Growing Up Together: Structured Exploration for Large Action Spaces

Gabriel Synnaeve[*]
Facebook AI Research

Jonas Gehring[*]
Facebook AI Research

Zeming Lin[*†]
New York University

Daniel Haziza[*]
Facebook AI Research

Nicolas Carion[†]
New York University

Daniel Gant[†]
Independent

Laura Gustafson
Facebook AI Research

Dexter Ju
Facebook AI Research

Vegard Mella
Facebook AI Research

Danielle Rothermel
Facebook AI Research

Nicolas Usunier
Facebook AI Research

Abstract

Training good policies for large combinatorial action spaces is onerous and usually tackled with imitation learning, curriculum learning, or reward shaping. In this paper, we propose another method that addresses this problem: growing the action space of the policy during training can assist exploration and speed up convergence without any external data (imitation), with less control over the environment (curriculum), and with minimal or no reward shaping. We evaluate this approach on a challenging end-to-end full games army control task in StarCraft®: Brood War®[‡] by training policies through self-play from scratch. We produce a strong end-to-end baseline for this environment and task. We compare it to growing the spatial resolution and frequency of actions, which results in faster learning.

1 Introduction

Deep reinforcement learning's success stems in part from the handling of large state spaces via powerful function approximation. But how can training algorithms cope with large, high-dimensional or combinatorial action spaces? Approaching domains with larger action spaces over longer horizons leads to an explosion of the space of trajectories. Numerous methods exist to structure exploration in such environments: imitation and inverse reinforcement learning [37, 30], curriculum learning [39, 11, 4], reward shaping [29], options [42], or hierarchical reinforcement learning [10]. Each technique respectively requires expert traces or demonstrations, modifications of the task or environment, prior knowledge on the value structure, on the action structure, or on the state structure. These requirements limit general and broad application.

We study an approach that leverages prior knowledge on the action and policy space to greatly reduce the sample complexity of exploration: growing the action space (GAS) of a policy during its training. Recently proposed by Farquhar et al. [13], GAS does not require demonstrations, nor modifying the

[*]equal contribution, correspondence to gabriel.synnaeve@gmail.com

[†]Work done while at Facebook AI Research. Work done in 2019.

[‡]StarCraft is a trademark or registered trademark of Blizzard Entertainment, Inc., in the U.S. and/or other countries. Nothing in this paper should be construed as approval, endorsement, or sponsorship by Blizzard Entertainment, Inc.

environment or tuning reward functions, and the induced hierarchy of actions remains actionable. Instead, it requires (i) a form of hierarchy or abstraction over actions, (ii) a representation of different levels of detail within a single model, (iii) a training procedure and growth schedule. In this work, we address (i) by leveraging the inherent spatial and temporal hierarchy in the environment we consider. We start by learning a policy at coarse levels of detail, i.e. at low resolutions and by taking actions at low frequency. This forms the basis of the policy computation at finer levels of detail, and we increase the action resolution as training progresses. We represent the policy (ii) as a ConvNet [22] encoder-decoder with LSTMs [18], decoding directly to the finest action resolution and obtaining coarser actions via spatial pooling. Finally, we address (iii) by hyper-optimizing the action space growth schedule with Population Based Training [19]. This way, action space refinements are directly tested on their ability to improve performance, and the result is a natural curriculum dictated by the policy's current abilities.

We demonstrate the efficacy of GAS with an experimental study on full games of StarCraft: Brood War, a partially observable real-time strategy game. Players can control hundreds of units executing simultaneous, durative actions in games averaging 15-20 minutes. While the state space is large, the major exploration issue in StarCraft stems from a combinatorial action space. The game engine runs at 24 frames per second and a player can potentially issue a different action at each frame. Actions in StarCraft are the combination of a set of units executing a single command type, optionally targeted at a specific position. The lower bound of the branching factor b, the number of actions available at any point in time, is 10^{50} [32], and a game spanning 15 minutes with one action per second corresponds to a depth d of 900. The game complexity is thus $b^d = (10^{50})^{900}$. We consider a setting of military unit control with three action types and a target resolution of 64×64; still, the number of possible actions per step for an average of 80 live units is $2^{80} \times 64 \times 64 \times 3 \approx 1.5 \times 10^{28}$.

Our primary motivation for applying GAS to StarCraft is that a human player of average skill can play strategic full games with 30 effective actions per minute, with imprecise clicks and timings, and by grouping units together. We thus assume that it is more important to first master those broad strokes before filling in the details of optimal micromanagement. Starting from random initialization, our policy neural network initially takes one action every 10 seconds and at an 8×8 target resolution. For unit selection, we implement grouping by spatially correlated sampling, favoring the consideration of a single group only at the beginning of training and eventually requiring the selection of individual units. For a game of 15 minutes, the number of possible action sequences grows from $(2^1 \times 8 \times 8 \times 3)^{90} = 384^{90}$ to $(1.5 \times 10^{28})^{900}$ over the course of training.

Our contributions are as follows. We propose a spatio-temporal action space resolution decomposition in the context of real-time strategy games and realize it with a single parameterization, so that growing the action space of a trained model minimizes negative effects on performance. We describe a large-scale self-play training setup in which we, to the best of our knowledge, train the first end-to-end model for military units control in full games of StarCraft with self-play, from scratch. Finally, we showcase the potency of GAS as an exploration curriculum achieving faster learning and equal (in full games) or higher (in scenarios) performance compared to acting purely at the finest resolution. Our training setup is available as open-source code at `https://github.com/TorchCraft/TorchCraftAI`.

2 Related Work

Curriculum learning The motivation for curriculum learning is to present a learning algorithm with data of increasing difficulty. This approach is particularly appealing in the context of neural networks as their standard optimization algorithm, (stochastic) gradient descent, is already incremental in nature [11, 4]. For reinforcement learning, curricula are commonly defined over tasks [39]. They rely foremost on a suitable environment that allows the definition of such tasks, are problem-specific and require careful tuning. One approach to automate the definition of tasks is reverse curriculum learning, in which initial states are generated with gradually increasing distance from goal states [14]. In adversarial settings such as games, self-play [35, 50] represents a natural form of curriculum as agents face opponents of the same or a similar ability [3]. Its effectiveness is showcased in recent studies, solving a variety of tasks [20, 41, 1, 43].

Hierarchical reinforcement learning and options A general approach for tackling challenging tasks is to exploit inherent structure in the domain of interest. In reinforcement learning this can be achieved by hierarchical decision-making, with the options framework representing a popular

formalization [45]. Policies operating at different levels of temporal abstraction promise faster learning and better transfer between tasks, and various works propose methods for automatic option discovery [25, 42, 24]. A possible alternative to the explicit specification of options is to tailor the design of neural network policies to yield hierarchical behavior [9, 53]. Other approaches concern themselves with structure in state or observation spaces. With a suitable state abstraction and proper aggregation, exploration can be greatly accelerated [31]. However, finding the right abstraction is critical, and one can either settle for a static one or begin with a coarse state space that is refined over the course of learning [26, 27]. Irrespective of the method selected, sufficient domain knowledge is required.

Large action spaces If the number of possible actions is very large, e.g. in combinatorial action spaces, sufficient exploration of their effects is a challenging endeavor. To this end, Farquhar et al. [13] introduce the concept of growing action spaces on a multi-agent micro-management scenarios in StarCraft. Increasing the action space from few or one to several groups of units induces a curriculum as the learning agent obtains successively finer controls. We extend this approach with both a spatial and temporal action hierarchy, add hierarchical structure to policy instead of value models and apply it to unit control in full StarCraft games. A similar approach is utilized by Murali et al. [28] with a curriculum over dimensions of control of a robotic arm. In this case, the curriculum prescribes a sampling strategy for the dimensions in order to guide exploration. Czarnecki et al. [8] propose a curriculum over agents by mixing two policies defined on a coarse and fine action space, respectively. Our work differs in that it integrates action hierarchies with multiple levels into a single policy parameterization.

Reinforcement learning in real-time strategy games Various works concern themselves with unit control in isolated StarCraft micro-management scenarios via reinforcement learning [46, 7, 52, 13, 5]. Here, we focus on control of military units in full games for which the majority of past approaches employed classical search techniques. Recently, large-scale self-play setups produced high-performing agents in Dota and StarCraft II, two popular real-time strategy games. OpenAI [33] train separate policy networks to control the five individual units for Dota and beat professional teams. Their final agents accumulated 45,000 years of self-play experience during training with a manually tuned, dense reward function. Vinyals et al. [54] reach Grandmaster skill level by competing against human players on public StarCraft II game servers. Their model is initialized from 971,000 human game replays via imitation learning and then optimized via Population Based Training, with distinct types of agents and prioritized fictitious self-play to ensure continued learning progress. Both works highlight the difficulty of exploration in such complex environments. We limit our experiments to military unit control but train our policies from scratch, use minimal reward shaping only, and employ common ranking schemes for population management.

3 Growing Action Spaces

We present a formalisation of GAS for Markov Decision Processes (MDPs) defined by a tuple $\mathcal{M} \coloneqq \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ a finite set of actions, $\mathcal{P}$ the transition probability between states conditioned on actions, $\mathcal{R}$ the reward function for reaching a state through an action, and with a reward discount factor γ. The goal is find an optimal policy for $\mathcal{M}$ denoted π^* that maximizes the discounted sum of rewards, or return. For an action set $\mathcal{A}' \subset \mathcal{A}$ only a subset of states $\mathcal{S}' \subseteq \mathcal{S}$ might be reachable and hence only a subset of the codomain of $\mathcal{R}$. By a slight abuse of notation, let $\mathcal{R}'$ be reward function induced by restricting actions to $\mathcal{A}'$. It is often possible to extract a smaller MDP $\mathcal{M}'$, in the sense of consisting of fewer transitions and smaller sample complexity, from a given MDP $\mathcal{M}$. To the extent that $\mathcal{R}'$ samples salient values of $\mathcal{R}$, $\mathcal{M}'$ may be helpful for structuring and accelerating the exploration of the original MDP $\mathcal{M}$.

Let $\pi^{*\prime}$ (resp. π^*) be the optimal policy in $\mathcal{M}'$ (resp. $\mathcal{M}$). Since $\mathcal{A}' \subset \mathcal{A}$, $\pi^{*\prime}$ is a valid policy in $\mathcal{M}$. In general there is no guarantee that $\pi^{*\prime}$ is optimal in $\mathcal{M}$, and it is possible that $D(\pi^{*\prime}, \pi^*) \to \infty$ for some notion of distance D between policies. Consequently, the application of GAS assumes that $\mathcal{M}'$ can be used as a proxy for exploration or as a curriculum for $\mathcal{M}$. To avoid situations in which the transfer from $\pi^{*\prime}$ to π^* complicates exploration in $\mathcal{M}$, it may be beneficial to not reach the optimal policy in $\mathcal{M}'$. This can be achieved by mixing action spaces, i.e. training π and π' simultaneously [8, 13], or via entropy regularization. We note that in above formulation, there is no requirement for representations of π' and π to share parameters. In order to illustrate the generality of

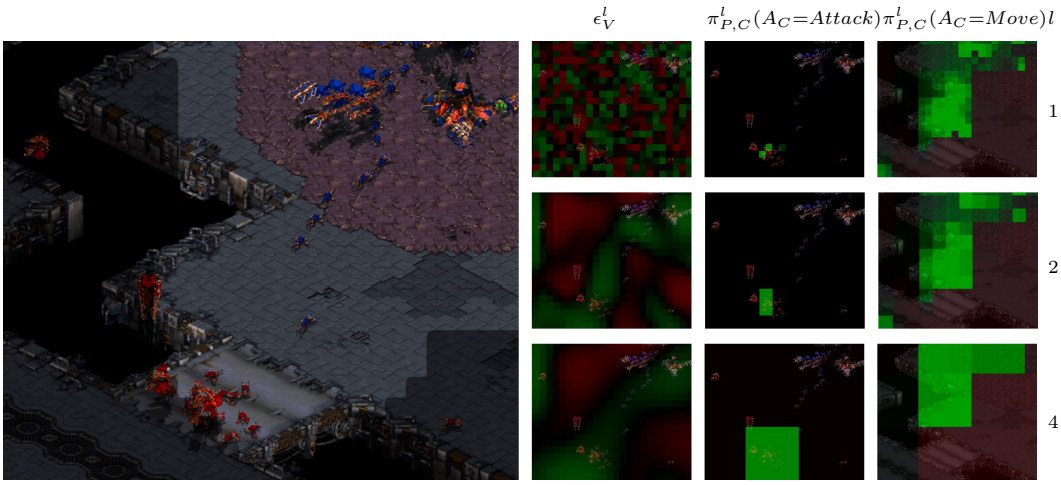


Figure 1: Visualization of spatially correlated noise and action space resolution. **Left** Close-up of a StarCraft: Brood War game with two players (blue and red), representing approximately 12% of the whole map area. The shaded area corresponds to the "fog of war" limiting the blue player's observations. **Right** For the agent playing blue, we show (from left to right) the sampling noise for unit selection ϵ_u^l, example distributions over positions for $Attack$ and $Move$ commands at different spatial levels l (top to bottom). Green represents high values, red stands for low values; actual values are scaled for visualization purposes.

GAS, we provide further categories and examples in Appendix A; the following sections will focus on its application to real-time strategy games.

4 GAS Modeling for Real-time Strategy Games

4.1 Task

We consider the problem of military unit control in StarCraft: Brood War in full game-play, which also requires management of worker units for resource gathering and ensuring economic progress to enable the production of military units in the first place. In our setup, these tasks are handled by rule sets provided by the StarCraft bot platform *TorchCraftAI* [15, 47]. The rule set includes a planning algorithm to execute a scripted unit build order which dictates the high-level strategy over the course of the game. Within each experiment, we provide identical rule configurations for all agents.

As is common in real-time strategy games, observations in StarCraft are limited to the immediate vicinity of the player's own units. The observations our agent receives are represented symbolically in a similar fashion as described by Farquhar et al. [13]. The win condition of a game is to destroy all buildings of the opponent. The final reward is 1 for a win, -1 for a loss.

4.2 Action Space Hierarchy

We consider a combinatorial action space inspired by the primitive actions of the Brood War game engine and typical human interaction via mouse and keyboard. At each time step, we select a subset of all controllable units, a single target location and a single command. Units will execute an action until a new one is issued to them specifically, allowing for control of multiple groups of units separately despite selecting only one action at each time step. We outline our proposed action space hierarchy below and refer to Appendix B for a comprehensive formal definition.

Based on the natural spatio-temporal hierarchy of the game, we propose to grow (1) the action space of unit selection, (2) the target position, and (3) the rate at which new actions are selected. Target position actions A_P are mapped to a fixed grid covering the whole playing area. The first and lowest level $l = 1$ corresponds to *build tiles*, each covering a 32-by-32 pixel region. With each successive action level we halve the resolution of the grid, so that each action at level $l + 1$ corresponds to 4

neighboring actions at level l. When taking an action at level l, a corresponding action at $l-1$ will be selected at random, proceeding until the lowest level.

Unit selection is modelled as a combination of binary actions A_v over build tiles. If $A_v = 1$, all units located in the respective grid cell will be included in the final action. The resulting joint action space $\mathcal{A}_V$ is large, and even with a common situation of 80 live units on the map up to $2^{80} \approx 10^{24}$ actions would need to be considered. As human players frequently perform selections based on proximity of units, we implement action space growing by biasing the selection in a similar fashion. A_v is sampled with spatially correlated yet uniform noise ϵ_V^l for different action levels l. At the lowest level of actions, this noise is generated uniformly, i.e. $\epsilon_V^1 \sim \mathcal{U}(0,1)$. For $l > 1$, we obtain spatial correlation by smoothing. Given an arbitrary smoothing kernel K_l, we can maintain uniformity over multiple time-steps by transforming the noise via $\mathrm{cdf}(K_l * \mathrm{icdf}(\epsilon_V^1))$, where icdf and cdf are the (inverse-)cumulative distributions of a Gaussian, and $*$ indicates the convolution operator. This ensures that the expectation of ϵ_v^l is maintained across different levels.

We support three primitive commands offered by the StarCraft game engine, $A_C \in \{Attack, Move, AttackMove\}$. At each time step, a policy operating at level l produces both a joint distribution over positions and commands $\pi_{P,C}^l$ and over units π_V^l. In Figure 1, we visualize our proposed hierarchy on a small sub-section of a 64-by-64 build tile map. The unit selection noise ϵ_V^l corresponds to uniform random noise at the lowest level and will gradually increase in spatial correlation at higher levels. The distribution over positions is shown for two of the three commands. The decrease in resolution for higher action levels directly results in fewer possible actions at the expense of accuracy. For example, the actions over which $\pi_{P,C}^4(A_C = Attack)$ is defined make it impossible to select an individual target among the red player's units on the bridge at the bottom. The policy can only chose whether or not to engage in a fight with the units on the bridge.

Finally, we structure the temporal dimension of the action space, motivated by the durative nature of StarCraft's in-game actions. With the policy making a decision every second of game time, we expect that we will rarely experience the full effect of extended actions such as moving units across long distances. Hence, we modify the number of time steps between issuing new commands to the game engine, denoted with $\delta_t^l = l$. The lowest level, $l = 1$, corresponds to taking an action at every time step. At a coarse level, only few actions can be performed during a game, e.g. 120 actions for $l = 8$ in a 16-minute game.

4.3 Multi-Scale Policy Model

Whenever the action space is refined, it is crucial to retain previously acquired higher-level behaviors for steady and fast learning progress. We achieve this by jointly training all action space levels, employing a single neural network parameterization for all resulting policies π^l. For unit selection, the same policy is used for different action levels and the correlation of the noise ϵ_V^l is adjusted. The hierarchy over $\mathcal{A}_{P,C}$ is obtained by parameterizing $\pi_{P,C}^1$ with learnable weights while $\pi_{P,C}^{l+1}$ is obtained by average-pooling $\pi_{P,C}^l$ with a 2-by-2 kernel for $l > 1$.

For our parameterization, we opt for an encoder-LSTM-decoder model, where a ResNet trunk [16] is used for encoding similar to Farquhar et al. [13]. Symbolic unit observations are first encoded individually, with the resulting embeddings then placed at their corresponding locations for concatenation with spatial features. To ease memorization of past observations, a spatially replicated LSTM performs temporal integration of the resulting features [40, 48] before the ResNet trunk is applied.

The encoder output is provided to a LSTM and decoded to its original resolution using upsample-convolution blocks with skip connections to the ResNet trunk. We employ two separate decoders for $\pi_{P,C}$ and π_V, such that we can condition the unit decoder on the sampled position and command. We only duplicate the upsample-convolution blocks of the model and share the LSTM and ResNet trunk.

Attack actions can only be issued to currently visible opponent units, and unit selection can be constrained to locations where the player's units are present. We mask out invalid actions by constraining the respective probabilities in $\pi_{P,C}(A_C = Attack)$ and π_V. Similarly, we implement play at coarser time resolutions by masking out all possible actions. For a time step t and temporal action level l, the agent will obtain policy gradients if $t \bmod \delta_t^l = 0$ only. Otherwise, gradients are solely provided by back-propagation through time used for the LSTM cells. An exhaustive model specification and details on game state featurization are provided in Appendices C.1 and C.2.

4.4 Growing Schedules

Over the course of training, we successively decrease the action level l to obtain finer-grained policies in a self-play setting of several agents. The common approach of hand-crafting a growing schedule comes with the pitfall that optimal training might require non-monotonic schedules. This is amplified by growing three dimensions of the action space in parallel: target positions, correlation of unit selection, and the time between actions. For example, deciding whether to flee a battle requires frequent actions but only a coarse target position, while tactical positioning is possible on larger time spans but can require very specific target and unit selection. We thus avoid the manual definition of schedules and introduce a refinement of the action space as a possible mutation operation in Population Based Training. With the premise that finer actions enable better game-play, appropriately-timed action space refinements should provide a competitive advantage for the respective agent.

Contrary to prior work on GAS, the single parameterization of policies detailed in §4.3 enables the use of *smooth* growing schedules in which multiple action levels can be utilized. We specify a distribution over action levels and perform refinement by by shifting the probability mass towards lower levels. The final resolution corresponds to a distribution with $Pr(l = 1) = 1$ and 0 for all $l > 1$. At each time step, a level is first sampled according to the current distribution, and the action will then be sampled from the respective policy. We apply a smooth growing schedule to the position action and specify single levels otherwise.

5 Experiments

Scenarios We select three different scenarios to run experiments and ablations, and investigate the effect of our proposed action space curricula on training speed and final performance. An *adversarial scouting* experiment allows us explore the effect of action space growing in an isolated setting. In the *mid-game* scenario, where we obtain relatively short matchups, players start out with two bases as well as military units and defensive buildings. Finally, we play *full* games where both players start out as in a standard game of StarCraft.

Fixed Baselines We ground the evaluation of all trained models by competing against handicapped versions of the fixed, rule-based StarCraft bot *CherryPi*. *CherryPi* is a top StarCraft bot: it won the SSCAIT 2017/2018 tournament and came in second at the AIIDE 2018 competition [56, 6]. We note *CPI* for the full bot with no handicap. *CPIIdle* does not take actions for the first 40 seconds of game time; *CPIRestrict50* and *CPIRestrict25* are versions that drop 50% and 25% of their actions randomly. The best two baselines are *CPI* and *CPIRestrict25*. Regarding economy and worker unit management, our trainable policy is identical to *CPI*.

Training Setup We train a population of several agents in parallel [19] where each agent is trained with IMPALA [12]. After a random pairing for matchups, a game trace is collected by playing another agent in the population; multiple games are played in parallel in a pool of rollout workers shared among the whole population. Chunks of traces are aggregated in a replay buffer and sampled for training with prioritized experience replay [38], and gradients are then computed with V-Trace off-policy correction.

After a fixed number of model updates, agents produce individual checkpoints of their parameters. Similar to Li et al. [23], the resulting policy enters an evaluation round with past population members. We compute a ranking using TrueSkill [17] and cull agents accordingly, replacing them by new ones initialized from a previous checkpoint and with hyper-parameters subject to mutation. We limit the number of archived agents – checkpointed policies – by excluding the worst-ranking ones from further evaluations and not considering them when initializing a new agent. Hyper-parameters for policy training and population management are equal for all experiments in a given scenario. A detailed listing is provided in Appendix C;. In all cases, the learning rate and action space levels are candidates for mutation.

Metrics While our PBT setup optimizes for the TrueSkill of a population, we can measure and compare the success of training runs with additional metrics. First, we compare win-rates against the fixed baselines described above. These baselines do not take part in any training games, except that they are taken into account for the PBT evaluation rounds in the *mid-game* scenario. In our full game

setup, we let members of different populations directly compete against themselves and analyze the resulting TrueSkill and Nash ratings [2].

5.1 Adversarial Scouting

In this task, the agent starts as in a normal game on a map with 4 starting locations, and the game ends when one player discovers the other player's base. Intuitively, randomly exploring at a high spatio-temporal frequency is harder than exploring at a lower spatio-temporal frequency, and splitting units to check the remaining locations should be beneficial. Therefore, growing all three of U, P, and T resolutions should allow the model to explore much faster.

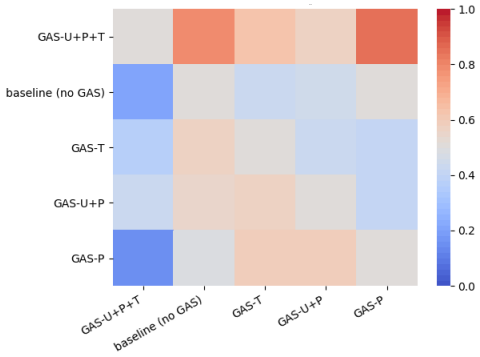


Figure 2: Results for the *adversarial scouting* experiment, where being first to discover the opponent's base results in a win. We play 30 games between each agent after training, and display the winrate for each row.

For both tasks, after we finish training, we pit the agents against each other and show the winrates in Figure 2. The best agent in the GAS run was able to discover that quickly moving the initial single unit ("Overlord") towards the closest enemy base candidate location was a good strategy. Because this unit is very slow and rarely reaches the base before other units spawn, it shows that the model is able to pick up a very long-horizon reward signal. Additionally, the model will move the faster units to a different base than the initial overlord as soon as they are produced. Figure 2 indicates that growing all three action space dimensions is most effective on this task.

5.2 Mid-Game Scenario

We propose a mid-game scenario to ablate our experiments. We use a map with 4 bases, and begin the games with each player having a few military units and controlling of two of the bases. In order to compare the effects of our GAS scheme and population based training we run: (1) a baseline with no GAS, (2) Growing both unit selection and target position resolution, (3) Growing only temporally, (4) Growing all three axes. To further speed up training, we provide an additional, small reward signal based on the differences in required army supply, i.e. unit counts.

Over the course of the training, agents trained with GAS learn different strategies as the overall level of the population increases. First, agents learn to gather their military units in one base, and wait for the enemy to attack. Then, they value the center of the map and fight over it to prevent the enemy from gathering all its units together. Later on, they learn that they should retreat their units to defend when their base is under attack, or sometimes choose to trade bases if they are already attacking an enemy base. Finally, they learn to refine their unit selection to handle different unit types differently.

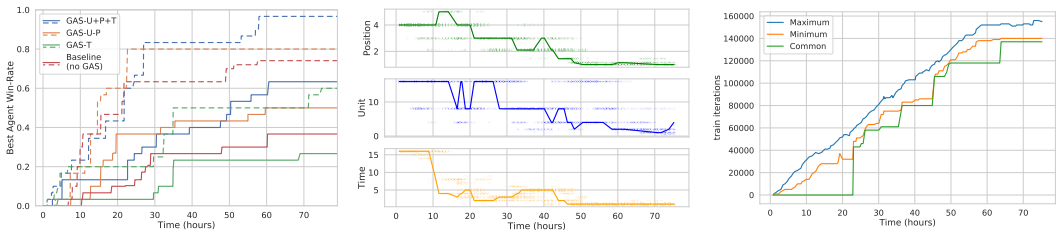


Figure 3: Results and analysis for the *mid-game* scenario. **Left** Best win-rate obtained up to different wall clock times against *CPI* in full and *CPIRestrict25* in dashed lines. **Center** Resolution schedules found by PBT: each dot represents an agent, and the line is the schedule for the lineage of the best agent in the final evaluation round. **Right** Population diversity for GAS: bounds of training iterations for agents in the population, and training iterations of their common ancestry.

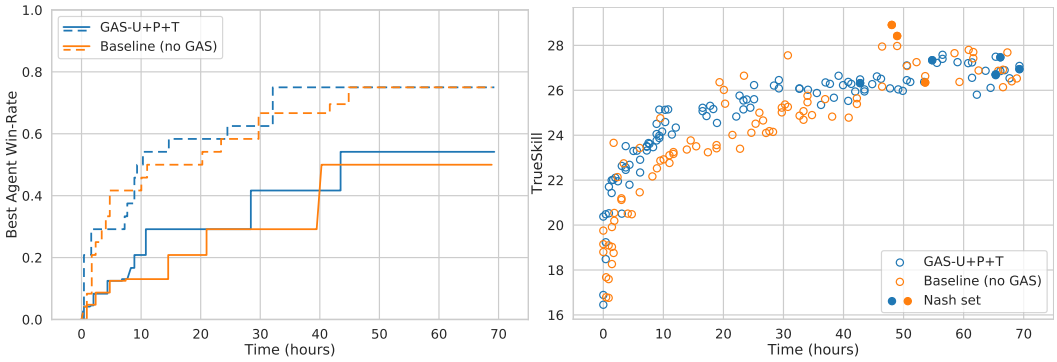


Figure 4: Results for *full* games. **Left** Best win-rate obtained up to different wall clock times against *CPI* in full and *CPIRestrict25* in dashed lines. **Right** TrueSkill ratings and Nash equilibrium set, determined by a tournament pitching top agents of the GAS and baseline runs against each other. Each data point represents a single checkpointed policy, placed at the wall clock training time it was produced.

After training for 80 hours, we compare the results of the ablations in Figure 3: training runs without GAS for U, P, and T obtain lower maximum win-rates against *CPI* and *CPIRestrict25* (fig. 3 left). The policies trained on those runs do not use the "Move" action and almost exclusively rely on "Attack-Move" actions. While "Move" actions are essential to retreat or flee, they are almost impossible to learn without a proper exploration scheme. Indeed, for a retreat action to be successful, the policy has to select an entire group of units (retreating half of the army causes the loss of the other half), and commit to this action for a few seconds until the group reaches a safe place.

In the population trained with GAS we observe steady progress. At a given time, different agents are training at different levels of detail and exploring a diverse set of strategies (fig. 3 center). For time resolution for instance, progress is not monotonic: even after reaching the finest resolution, it still makes sense to explore at a coarser resolution to discover new strategies before refining at the lowest level as shown by the highlighted schedule of the best final agent. Periods of exploration and exploitation alternate at the scale of the population as well. In Figure 3 on the right we plot the minimum and maximum number of training iterations for archived models and compare it to the training iterations for the most recent common ancestor of the archived population. If the gap to the common ancestor increases, so does the overall diversity of the policies. When a new and significantly stronger policy is discovered, its offspring gradually fills the archive slots until all agents share an ancestor.

5.3 Full Game

Finally, we compare GAS against the baseline (operating at the highest-resolution action space) in full games on a two-player map, starting from the standard StarCraft state of an initial building and five worker units. Here, we grow the action space along all three dimensions: correlation of unit selection, target positions, and the time interval between taking actions. In contrast to the *mid-game* experiment, we provide a single win-lose reward per game only and do not rank against *CPI* versions in evaluation rounds. We observe that unit control can be learned from scratch in both cases as agents achieve a final win rate above 50% against *CPI* without facing it in any training game (fig. 4 left). With and without GAS, *CPIRestrict25* is dominated after 10 hours of wall clock training time and *CPI* shortly after 40 hours, with the GAS agents achieving win-rate thresholds faster in general.

We further compare performance by playing a tournament between the best 4 checkpointed agents from every PBT evaluation round. With a total of 195 agents we play 25 games per matchup and consider intra- as well as inter-population pairings. We then compute rankings by TrueSkill (fig. 4, right). During the first half of training, the population successively growing its action space exhibits higher TrueSkill on average while the baseline run yields several top-ranking agents. Afterwards, TrueSkill averages are similar. GAS produces the first agent in the Nash set, and the baseline run contains the four agents with the highest TrueSkill. Looking purely at the Nash ratings for this tournament, the difference between the two populations is less pronounced (Appendix D). The Nash

set is made up of five GAS and three baseline agents. We conclude that although GAS makes faster progress overall and is first to produce a Nash set entry, neither run discovered a truly dominating strategy.

6 Conclusion

In this work, we proposed methods for successively growing the action space of reinforcement learning agents in the context of real-time strategy games. Our policy architecture and sampling scheme enable exploitation of hierarchies over actions without introducing additional parameters. Consequently, experience obtained with coarse high-level actions directly helps training low-level actions, and the result is faster acquisition of better policies with equal or higher final performance. In a challenging military unit control task in StarCraft: Brood War, we beat a strong rule-based bot in full games without reward shaping or supervised policy initialization.

References

[1] Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. Emergent Tool Use From Multi-Agent Autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

[2] Balduzzi, D., Tuyls, K., Perolat, J., and Graepel, T. Re-evaluating evaluation. In *Advances in Neural Information Processing Systems 31*, pp. 3268–3279. 2018.

[3] Bansal, T., Pachocki, J., Sidor, S., Sutskever, I., and Mordatch, I. Emergent Complexity via Multi-Agent Competition. *arXiv preprint arXiv:1710.03748*, 2017.

[4] Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, pp. 41–48. ACM, 2009.

[5] Carion, N., Usunier, N., Synnaeve, G., and Lazaric, A. A structured prediction approach for generalization in cooperative multi-agent reinforcement learning. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8128–8138. Curran Associates, Inc., 2019.

[6] Churchill, D. StarCraft AI Competition Results. `https://www.cs.mun.ca/~dchurchill/starcraftaicomp/2018/`, 2018. (Accessed: 2019-11-29).

[7] Churchill, D., Saffidine, A., and Buro, M. Fast Heuristic Search for RTS Game Combat Scenarios. In *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'12, pp. 112–117, 2012.

[8] Czarnecki, W. M., Jayakumar, S. M., Jaderberg, M., Hasenclever, L., Teh, Y. W., Osindero, S., Heess, N., and Pascanu, R. Mix & Match - Agent Curricula for Reinforcement Learning. *arXiv preprint arXiv:1806.01780*, 2018.

[9] Dayan, P. and Hinton, G. E. Feudal Reinforcement Learning. In *Advances in Neural Information Processing Systems 5*, pp. 271–278, 1993.

[10] Dietterich, T. G. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[11] Elman, J. L. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.

[12] Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *International Conference on Machine Learning*, 2018.

[13] Farquhar, G., Gustafson, L., Lin, Z., Whiteson, S., Usunier, N., and Synnaeve, G. Growing action spaces. *arXiv preprint arXiv:1906.12266*, 2019.

[14] Florensa, C., Held, D., Wulfmeier, M., Zhang, M., and Abbeel, P. Reverse curriculum generation for reinforcement learning. 2017.

[15] Gehring, J., Lin, Z., Haziza, D., Mella, V., Gant, D., Carion, N., Ju, D., Rothermel, D., Gustafson, L., Kharitonov, E., Usunier, N., and Synnaeve, G. TorchCraftAI v1.1, July 2019. URL `https://doi.org/10.5281/zenodo.3341787`.

[16] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[17] Herbrich, R., Minka, T., and Graepel, T. Trueskill™: a bayesian skill rating system. In *Advances in neural information processing systems*, pp. 569–576, 2007.

[18] Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[19] Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

[20] Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castaneda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., et al. Human-level performance in first-person multiplayer games with population-based deep reinforcement learning. *arXiv preprint arXiv:1807.01281*, 2018.

[21] Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., and Dabney, W. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2019.

[22] LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pp. 396–404, 1990.

[23] Li, A., Spyra, O., Perel, S., Dalibard, V., Jaderberg, M., Gu, C., Budden, D., Harley, T., and Gupta, P. A generalized framework for population based training. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, 2019.

[24] Machado, M. C., Bellemare, M. G., and Bowling, M. A Laplacian Framework for Option Discovery in Reinforcement Learning. In *International Conference on Machine Learning*, 2017.

[25] McGovern, A. and Barto, A. G. Automatic discovery of subgoals in reinforcement learning using diverse density. *Proceedings of the 18th International Conference on machine learning*, 2001.

[26] Moore, A. W. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. In *Advances in neural information processing systems*, pp. 711–718, 1994.

[27] Munos, R. and Moore, A. Variable resolution discretization in optimal control. *Machine learning*, 49(2-3): 291–323, 2002.

[28] Murali, A., Pinto, L., Gandhi, D., and Gupta, A. Cassl: Curriculum accelerated self-supervised learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6453–6460. IEEE, 2018.

[29] Ng, A. Y., Harada, D., and Russell, S. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, volume 99, pp. 278–287, 1999.

[30] Ng, A. Y., Russell, S. J., et al. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, volume 1, pp. 2, 2000.

[31] Nouri, A. and Littman, M. L. Multi-resolution exploration in continuous spaces. In *Advances in neural information processing systems*, pp. 1209–1216, 2009.

[32] Ontanón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., and Preuss, M. A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in games*, 5(4):293–311, 2013.

[33] OpenAI. How to Train Your OpenAI Five. `https://openai.com/blog/how-to-train-your-openai-five/`, 2018. Accessed: 2019-05-16.

[34] Ronneberger, O., Fischer, P., and Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv:1505.04597 [cs]*, May 2015. URL `http://arxiv.org/abs/1505.04597`. arXiv: 1505.04597.

[35] Samuel, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 1959.

[36] Sandholm, T. Abstraction for solving large incomplete-information games. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pp. 4127–4131, 2015.

[37] Schaal, S. Learning from demonstration. In *Advances in neural information processing systems*, pp. 1040–1046, 1997.

[38] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[39] Selfridge, O. G., Sutton, R. S., and Barto, A. G. Training and tracking in robotics. In *IJCAI*, pp. 670–672, 1985.

[40] Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-k., and Woo, W.-c. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. In *Advances in Neural Information Processing Systems 28*, pp. 802–810. 2015.

[41] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[42] Stolle, M. and Precup, D. Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*, pp. 212–223. Springer, 2002.

[43] Sukhbaatar, S., Lin, Z., Kostrikov, I., Synnaeve, G., Szlam, A., and Fergus, R. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.

[44] Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*, volume 1. MIT press, 1998.

[45] Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[46] Synnaeve, G. and Bessiere, P. A bayesian model for rts units control applied to starcraft. In *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, pp. 190–196. IEEE, 2011.

[47] Synnaeve, G., Nardelli, N., Auvolat, A., Chintala, S., Lacroix, T., Lin, Z., Richoux, F., and Usunier, N. TorchCraft: A Library for Machine Learning Research on Real-Time Strategy Games. *arXiv preprint arXiv:1611.00625*, 2016.

[48] Synnaeve, G., Lin, Z., Gehring, J., Gant, D., Mella, V., Khalidov, V., Carion, N., and Usunier, N. Forward Modeling for Partial Observation Strategy Games - A StarCraft Defogger. In *Advances in Neural Information Processing Systems 31*, pp. 10738–10748. 2018.

[49] Tallec, C. and Ollivier, Y. Can recurrent neural networks warp time? In *International Conference on Learning Representations*, 2018.

[50] Tesauro, G. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[51] Todorov, E., Erez, T., and Tassa, Y. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

[52] Usunier, N., Synnaeve, G., Lin, Z., and Chintala, S. Episodic Exploration for Deep Deterministic Policies for StarCraft Micromanagement. In *International Conference on Learning Representations*, 2017.

[53] Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3540–3549. JMLR. org, 2017.

[54] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[55] Wu, Y. and He, K. Group Normalization. *arXiv:1803.08494 [cs]*, June 2018. URL `http://arxiv.org/abs/1803.08494`. arXiv: 1803.08494.

[56] Čertický, M., Churchill, D., Kim, K.-J., Čertický, M., and Kelly, R. StarCraft AI Competitions, Bots, and Tournament Manager Software. *IEEE Transactions on Games*, 11(3):227–237, 2019.

A Categories and Examples of GAS

In the continuous control setting (e.g. MuJoCo [51]), we can often assume a C-Lipschitz action-value function Q with $|Q(x, u + \epsilon) - Q(x)| < C\epsilon$. When given a continuous action space $\mathcal{A}$, it is then common to apply discretization to obtain a finite set of actions $\mathcal{A}'$. Different levels of abstraction now correspond to different resolutions of discretization. Another possibility is to use a selected subset of the action space dimensions only, or, in a practical setting, a subset of the actuators of the robotic arm described in §1. For categorical action spaces, contingent hierarchies or taxonomies can be leveraged: if actions can be recursively grouped into fewer ones, the action space can be grown along the resulting tree structure. Such abstractions exist naturally in card games like Poker [36] and reinforcement learning environments such as DMLab [8].

However, a trivial taxonomy or structure for restricting the action space is not always available. As an example, consider the task of selecting a subset from a set of N elements. The action space of N binary actions grows exponentially to 2^N. Restricting the cardinality of the selected subset $M < N$, for example so that at most M or at least M elements are selected, will not be a sensible choice. On the other hand, these binary actions may be subject to spatial or semantic constraints. Examples include traffic light control or multi-node job scheduling on clusters where performance can be impacted by network proximity. We exploit a similar spatial relationship for the StarCraft unit control task that we consider here. With 80 live units to select, which is not an unusual game situation, exploring an action space with $2^{80} \approx 1.2 \times 10^{24}$ possible subsets is extremely challenging. Indeed, human payers do not require millions of hours of game-play to realize that the number of meaningful selections is limited, e.g. to selecting all units or a single group of nearby units. We therefore exploit the spatial relationship of the candidate elements by introducing correlated sampling. While the policy network always outputs a selection probability for each unit, the sampling procedure ensures that spatially close units are more likely to be selected or ignored together. The action space resolution is controlled by varying the amount of spatial correlation (§4.2). The sole requirement for this growing strategy is the ability to define a meaningful distance measurement between set elements.

In the options framework proposed by Sutton et al. [45] and Stolle & Precup [42], a Markov option is executed by choosing an action according to its distribution and then either choosing a new option with probability $\beta(s_t)$ or following the current option. Here, one method of growing the action resolution would be to slowly warm up β during the training procedure. This results in agents that, at the beginning of training, will follow options for a long time before switching. Orthogonal to the construction of options, complex environments may offer actions that are already durative. If we expect that the frequency at which we take actions is generally too high to experience the effect of such durative actions, one possible avenue is to simply execute the action multiple times in a row. By slowly lowering the number of repeats during learning we can again grow the effective action space. For example, random exploration might not be effective for control of a robotic arm, but encoding the inductive bias of moving in a certain direction for a long time in the beginning of training could speed up learning.

B Detailed Action Space Description

Formally, each action is a 3-tuple $A = \langle A_U, A_P, A_C \rangle$. A_U is itself a combinatorial action with $A_U = \langle A_u \rangle_{u \in U}$ for a set of units U and binary actions A_u for selection and non-selection, respectively. The target position $A_P = \langle x, y \rangle$ corresponds to a pixel coordinate on the map. We support three primitive commands offered by the StarCraft game engine, $A_C \in \{Attack, Move, AttackMove\}$. The actions respectively result in attacking an enemy unit, moving to a target location, and moving to a location while engaging in fights with opponent units encountered along the way. Commands and positions are directly tied to each other and we thus model $\langle \mathcal{A}_P, c \rangle$ as a joint action space $\mathcal{A}_{P,C}$. Below, we detail how we grow the action space for target positions and unit selection; our approach for increasing the frequency of actions is described in §4.2.

B.1 Target Position

Denoting with $\mathcal{A}_P^l$ the action set over positions at level l, $\mathcal{A}_P^1$ is defined over build tiles, each corresponding to a 32-by-32 pixel region. With each successive level, we halve the resolution of

positions so that $4|\mathcal{A}_P^l| = |\mathcal{A}_P^{l+1}|$. Consequently, a 2-by-2 neighborhood of positions in level l is collapsed to a single action in level $l+1$. As noted in §4.3, we obtain higher-level actions by average pooling. Formally, for a history of observations until the current time-step $o_1, \ldots, o_t$ and a grid cell location x^{l+1}, y^{l+1}, the policy $\pi_{P,C}^{l+1}$ is defined as

$$\pi_{P,C}^{l+1}(A_C, y^{l+1}, x^{l+1}|o_1, \ldots, o_t) = \frac{1}{4}\sum_{m=0}^{1}\sum_{n=0}^{1}\pi_{P,C}^{l}(A_C, 2y^l + m, 2x^l + n|o_1, \ldots, o_t)\ .$$

Sampling a position at level $l+1$ amounts to randomly selecting any of the corresponding positions at level l; at the finest level we select the center pixel of the respective build tile. This action hierarchy results in multiple joint action spaces $\mathcal{A}_{P,C}^l$, with $\mathcal{A}_C$ being fixed to the three supported commands at each level.

B.2 Unit Selection

Per-unit actions A_u are modelled on a fixed grid with a proxy action space $\mathcal{A}_V$, consisting of binary actions A_v for each build tile on the map. If $A_v = 1$, all units located in the corresponding build tile will be selected for the joint action A. While the resulting action space is large ($2^{4096} \approx 10^{10^3}$ for a 64-by-64 build tile map), only a limited number of build tiles will contain controllable units at a given time step. Human players however realize quickly that there are only a limited number of meaningful selections, e.g. selecting all units or a single group of nearby units. As described in §4.2, we use correlated noise ϵ_V^l to achieve action space growing when sampling A_v:

$$A_v = \begin{cases} 1 & \text{if } \pi_v > \epsilon_v^l \\ 0 & \text{otherwise}\ , \end{cases}$$

with π_v denoting the probability of selecting units at position v according to the current policy π. For the finest level, $\epsilon_V^1 \sim \mathcal{U}(0,1)$, while $l > 1$ is obtained by smoothing with a Gaussian kernel. For $l > 1$, we obtain spatial correlation by smoothing while maintaining uniformity (in expectation) over multiple time-steps as follows:

$$\epsilon_V^k = \mathrm{cdf}(K_l * \mathrm{icdf}(\epsilon_V^1)),$$

where icdf and cdf are the (inverse-)cumulative distributions of a Gaussian, and $*$ indicates the convolution operator. For our experiments, we use the $K_l\,(x, y) = \exp\left(-\left(x^2 + y^2\right) / \left(2 \cdot \sigma_l^2\right)\right)$ as the smoothing kernel, where σ_l is a characteristic distance expressed in build tiles.

C Training Details

C.1 Features

We provide our models with symbolic per-unit features and spatial map features with a build tile (32×32 pixels) resolution. The following per-unit features are used, normalized as in `UnitStatVFeaturizer` from Gehring et al. [15] and resulting in 78-dimensional feature vectors x_s:

- X and Y pixel position
- X and Y velocity
- Health, shield and energy levels
- Cool-down for ground and air weapons
- Armor and shield armor levels
- Armor type
- Attack, range, and damage type for ground and air weapons
- A total of 52 unit flags such as `Attacking` or `CarryingMinerals`[1]

We do not use any memorization of opponent units, i.e. for the policy model we featurize units that can be observed at the time of acting only. We also supply unit walk tile positions x_p and a one-hot encoding of the unit type x_i.

[1] See `https://github.com/TorchCraft/TorchCraft/blob/8f09ba0/include/frame.h#L108` for a listing of all unit flags.

Spatial map features x_m are provided as a three-dimensional $C \times Y \times X$ tensor. The following features are extracted with *walk tile* resolution (8×8 pixels) and then downsampled to build tile resolution by average pooling.

- Binary walkability information
- Ground height as one-hot over 4 channels
- Binary information about creep and fog-of-war, in separate channels
- Binary indicators for building and resource presence, in separate channels
- Candidates for the enemy starting location, set to zero when a location was found to not be the actual starting location
- Absolute X and Y positions (from 0 to 1) for every walk tile

C.2 Model

As described in Section 4.3 we use a shared encoder followed by individual decoders for $\pi_{P,C}$ and π_V. The encoder consists of (1) a module $\mathbf{R}$ that embeds the units and adds scatter connections to the spatial map, and (2) a ResNet $\mathbf{E}$ that encodes the statistics of the game map at multiple different scales.

The decoder $\mathbf{D}$ successively upsamples the multiple different scales output by $\mathbf{E}$ and outputs a final policy map at the size of the entire map. We apply GroupNorm [55] to the output of all convolution blocks.

Recall that the model has two separate $\mathbf{D}$ branches, one to generate the position $\pi_{P,C}$ and another to generate the unit targets π_V. We condition $\pi P, C$ on the current selection of units. Hence, the $\mathbf{D}$ module for $\pi_{P,C}$ takes as an additional input the encoded actions of π_V.

The $\mathbf{R}$ module takes in as input the map features x_m and the unit statistics x_s, the unit types x_i, and the unit locations x_p. For each faction f, each unit is individually embedded into $e_R^f = \mathrm{MLP}(\mathrm{Concat}(x_s, \mathrm{Embed}(x_i))$ Then, we merge e_R^f and x_m of player f with scatter connections - for all units $K = k_2, k_2, \ldots$ with the same x_p, we sum together their unit embeddings and place it at the location in a map, e.g. $e_m^f[x_p^k] \leftarrow \sum_{k \in K} e_R^{f,k}$. Finally, we process this with a single $4\times$ downsampling, to convert "walktile" information to "buildtile" level information.

The $\mathbf{E}$ module is a standard ResNet. We choose to use 5 ResNet blocks of 4 layers each, where the end of each block has a downsampling layer, and the result is output to either the linear projection for the value model, or as skip connections for $\mathbf{D}$ Each block starts with 28 hidden units, and doubles after each block for a final embedding dimension of 448. All blocks downsample by 2 except the first and last, which do a $4\times$ downsampling.

We employ two single-layer LSTMs in the policy model for temporal integration of observations. We process the output of the first ResNet block of the encoder $\mathbf{E}$ with a small 28-unit, spatially replicated LSTM [48]. Furthermore, we feed the final output of $\mathbf{E}$ to a standard 448-unit LSTM, In both cases, the LSTM output is concatenated with its input for further processing within $\mathbf{E}$ and $\mathbf{D}$. We initialize the LSTM weights as proposed by Tallec & Ollivier [49], and follow Kapturowski et al. [21] in using a small LSTM burn-in of two frames during training.

The $\mathbf{D}$ module uses a series of upscale-convolutions to upsample representations output from $\mathbf{E}$. $\mathbf{E}$ followed by $\mathbf{D}$ looks like a U-net [34] style convolutional network, with extra LSTMs as described above. Each decoder network takes in as input the skip connection from $\mathbf{E}$, and outputs a upscaled low dimensional representation. The same block of residual layers apply, where the upsampling is at the beginning of the layer. We use 5 blocks of 2 layers each, for a total of 10 convolutional layers in the decoder. The output of the network at the finest scale is just 12 dimensions, and coarsest scale is 192 dimensions. We choose a small embedding dimension to force most of the information content to be represented by the skip connections from the encoder.

Please check Figures 5, 6 and 7 for a schematic of the model.

Model training is performed in an actor-critic fashion Sutton & Barto [44]. While the policy is computed on partial observations only, we grant the critic access to full observations. The critic function is parameterized separately from the policy network using the same architecture. The availability of full observations removes most of the need for temporal integration so that we do not include recurrent layers in the critic network.

Parameter	Scouting	Mid-Game	Full Game
Active population size	2	8	8
Maximum archived agents	2	16	64
Ratio of training games vs. archived pop.	0.3	0.1	0.3
Batch size	120	56	120
BPTT steps	16	48	24
Checkpoint interval (updates)	500	1000	800
Evaluations per hour (avg)	5.18	1.60	1.58
Entropy factor (fixed/mutated)	10^{-4}	Mutated	10^{-4}
PBT Mutation rate	0.5	0.8	0.5
Ranking against *CPI* versions	No	Yes	No
Reward	Win/lose	Win/lose + Δsupply	Win/lose
Discount factor	1.0	0.997	0.99
Replay buffer size	32000	120000	60000

Table 1: Training hyper-parameters per scenario.

To summarize, with input features described above x_m, x_s, and x_p:

- The unit selection head for π^0_V as described in 4.3 is $\mathbf{D}_V(\mathbf{E}(\mathbf{R}(x_m, x_s, x_p)))$. Let us denote the actual sampled policy as Π_V.
- The position and command policy head $\pi^0_{P,C}$ as described in 4.3 is $\mathbf{D}_{P,C}(\mathbf{E}(\mathbf{R}(x_m, x_s, x_p)), \Pi_{P,C})$.
- The value head is $\mathrm{Linear}(\mathbf{E}_Z(\mathbf{R}_Z(x_m, x_s, x_p)))$ with full observation, i.e. with all opponent and allied units featurized.

C.3 Hyper-Parameters

We detail the specific hyper-parameters for each of the scenarios described in §5 in Table 1. In all experiments, we further set the following hyper-parameters:

Parameter	Value
Training algorithm	IMPALA
Optimizer	Adam
Initial replay buffer fill rate	20%
Gradient clipping via projection	1.0
BWAPI (StarCraft) frame skip	4

C.4 Smooth Target Position Schedules

As detailed in §4.4, we use a smooth growing schedule for the target position action. Instead of a operating at a fixed action level for the entirety of a game, we sample a level every time we take an action from a probability distribution over all action levels. In Table 2, we list the distributions that are used during training. They are indexed using a meta-level from 1 (corresponding to action level 1) to 5 (the initial distribution at the start of training). The mutation for PBT is then to either increase or decrease the meta-level.

C.5 Computing Infrastructure

All experiments are performed on a mix of GPU machines (for training and model forwards) and CPU-only machines (for rollouts, i.e. acting and executing the StarCraft game engine). For the scouting experiment, each population member uses 4 GPUs for training. For the mid-game and full game experiments, training is distributed over 8 GPUs for each population member, i.e. 64 GPUs in total are used for training only. Another set of GPUs is dedicated to model forwards for active and archived population members (80 for the full game setting). Two separate pools of CPU-only

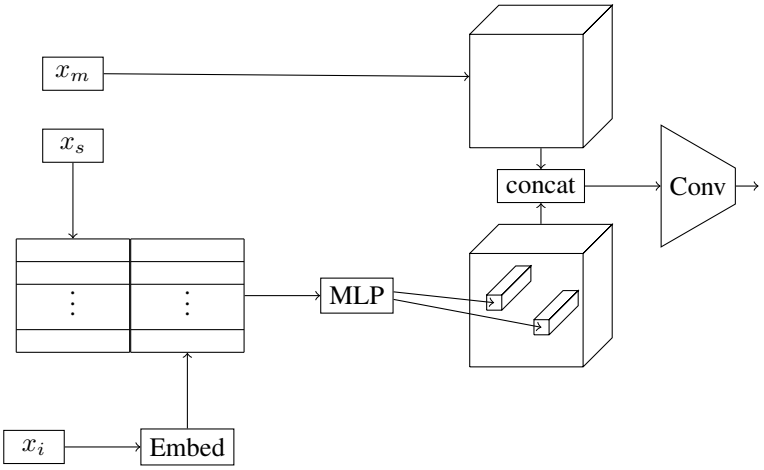


Figure 5: The embedding module R, which converts x_m, x_s, x_p, and x_i, the map features, unit statistics, unit types, and unit positions respectively, to a spatial map that E takes as input.

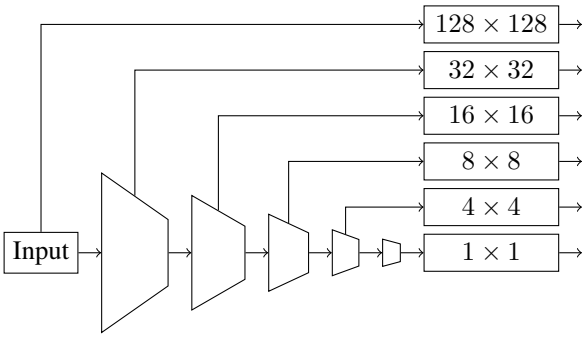


Figure 6: The encoder module E, which passes an input spatial map through 5 blocks of downsampling, and outputting the results at each resolution. Each block is a 4-layer ResNet block.

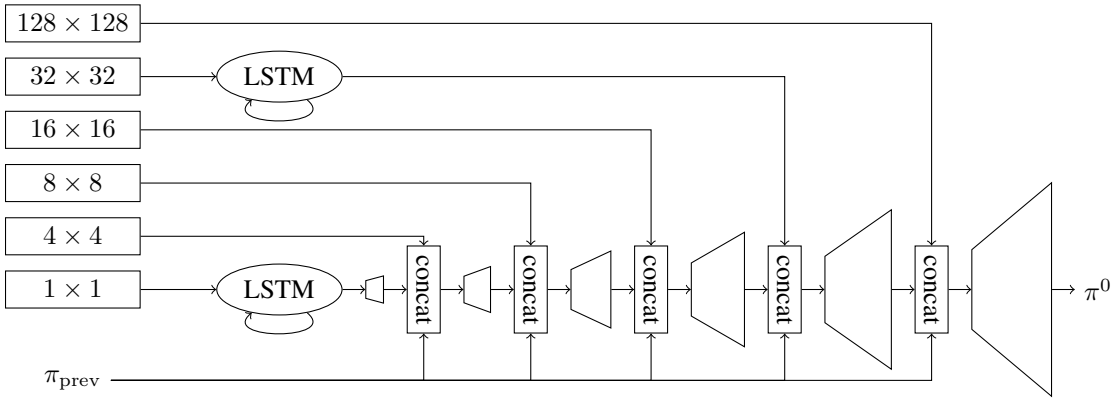


Figure 7: The decoder module D, which passes the input spatial maps through 5 success layers of upsampling, resulting in a walktile resolution of the policy. Recall that the final π^1 is again downsampled to obtain the resultant π^l at each level of the growing scheme. In cases where we have to model autoregressively, we input the previously sampled $\pi_{P,C}$ - this branch is turned off for the calculation of $\pi_{P,C}$.

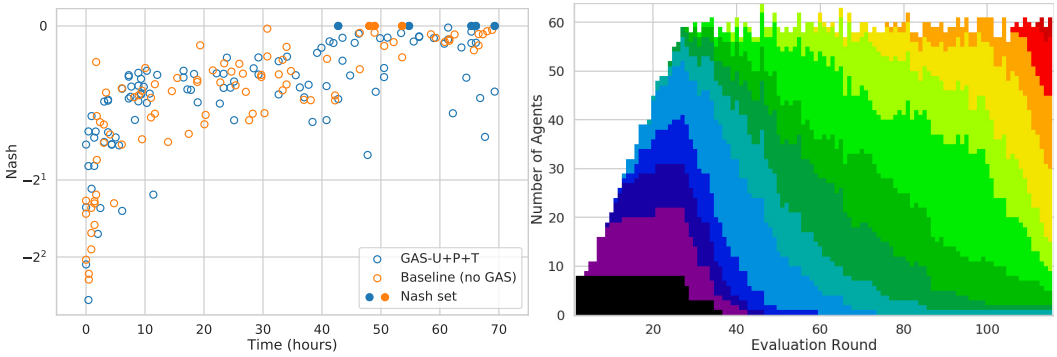
	Action Levels					
Meta-Level	1	2	3	4	5	6
1	1.0	0	0	0	0	0
2	1.0	0.5	0.25	0	0	0
3	0.5	1.0	0.5	0.25	0	0
4	0.25	0.5	1.0	0.5	0.25	0
5	0	0.25	0.5	1.0	0.5	0.25

Table 2: Meta-levels for smoothly growing the target position action space. Each meta-level is a distribution over actual action levels, obtained by normalizing the corresponding frequencies.

machines handle training and evaluation games, respectively. For the full game experiments, we use 880 cores for training and 1800 cores for evaluation.

D Full Game Results

In Figure 8a, we provide another version of Figure 4 (right) in which policies are ranked by Nash value instead of TrueSkill. In Figure 8b, we show how checkpointed agents from different generations make up the archived population during training. An agent's generation count is increased whenever it is warm-started from a checkpointed policy.

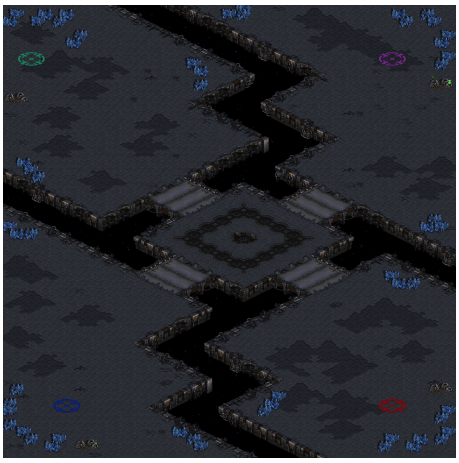


(a) Tournament results.

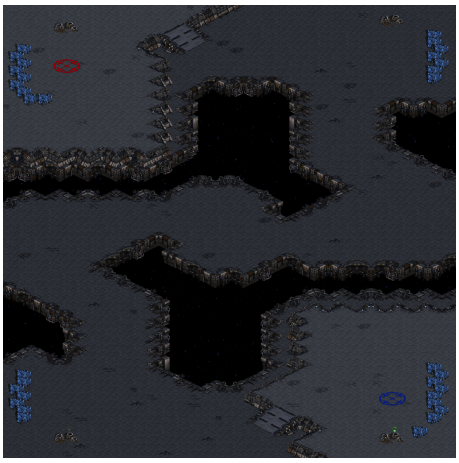
(b) Generations of archived agents.

Figure 8: **(a)** Corresponding to Figure 4. Nash values and Nash equilibrium set, determined by a tournament pitching top agents of the GAS and baseline runs against each other. Each data point represents a single checkpointed policy. **(b)** Generations of archived agents over the course of the GAS training run. Each color marks a different generation. As training progresses, new generations with refined action spaces take over the available slots.

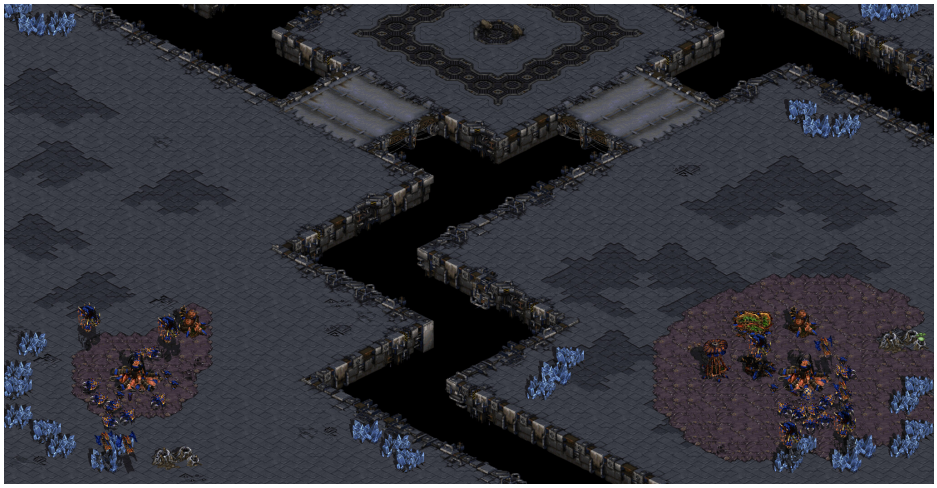
E Scenarios



(a) Mid-game scenario map



(b) Full game map



(c) Initial configuration on the Mid-Game scenario. Each player gets 2 bases out of the 4 randomly, a few military units, buildings required to spawn more military units, and some static air/ground defense

Figure 9: StarCraft maps used for mid-game scenario and full game.