Appendix

1 Preliminaries

1.1 Symbolic Planning

We assume that the agent starts with a domain knowledge (as described by [Gizzi *et al.*2021]) which is defined as $\Sigma = \langle \mathcal{E}, \mathcal{F}, \mathcal{S}, \mathcal{O} \rangle$, where $\mathcal{E}$ is a finite set of known entities within the environment such that $\mathcal{E} = \left\{e_1, \dots e_{|\mathcal{E}|}\right\}$. $\mathcal{F}$ is a limited set of known predicates with their negations. The predicate descriptors are characterized such that $\mathcal{F} = \left\{f_1(\odot), \dots f_{|\mathcal{F}|}(\odot)\right\}, \odot \subset \mathcal{E}$. For each predicate $f_i\left(\odot_i\right)$, there exists a negation of that predicate $\neg f_i(\odot)$, where $f_i(\odot), \neg f_i(\odot) \in \mathcal{F}$. $\mathcal{S}$ is the set of symbolic states in the world such that $\mathcal{S} = \left\{s_i \dots s_{|\mathcal{S}|}\right\}$. $\mathcal{O}$ denotes the set of known action operators such that $\mathcal{O} = \left\{o_1, \dots o_{|\mathcal{O}|}\right\}$. In order to define action operators, we use Planning Domain Definition Language (PDDL) [McDermott *et al.*1998]. Each operator is defined with a set of preconditions and effects, denoted $\psi_i, \omega_i \in \mathcal{F}$. The preconditions ψ_i of o_i indicate the predicates that must hold true before executing o_i, and the effects ω_i of o_i indicate the predicates that will be assumed to hold true after successful execution of o_i. The predicate descriptors composing both ψ_i and ω_i may moreover incorporate negations, demonstrating predicate descriptors that must be untrue before and after execution, respectively. We define a planning task as $T = (\mathcal{E}, \mathcal{F}, \mathcal{O}, s_0, s_g)$, in which $s_0 \in \mathcal{S}$, is the set of starting states and $s_g \in \mathcal{S}$ is the set of goal states. The solution to the planning task T is the plan $\mathcal{P} = \left[o_1, \dots o_{|\mathcal{P}|}\right]$.

1.2 Reinforcement Learning

The RL problem is formalized as a Markov Decision Process (MDP) that is represented by a tuple $\langle \tilde{\mathcal{S}}, \mathcal{A}, \tau, R, \gamma \rangle$. At every timestep t, the RL agent receives a state representation $\tilde{s}_t \in \tilde{\mathcal{S}}$ (in practice the access to the "true" state may be impossible therefore, we use observations as an approximation) and explores an unknown environment by taking action $a_t \in \mathcal{A}$. A reward $r_{t+1} \in R$ is provided based on the action taken by the agent to reach the next state $\tilde{s}_{t+1}$. The agent learns to maximize the expected return value $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ for every state at time t. The discount factor $\gamma \in [0, 1)$ determines the importance of immediate and future rewards.

2 Additional Algorithms

Algorithm 1 ***RewardFunctionGenerator*** $(\Sigma, \mathcal{P}, o_i) \rightarrow \mathcal{S}_r$

$\Psi = \{o_1^{\psi_1,\psi_2,\cdots}, o_2^{\psi_1,\psi_2,\cdots}, \ldots\}$: preconditions of all the known operators
$\Omega = \{o_1^{\omega_1,\omega_2,\cdots}, o_2^{\omega_1,\omega_2,\cdots}, \ldots\}$: effects of all the known operators
$\mathcal{S}_r \rightarrow \emptyset$
for o_j in *reversed*$(\mathcal{P})$ **do**
  if $o_j \neq o_i$ & $\Psi_{o_j} \neq \Omega_{o_i}$ **then**
    $\mathcal{S}_r \cup \{\Psi_{o_j}\}$
    for $\omega_{o_j} \in \Omega_{o_j}$ **do**
      if $\omega_{o_j} \subseteq \mathcal{S}_r$ **then**
        $\mathcal{S}_r \setminus \{\omega_{o_j}\}$
      end if
    end for
  end if
  $\mathcal{S}_r \cup \{\Omega_{o_i}\}$
end for
return $\mathcal{S}_r$ =0

3 Novelty descriptions

In this section we describe the details of each novelty.

3.1 Rubber tree

In this novelty, as described in Table 2 the agent cannot extract *rubber* from the regular trees anymore, but a new entity *rubber-tree* appears in the environment and the agent needs to use the *rubber-tree* for *rubber* extraction. We have two versions of this novelty, in the *easy* version, the agent needs to select the *tree-tap* and standing in front of a *rubber-tree* it needs to use the action *extract-rubber* to extract rubber. In the hard version, the agent's action space is augmented with a new action called *place-tree-tap*, and in order to get rubber, the agent now needs to place the *tree-tap* in front of the *rubber-tree* and *rubber* can only be extracted when the agent is in front of the placed *tree-tap*. The agent's knowledge base does not have any *in front of* predicate. The agent's executor learns the policy to place the new entity of *tree-tap*, locate itself in front of it, and then extract *rubber*. We observe that the agent is successful in synthesizing executors with predicates missing from the knowledge base.

3.2 Axe to break

In this novelty, a novel entity *axe* appears in the environment and the agent cannot break *trees* without holding the *axe*. We implement two versions of the novelty, easy and hard. In the easy version, the *axe* is present in the agent's inventory, and the agent needs to learn to select the *axe* and use it in front of the *tree* to get *tree-log* into its inventory. In the hard version, the *axe* is present in the environment, and the agent needs to find it to get into its inventory and then use it to break *trees*. We observe that the agent is able to generate executors that can reason about the affordances of new entities in the environment, i.e. learn to pick up the *axe*, select it, and then *break* the *tree* by locating itself next to the *tree*.

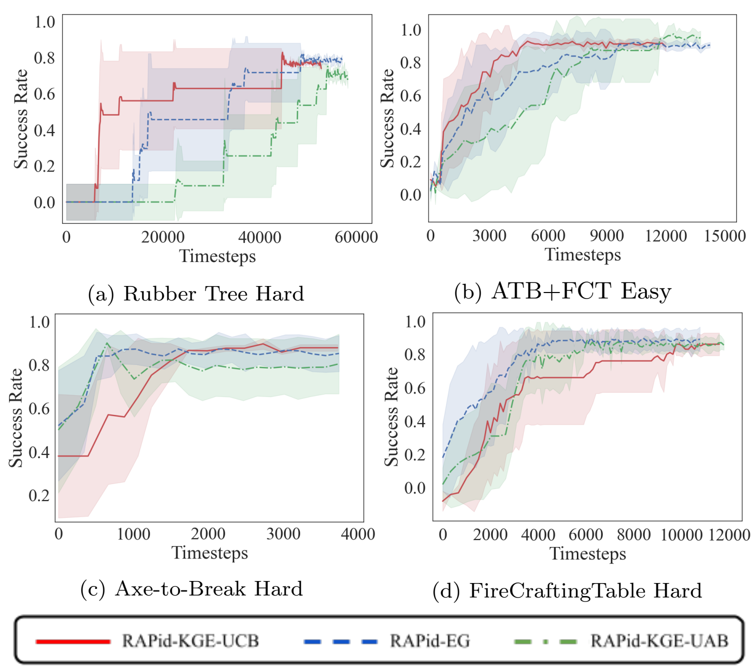
3.3 Fire crafting table

In this novelty, the agent cannot access the *crafting-table* since its set on *fire*. Two novel entities *water* and *fire* appears in the environment, and novel actions *spray* and *select-water* are added to the list of available actions. In order to access the *crafting-table* the agent needs to select *water* and use the action *spray* in front of the *crafting-table* in order to remove it from *fire*, and hence successfully access it. In the easy version, the *water* is already present in the agent's inventory and in the hard version *water* is present in the environment and the agent needs to collect it to use it. The agent's sub-symbolic representation has information about the status of the *fire* (on/off). The agent's learned executor in successfully dealing with more than one novel entities, and also novel predicates that are absent in the agent's knowledge base.

3.4 Scrape plank

In this novelty, the *break* action has no effect, and a novel action *scrape-plank* is augmented in the agent's action space. The dynamics of the world changes in a way that the agent can no longer break *trees* to get *tree-logs*, but it can scrape planks from trees and using the action *scrape-plank* in front of a *tree* it gets 4 *planks* into its inventory. This novelty emphasizes on failure of effects of the operator rather than failure of the action itself. The executor learns a policy to generate *planks*, by utilising the *scrape-plank* action to move ahead in the plan. Thus, the agent learns to tackle a novel scenario even when there are no new entities in the environment.

4 Results on the various versions of ***Knowledge-guided-exploration*** function



(a) Rubber Tree Hard

(b) ATB+FCT Easy

(c) Axe-to-Break Hard

(d) FireCraftingTable Hard

Figure 1: These plots illustrate the performance of the proposed RAPid-Learn methods during the *Learn* phase of the associated novelty. The baseline methods take orders of magnitude more timesteps to converge, We therefore plot their *Learn* phase performance separately (Appendix section 2).

Fig 1 compares the learning curves different types of *knowledge-guided-exploration* functions described in this paper. In the case of single novelty injection experiment, in this case, the *FireCraftingTable Hard* novelty injection, we observe that RAPid-KGE-UAB performs the best among all the agents, and converges to a post-novelty performance of 95%. In the case of explicit multiple operator failures, the ATB+FCT Easy novelty, we notice that the knowledge-guided-exploration using upper confidence bounds (RAPid-KGE-UCB) aids in adapting to the task quicker than the other versions of RAPid-Learn. Finally, for implicit multiple operator failures, in the Rubber Tree Hard novelty, we see that RAPid-Learn learns to adapt to the novelty in about 28000 timesteps. Figure 3a shows that RAPid-KGE-UCB is significantly helpful in adapting to this novelty.

All the versions of RAPid-Learn learn significantly faster than the baselines, and successfully learn a robust and a sample efficient policy.

5 Baseline Approaches

5.1 Prenovelty performance

Pre-novelty		
Agent	**Time to learn (timesteps)**	**Pre-novelty performance (success rate %)**
		Mean $\pm$ SD
Actor-Critic Transfer	2.47×10^5	0.95 ± 0.22
Policy Reuse	1.48×10^6	0.95 ± 0.22

Table 1: Performance and timesteps trained for the best pre-novelty model that was later used in transfer learning the novelty environments

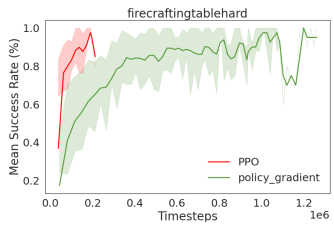
Table 1 shows how many timesteps the pre-novelty models have experienced until reaching convergence and their performance on 100 evaluation instances of the environment. Figure 2 compares the learning curves of these baselines.

5.2 State & Action Placeholders

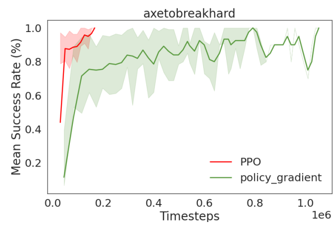
The injected novelties alter the shape of the observation and action spaces of the environment. E.g. the Fire Crafting Table novelty adds new observations for the *Water* item in the inventory and the LiDAR-sensor, as well as the *Select Water* and *Spray* actions. In order to allow transferring the model from pre-novelty to post-novelty environments, we pad the observation and action spaces of all environments (pre- and post-novelty) with placeholder elements such that all of them have the same shape. Observation placeholders simply always return zero. When the model chooses to perform a placeholder action, the action actually passed on to the environment is always the *Approach Treelog*.

5.3 Reward Shaping

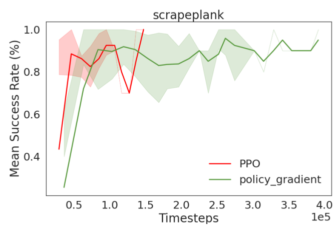
In early experiments, the pre-novelty task with sparse rewards proved intractable for pure RL approaches. Moreover, the RAPid-Learn agent has access to the full



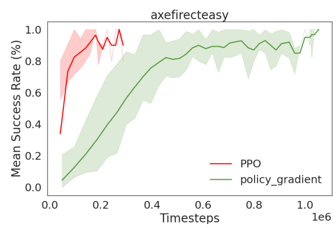
(a) Fire-crafting-table hard



(b) Axe-to-break hard



(c) Scrape Plank



(d) Axetobreak+FCT(easy)

Figure 2: Baseline (Policy Reuse and Actor-Critic Transfer) learning curves on the four novelties. Each plot demonstrates the performance during the *Learn* phase.

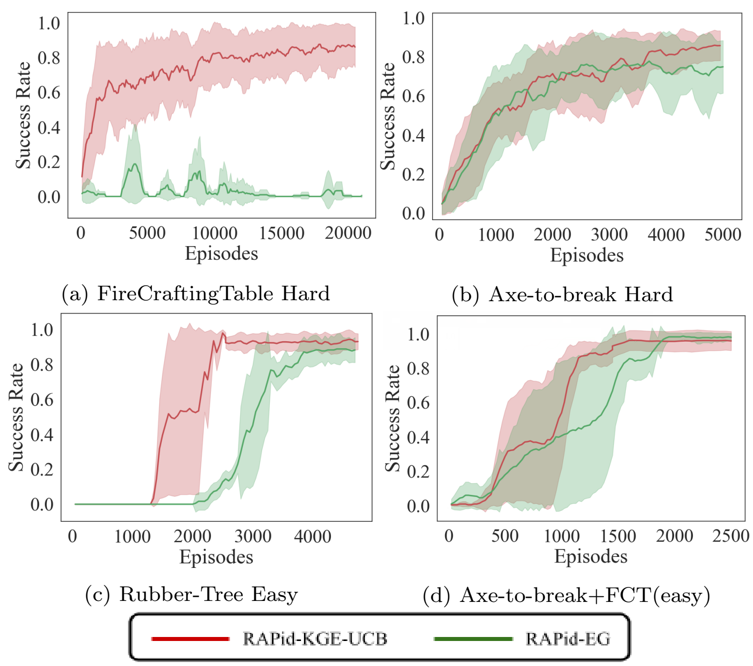
PDDL description of the environment, which heavily biases any comparison in its favor. We therefore employ strong reward shaping to allow the RL baseline to learn the prenovelty task. A small reward (+50) is given for completing intermediate steps (e.g. crafting sticks or planks) when they are needed for the next major step in the crafting process. Larger rewards are given for the subgoals of crafting the treetap (+200), extracting rubber (+300) and finally crafting the pogostick (+1000). Note that the rewards are only given if the intermediate step is the appropriate thing to do next. I.e. no reward is given for breaking a tree, if the agent already has a treetap and enough sticks and planks in its inventory to craft the pogostick. In this case, a reward would only be given for obtaining the missing ingredient (rubber). Making the reward conditional on the stage within the crafting process was essential in getting the agent to learn the pre-novelty task. In early versions of the reward shaping, we attempted to give reward for completing intermediate steps (approaching a tree, crafting sticks, etc.). This was exploited by the agent, which learned that the reward could be optimised by periodically turning away and re-approaching a tree until the end of its life.

6 Ablation study

To examine the impact of hierarchical actions, we perform experimental evaluation in absence of hierarchical actions. For our experiments, hierarchical actions are defined by the (`approach`) operator, followed by the entity to approach. This hierarchical action is composed of the primitive actions (`turn` left, turn right and move forward). An example of an hierarchical action is `approach tree_log`, where the agent determines the location of one of the `tree_log` in the environment and formulates a plan to reach the entity. This plan is computed using the A^* algorithm [Hart *et al.*1968].

Incorporation of hierarchical actions simplifies novelty adaptation, as the agent has the opportunity to accommodate the novelty in fewer interactions with the environment. Fig 3. We can observe that for the complex *fire-crafting-table hard* novelty, the *RAPid-EG* approach fails to converge to a successful policy. The difficulty for the agent to successfully complete the task can be attributed to two reasons:

- The agent has to follow a series of movement actions to first collect the *water* in its inventory, and then to reach the *crafting table* where it needs to spray the *water* by holding it, in order to diffuse the fire.
- While it performs the above mentioned steps, it should avoid reaching the state from which it cannot plan again. This will happen if the agent crafts crafts extra sticks in its inventory, making it impossible to craft a pogo stick with the resources available in the environment and in its inventory.



(a) FireCraftingTable Hard
(b) Axe-to-break Hard
(c) Rubber-Tree Easy
(d) Axe-to-break+FCT(easy)

Figure 3: The plots show the learning curves for four novelties. Each plot demonstrates the performance during the *Learn* phase.

In absence of any informed exploration, the baseline approach 'RAPid-EG' takes longer to converge to a successful policy in absence of hierarchical actions. Thus, we can see that hierarchical actions play a crucial role in increasing the sample efficiency of the outlined approach.

7 Hyperparameters

7.1 Convergence criteria

To ensure that the learner has learned to reach the goal, we verify if the agent achieved the goal $\delta_g \geq \delta_G$ in the last η episodes with an average reward $\delta_r \geq \delta_R$. Furthermore, to ensure the agent has converged, and is not learning any further, we verify if the success rate for the agent in the past $\eta + \upsilon$ episodes is equal to the success rate for the agent in the past η episodes.

7.2 Baseline Hyperparameters

Parameter	**Value**
learning_rate	$3e-4$
batch_size	64
n_epochs	10
n_steps	2048
gamma	0.98
gae_lambda	0.95
clip_range	0.2
max_grad_norm	0.5
ent_coef	0.0
vf_coef	0.5
optimiser	Adam

Table 2: Hyperparameters used for the PPO models (both pre- and post-novelty).

See Table 2 for an overview of the hyperparameters used to train the Actor-Critic Transfer baselines. Note that with the exception of gamma, these are the default parameters used in [Raffin *et al.*2019]. The Policy Reuse baseline models utilise the exact same implementation that underlies the Learner in RAPid-Learn, with identical hyperparameter settings, as shown in Table 3

7.3 RAPid-Learn Hyperparameters

See Table 3 for the hyperparameters used in the RAPid-Learn experiments.

8 Statistical significance

To demonstrate that the average success rate of RAPid-Learn is consistently higher than the baseline approaches, we perform an unpaired t-test. The data is generated from 10 trials for each method, and in each trial, the performance of 100 post-novelty episodes is measured. For the experiment, we consider a confidence interval of 95% and evaluate the t-value between the best performing

Parameter	Value
max-epsilon (ϵ_{max})	0.3
min-epsilon (ϵ_{min})	0.05
guided curriculum parameter ($\rho_{\max}$)	0.3
guided curriculum parameter ($\rho_{\min}$)	0.05
parameter decay speed	ln(0.01)/2000
UCB- parameter (c)	0.0005
UAB-parameter(μ)	2
update_rate	10
max episodes (e_{max})	100000
max timesteps (T)	300
Hidden Layers	24 (single layer network)
Discount factor (γ)	0.98
learning rate (α)	$1e-3$
reward threshold (δ_R)	900
episodes threshold (δ_G)	100
no of success (η)	96
positive reinforcement (ϕ_1)	1000
negative reinforcement (ϕ_2)	-350

Table 3: Hyperparameters for RAPid-Learn

RAPid-Learn approach and the two transfer learning baseline approaches (PPO-TL, PG-TL). Table 4 shows the results of the unpaired t-test. Thus, through the results, we see that our proposed approach, RAPid-Learn has a consistent performance in the post-novelty success rate. The results are always statistically significant, except in the case of the *Scrape Plank* novelty. In all the novel scenarios, RAPid-Learn has a much more sample efficient performance. Thus, RAPid-Learn not only achieves a better success rate, but also adapts to novel scenarios efficiently.

Novelty	Methods	p-value	statistical significance
ATB-Hard	RAPid-KGE-UCB $\leftrightarrow$ PPO-TL	0.00336	YES
ATB-Hard	RAPid-KGE-UCB $\leftrightarrow$ PG-TL	0.0012	YES
ATB+FCT-Easy	RAPid-KGE-UCB $\leftrightarrow$ PPO-TL	0.0062	YES
ATB+FCT-Easy	RAPid-KGE-UCB $\leftrightarrow$ PG-TL	0.0012	YES
FCT-Hard	RAPid-KGE-UAB $\leftrightarrow$ PPO-TL	< 0.0001	YES
FCT-Hard	RAPid-KGE-UAB $\leftrightarrow$ PG-TL	< 0.0001	YES
Scrape Plank	RAPid-EG $\leftrightarrow$ PPO-TL	0.0077	YES
Scrape Plank	RAPid-KGE-UCB $\leftrightarrow$ PG-TL	0.0608	NO

Table 4: Results of unpaired t-test between best performing RAPid-Learn and transfer learning baselines

References

[Gizzi *et al.*2021] Evana Gizzi, Amel Hassan, Wo Wei Lin, Keenan Rhea, and Jivko Sinapov. Toward creative problem solving agents: Action discovery through behavior babbling. In *ICDL*, pages 1–7, 2021.

[Hart *et al.*1968] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[McDermott *et al.*1998] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language, 1998.

[Raffin *et al.*2019] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2019.