Parameter	Value	Description
batch size	512/8192	Learning batch size
minibatch	256/512	Learning minibatch size
sgd iters	1	Optimization epochs per batch
frag length	32/256	Rollout fragment length
unroll	16	BPTT unroll horizon
λ	1.0	Standard GAE parameter
kl	0.2	Initial KL divergence coefficient
lr	5e-5	Learning rate
vf	1.0	Value function loss coefficient
entropy	0.0	Entropy regularized coefficient
clip	0.3	PPO clip parameter
vf clip	10.0	Value function clip parameter
kl target	0.01	Target value for KL divergence

Table 3: Training hyperparameters

A Training

We use RLlib's default PPO implementation. This includes a number of standard optimizations such as generalized advantage estimation and trajectory segmentation with value function bootstrapping. We did not modify any of the hyperparameters associated with learning. We did, however, tune a few of the training scale parameters for memory and batch efficiency. On small maps, batches consist of 8192 environment steps sampled in fragments of 256 steps from 32 parallel rollout workers. On large maps, batches consist of 512 environment steps sampled in fragments of 32 steps from 16 parallel workers. Each rollout worker simulates random environments sampled from a pool of game maps. The optimizer performs gradient updates over minibatches of environment steps (512 for small maps, 256 for large maps) and never reuses stale data. The BPTT horizon for our LSTM is 16 timesteps.

B Population Size Magnifies Exploration

Population	Lifetime	Achievement	Player Kills	Equipment	Explore	Forage
4	89.26	1.40	0.00	0.00	7.66	17.36
32	144.14	2.28	0.00	0.00	11.41	19.91
256	227.36	3.32	0.00	0.00	15.44	21.59

Table 4: Accompanying statistics for Figure 4

C The REPS Measure of Computational Efficiency

Formatted equation accompanying our discussion of efficient complexity on the project site

$$\textbf{Real-time experience per second} = \frac{\textbf{Independently controlled agent observations}}{\textbf{Simulation time} \times \textbf{Real time fps} \times \textbf{Cores used}} \quad (1)$$

Parameter	Value	Description
		Encoder
discrete	range$\times$64	Linear encoder for n attributes
continuous	$n\times$64	Linear encoder for n attributes
objects	64$n\times$64	Linear object encoder
agents	64$\times$64	Self-attention over agent objects
tiles	conv3-pool2-fc64	3x3 conv, 2x2 pooling, and fc64 projection over tiles
concat-proj	128$\times$64	Concat and project agent and tile summary vectors
		Hidden
LSTM	64	Input, hidden, and output dimension for core LSTM
		Decoder
fc	64$\times$64	Dimension of fully connected layers
block	fc-ReLU-fc-ReLU	Decoder architecture, unshared for key/values.
decode	block(key) $\cdot$ block(val)	state-argument vector similarity
sample	softmax	Argument sampler over similarity scores

Table 5: Architecture details

D Architecture

Our architecture is conceptually similar to OpenAI Five's: the core network is a simple one-layer LSTM with complex input preprocessors and output postprocessors. These are necessary to flatten the complex environment observation space and compute hierarchical actions from the flat network hidden state.

The input network is a two-layer hierarchical aggregator. In the first layer, we embed the attributes of each observed game object to 64 dimensional vector. We concatenate and project these into a single 64-dimensional vector, thus obtaining a flat, fixed-length representation for each observed game object. We apply self-attention to player embeddings and a conv-pool-dense module to tile embeddings to produce two 64-dimensional summary vectors. Finally, we concat and project these to produce a 64-dimensional state vector. This is the input to the core LSTM module.

The output network operates over the LSTM output state and the object embeddings produced by the input network. For each action argument, the network computes dot-product similarity between the state vector and candidate object embeddings. Note that we also learn embeddings for static argument types, such as the north/south/east/west movement direction options. This allows us to select all action arguments using the same approach. As an example: to target another agent with an attack, the network computes scores the state against the embedding of each nearby agent. The target is selected by sampling from a softmax distribution over scores.