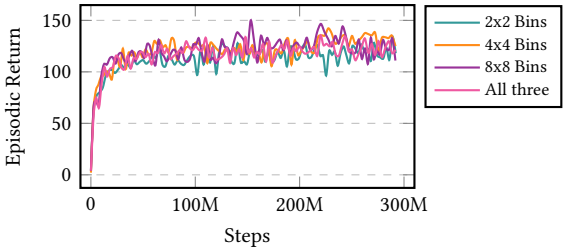

Figure 20: SPP Layer Size - Episodic return.

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.

As shown in Figure 18, 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 19 and 20 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.

6 CONCLUSION

In this paper, we address the challenge of Reinforcement Learning architectures that struggle to adapt to varying state representation sizes. We propose a novel architecture that combines Grid-wise Control and Spatial Pyramid Pooling to create a flexible model capable of acquiring knowledge from any grid-like environment without requiring structural changes. Consequently, this model can seamlessly transfer knowledge from one environment to similar ones. Additionally, we have developed a new training procedure that involves multiple environments with distinct state representation sizes. This approach can be tailored to emphasize certain environments more than others, depending on the situation's needs.

The proposed architecture and algorithm were evaluated using the Gym-μRTS framework. Our results show that our new agent outperforms other state-of-the-art models, achieving superior results with greater efficiency and generalization. Furthermore, even in scenarios where our agent was at a disadvantage, with a reduced training budget, it surpassed the baseline agents in some cases.

As for future work, we plan to expand our map roster to see if the agent's performance can remain high across an arbitrary number of maps. We also plan on investigating other strategies for PPO's environment selection – such as a weighted random selection – and their impact on the learning progress.

ACKNOWLEDGEMENTS

This work was partially supported by CAPES (finance code 001 and PRINT program), CNPq (grant 311900/2020-8), and Fapemig (grant PPM-00563-18).

REFERENCES

[1] Per-Arne Andersen, Morten Goodwin, and Ole-Christoffer Granmo. 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, 1–8.

[2] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47 (2013), 253–279.

[3] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*. 41–48.

[4] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680* (2019).

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[6] Shaked Brody, Uri Alon, and Eran Yahav. 2021. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491* (2021).

[7] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 580–587.

[8] Lei Han, Peng Sun, Yali Du, Jiechao Xiong, Qing Wang, Xinghai Sun, Han Liu, and Tong Zhang. 2019. Grid-wise control for multi-agent reinforcement learning in video game AI. In *International Conference on Machine Learning*. PMLR, 2576–2585.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence* 37, 9 (2015), 1904–1916.

[10] Shengyi Huang and Santiago Ontañón. 2022. A Closer Look at Invalid Action Masking in Policy Gradient Algorithms. In *Proceedings of the Thirty-Fifth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2022, Hutchinson Island, Jensen Beach, Florida, USA, May 15-18, 2022*, Roman Barták, Fazel Keshtkar, and Michael Franklin (Eds.). https://doi.org/10.32473/flairs.v35i.130584

[11] Shengyi Huang, Santiago Ontañón, Chris Bamford, and Lukasz Grela. 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, 1–8. https://doi.org/10.1109/CoG52621.2021.9619076

[12] Vince Jankovics, Michael Garcia Ortiz, and Eduardo Alonso. 2022. Efficient entity-based reinforcement learning. *arXiv preprint arXiv:2206.02855* (2022).

[13] Muhammad Junaid Khan, Syed Hammad Ahmed, and Gita Sukthankar. 2022. Transformer-Based Value Function Decomposition for Cooperative Multi-Agent Reinforcement Learning in StarCraft. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 18. 113–119.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2017. Imagenet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90.

[15] Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. 2020. Understanding the difficulty of training transformers. *arXiv preprint arXiv:2004.08249* (2020).

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[17] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International conference on machine learning*. PMLR, 1310–1318.

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

[19] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017). arXiv:1707.06347 http://arxiv.org/abs/1707.06347

[20] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[21] Yuandong Tian, Qucheng Gong, Wenling Shang, Yuxin Wu, and C Lawrence Zitnick. 2017. Elf: An extensive, lightweight and flexible research platform for real-time strategy games. *Advances in Neural Information Processing Systems* 30 (2017).

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

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

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

[25] Xiangjun Wang, Junxiao Song, Penghui Qi, Peng Peng, Zhenkun Tang, Wei Zhang, Weimin Li, Xiongjun Pi, Jujie He, Chao Gao, et al. 2021. SCC: an efficient deep reinforcement learning agent mastering the game of StarCraft II. In *International Conference on Machine Learning*. PMLR, 10905–10915.

[26] Won Joon Yun, Sungwon Yi, and Joongheon Kim. 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, 2967–2972.