A. Exploration Over Attributes

Traditional count-based exploration adds a reward bonus proportional to $N^{-0.5}$ in a reinforcement learning context where an empirical reward is provided. For finite MDPs, this bonus decays to 0 as $t \to \infty$, which means that in the long-time limit the agent is still finding a policy that optimizes the original MDP.

In our setting, however, we are interested in *pure exploration*, where the goal is to sample states (attribute transitions) somewhat uniformly, or to minimize the uncertainty in the empirical transition probabilities c_π. In this setting, it's problematic for the reward to converge to 0 uniformly as $t \to \infty$, because the optimal policy becomes degenerate when $r_t \to 0$.

Instead, we consider a reward of $r_t = f(t/N)$, where N is the visit count to this state (states are (ρ_i, ρ_j) transitions in our example). If f is a concave increasing function (e.g. $f(x) = x^{0.5}$) then in a bandit setting

$$\lim_{T\to\infty} \sum_{t=0}^{T} r_t/T$$

is maximized when states are visited uniformly. In an MDP setting where you can't achieve exactly uniform exploration, the maximum of this reward will be determined by the choice of f.

For Mazebase experiments (with a small graph), we found that the standard count-based reward of $N^{-0.5}$ actually performed worse than a random policy, while a reward of

$$\left(\frac{N}{t} + 0.001\right)^{-0.5}$$

(which is a smoothed version of $(t/N)^{0.5}$ to prevent large rewards which can destabilize training) outperformed random exploration. In the crafting environment, a random agent finds 18.6 edges on average, while the exploratory policy finds all 25 edges.

In StarCraft, which had a much larger graph with at least 400K edges, we found that $N^-0.5$ actually worked fine for exploration, discovering more than 50x more edges than a random policy.

B. Details of the Block Stacking experiment

The policy network π takes (i) a 128×128 image, which is featurized by a CNN with five convolutional layers and one fully connected (fc) layer to produce a 128d vector; and (ii) goal properties expressed as a 48d binary vector, which are transformed to a 128d vector. The two 128d vectors are concatenated and combined by two fc layers followed by softmax to produce an output distribution over actions. We use an exponential linear nonlinearity after each layer.

For the underspecified task, we consider the same distribution of tasks as the multi-step task, but provide only 70% of the attributes as the goal, at random. For the baseline models, we train them with the same distribution of underspecified goals as they see at test time, but this is not necessary for the AP model since it can plan over all goals that satisfy the underspecified goal.

Tables 5 and 6 show the performance of AP and baselines on both one-step and multi-step evaluation tasks, in 3×3 and continuous action spaces, respectively. In the continuous case, blocks can be dropped in any (x, y) with a fixed height $z = 1.5$ and $x, y \in [0, 1]$. The action space consists of choosing a discrete block id $b_{id} \in [0, 1, 2, 3]$ and continuous x, y. The inverse one-step models were trained on 2 million examples, inverse multi-step and AP models were trained on 1 million examples, and A3C models were trained to convergence (approximately 10 million examples). We perform each evaluation task 1000 times.

C. Details of the Mazebase experiment

The policy network is a fully connected network with two hidden layers of 100 units. The policy is trained with the Reinforce algorithm (Williams, 1992) to reach a neighboring set of attributes from the current state. Each round of training episodes terminate when the task completes or after 80 steps (i.e. $t_{max} = 80$); and a reward of 1 is received if the goal is reached[5]. An additional reward of -0.1 is given at every step to encourage the agent to complete the task quickly. We run each experiment three times with different random seeds, and report the mean success rate.

Some crafting tasks are pre-solved because the randomly chosen target item can already be in the inventory. However, such tasks have no effect on training and are also removed during testing.

D. Details of the StarCraft experiment

The game initializes with 4 SCVs, a command center, and nearby ore mines. The other types of units can be constructed, such as barracks, marines, supple depots, engineering bays, and missile turrets. However, the policy only controls SCVs, a command center, and barracks. The available actions to the policy differ depending on the unit type:

- **SCVs:** movements in 4 cardinal directions, mine ore, build a barracks, build a supple depot (also build a engineering bay, build a missile turret in the large version)
- **Command center:** train a SCV

[5]Once the attribute detectors are learned, the reward is intrinsic: the agent considers a local task complete when *it* decides the attributes have changed appropriately

Model	Training Data	1-step %	multi-step %	4-stack %	1-step underspecified	multi-step underspecified
A3C	1-step	98.5	8.1	1.9	65.7	6.6
A3C	multi-step	2.6	0.0	0.0	5.3	0.0
A3C	curriculum	98.2	17.0	2.9	8.2	0.2
Option-Critic	1-step	33.3	0.6	1.0	34.9	1.2
Option-Critic	multi-step	15.9	0.2	0.5	32.9	1.7
Option-Critic	curriculum	32.7	0.4	0.9	32	1.0
Inverse	1-step	**100**	9.1	0.5	**98.8**	18.8
Inverse	multi-step	94.1	13.7	4.6	71.2	9.6
AP (no c_π)	1-step	74.5	29.7	62.2	81.8	28.1
AP	1-step	98.8	**66.7**	**98.5**	97.8	**63.5**

Table 5. Model comparison on block stacking task accuracy. Baselines marked 'multi-step' or 'curriculum' get to see complex multi-step tasks at train time. The Attribute Planner (AP) generalizes from one-step training to multi-step and underspecified tasks with high accuracy, while reinforcement learning and inverse model training do not. AP outperforms baselines even with a curriculum of tasks. Ablating the normalized graph transition table c_π degrades AP performance substantially on multi-step tasks due to aliasing. Inverse one-step model was trained on 2 million examples, inverse multi-step and AP models were trained on 1 million examples, A3C models were trained to convergence.

Model	Training Data	1-step %	multi-step %	4-stack %	1-step underspecified	multi-step underspecified
A3C	1-step	0.3	0.0	0.0	0.0	0.0
A3C	multi-step	0.0	0.0	0.0	0.5	0.0
A3C	curriculum	0.3	0.0	0.0	0.0	0.0
Option-Critic	1-step	13.8	0.0	**4.4**	14.6	0.0
Option-Critic	multi-step	14.4	0.0	0.0	13.1	0.0
Option-Critic	curriculum	15.2	0.3	1.3	15.2	0.1
Inverse	1-step	60.1	8.6	0.0	29.3	2.5
Inverse	multi-step	0.7	0.0	0.0	1.0	0.1
AP (no c_π)	1-step	56.6	13.6	0.0	40.0	10.3
AP	1-step	**64.1**	**17**	0.4	**45.3**	**12.5**

Table 6. Model comparison in the *continuous* block stacking task. Inverse one-step model was trained on 8 million examples, inverse multi-step and AP models were trained on 10 million examples, A3C models were trained to convergence.

- **Barracks:** train a marine

Each units observes its 64x64 surrounding area with resolution of 4. Every time step, a policy outputs an action for each unit independently by taking its observation and the current attributes as an input.

Although the exact placement of units can be of importance in the game, here we only focus on their count. Hence, the attributes are chosen to be the number of units and resources of each type. Since any single unit can't observe everything in the game, detecting attributes from the observation alone is impossible. Therefore, the attributes are given as a part of the observation.

Multi-step tasks are generated by picking a random number for each unit type, with exception of ore and SCVs. The upper limits of those random number are set between 2 and 4 depending on the unit type.

The same reinforcement training procedure as Section 4.2 is employed for the RL baselines. For curriculum training, the upper limits are linearly increased during the curriculum training to make learning easy. Both baselines are trained for 30M steps.

Each episode starts at the initial state of the game and lasts 500 steps. The exploration policy π_e is trained with reinforcement learning with an intrinsic reward similar to Mazebase, although we find that scaling by number of transitions is unnecessary so we just use $c_{\pi_e}(\rho_i, \rho_j)^{-0.5}$ at each transition is more effective in StarCraft. The exploration policy is trained for 16 million steps, followed by training π and c_π for 14 million steps, with $t_{max} = 50$.