Advances in Cognitive Systems X (20XX) 1-6 Submitted X/20XX; published X/20XX

Transforming Environments to Evaluate Agent Adaptation

Dustin Dannenhauer DUSTIN.DANNENHAUER@PARALLAXRESEARCH.ORG
Noah Reifsnyder NOAH.REIFSNYDER@PARALLAXRESEARCH.ORG
AJ Regester AJ.REGESTER@PARALLAXRESEARCH.ORG
Matthew Molineaux MATTHEW.MOLINEAUX@PARALLAXRESEARCH.ORG
Parallax Advanced Research Corporation, Beavercreek, OH 45431 USA

Abstract

There is considerable recent interest in agents that can achieve goals and maximize performance in transformed environments. The ability of an agent to maintain performance after an environment transformation is sometimes referred to as *novelty robustness*. However, current test beds for open-world novelty require domain-specific processes to modify environment dynamics to achieve transformations that both require and permit agent adaptation; these processes are time consuming and biased. We introduce a novel, domain-independent environment transformation generator that randomly selects meaningful transformations meeting these requirements. We claim that this generator produces transformations that are *learnable*: i.e., after such a transformation occurs to its environment, a suitable learning agent can improve its performance online in the environment without human intervention. First, the generator creates permutations of an existing environment model, then it filters them to find those transformations that are relevant to, noticeable to, and controllable by the agent. We report the following contributions: a domain-independent algorithm and implementation for automatically generating environment transformations, an analysis of the space of possible transformations, and experimental evidence that the transformations selected by our generator are learnable.

1. Introduction

AI systems fail during deployment because real-world environments are unpredictable. The problem of responding to “unknown unknowns" that may crop up in an agent’s environment is large; we refer to this as open-world novelty, and consider evaluating an agent’s response to it. Specifically, this type of novelty concerns changes (or apparent changes) to the structure or dynamics of the environment an agent inhabits, referred to as an *environment transformation*. Recent work emphasizes robustness and adaptation to such transformations (McLure & Musliner, 2022). From an evaluation perspective, existing studies create ad hoc examples of environment transformations that are human-designed, and therefore biased. It is not clear that robustness to human-generated transformations indicates the ability to respond to unknown unknowns. To better evaluate these capabilities, we introduce the first automated *novelty generator* that randomly generates environment transformations to study open-world novelty.

We claim that novelty can be found by sampling a space that is too large to fully enumerate. Our generator searches through a space of environment transformations for examples that are *learnable*; that is to say, they meet certain basic criteria that allow and motivate an agent to learn: relevance, noticeability, and controllability. We focus here on the space of learnable novelties, as the space of unlearnable transformations are uninteresting. By definition, to be unlearnable they would have no affect on agents operating in the environment. Learnable environment transformations challenge an agent to respond differently than in their original environments. Our approach to novelty generation is consistent with Wiggins' (2006) Creative Systems Framework (CSF). To our knowledge, our novelty generator is the first instantiation of CSF for relational state-action environment creation.

Other work in AI sometimes uses the term "novelty" to refer specifically to novel agent experiences (e.g., Muhammad et al, 2020; Boult et al, 2021; Gamage et al, 2021); this perspective is valid, but we intentionally avoid any reference to an agent's knowledge in this work so as to avoid experimentation issues across agents with different knowledge representations.

In this paper, we describe a space of transformations over PDDL environments, and a novelty generator that samples that space to find challenging problems in open-world novelty. To demonstrate the success of this algorithm, we present experimental results describing the change in performance of a replanning agent that attempts to achieve goals in the presence of such transformations. We claim that the novelty generator selects transformations without bias, and that the transformations selected present a challenge to an agent.

2. Defining Environment Transformations

We describe environment transformations over a space of discrete-time, turn-taking environments. An environment σ can be described as a tuple $(S, A, O, Ag, Tu, \gamma, \omega)$ where S is the space of possible states, A is the space of possible actions, O is the space of observations, Ag the space of agents, Tu is a turn function $S \rightarrow Ag$ that determines which agent acts in a given state, γ is the probabilistic transition function $S \times A \times S \times Ag \rightarrow \mathbb{R}$, and ω is the probabilistic observation function $S \times O \times Ag \rightarrow \mathbb{R}$. The agent itself is an argument to the observation and transition functions, as different agents may have different perceptual interfaces and action capabilities. During environment interaction, each actor $\alpha \in Ag$ receives an observation o when it enters a state s according to probability $\omega(s, o, \alpha)$. Then, one environment-selected agent $Tu(s)$ selects an action a, and the environment transitions to a new state s' with probability $\gamma(s, a, s', Tu(s))$. The probability of transition due to an action from any other agent is 0:

$$\alpha \neq Tu(s) \implies \gamma(s, a, s', \alpha) = 0 \tag{1}$$

An environment transformation, or novelty, is a tuple $(\sigma, \sigma^M, \alpha)$ that describes two environments that produce different experiences for an agent. Here, σ is the original environment, $\sigma^M = (S^M, A^M, O^M, Ag^M, Tu^M, \gamma^M, \omega^M)$ is a modified environment, and α is an agent that can interact with either environment. To produce different experiences, the environments' transition functions (γ, γ^M) must differ, or their observation functions (ω, ω^M) must sometimes produce different observations *for agent* α. While we include the agent α in our definition of novelty $(\sigma, \sigma^M, \alpha)$, we

stress that the only change that occurs is between σ and σ^M and therefore, novelty as its defined here, is environment-based and **not** agent-based. Other theories of agent-based novelty distinguish whether observations are novel based on the agent's internal AI approaches (i.e. what is novel for an reinforcement learning agent may not be novel for a classical planning agent). In our formalism, environments refer to agents but do not refer to those agents' policies.

3. Transformation Space Description

We formally define and implement 24 transformations on PDDL+ domain descriptions; these transformations are the mechanism by which a new, novel domain description is produced. A summary of these transformations and a lower bound on the size of the space of novelties they produce are given in Table 2. Each transformation yields a number of novel domains; this value is given in the *Number of Instances* column. We argue that since it is unrealistic to expect a human to enumerate all transformations to an environment, the space of new environments created by performing these transformations will contain novelty. Applying multiple transformations to an original environment yields a space of novelties that is significantly large, increasing in size with each iteration of applying an additional transformation. While the space of novelties given for most transformations is straightforward, we highlight a few individual transformations here: *addDerivedPredicate* has $2 \cdot |\binom{C}{j}| \cdot 2 \cdot (j-1)!$ instances because $\binom{C}{j}$ are the conditions chosen for the preconditions, then the first 2 is because each condition can be positive or negated, and then the conditions are arranged in **and** and **or** (hence the second 2) and we will consider all combinations (explaining the !) of up to $j-1$ pairs of conditions; *addNewType* has an optional argument of *typeParent* and if no type parent is given, the number of instances is 1, similar to *addProcess*. In cases where a new predicate or type is added to the domain, the dynamics of the environment do not change; the added construct must be part of a state transition to affect an agent's experience (i.e. part of an action/event/process's preconditions or effects).

A lower bound number of new environments N produced by applying all transformations is:

$$\begin{aligned} 1 + |F| + |A| + |D| + 2|U| + 3|E| + (4+k)|T| + \\ |T \times T| + |U \times C| + |(A+E) \times C^{ex}| + \\ 2 \cdot |\binom{C}{j}| \cdot 2 \cdot (j-1)! + \\ 4|(A+E) \times C \times V| + \\ 2|\binom{C}{n} \times \binom{C}{m}| \end{aligned}$$

where each variable is defined in Table 2. Such a formula allows us to determine the space of transformed environments that can be produced by a single transformation. As an example, in Table 1 we take three domain files from the International Planning Competition in 2018, and show size of novel environments from a single applied transformation.

Key: Parameters	
m=2, n=2, k=2, j=2, v=100	
Domain	**Space of Novelties after 1 Transformation**
Data Network	1,570,323
Termes	42,794
Snake	48,573

Table 1. Estimate Lower Bound Space of Novel Domains for the Data Network, Termes, and Snake domains

4. Filtering to Find *Learnable* Novelties

It is easy to construct novelties that are clearly distinct but very difficult to learn about. For example, states may be different between two environments only in some subtle way that is not visible to an agent's sensors. Some unobservable novel event might cause an agent's immediate destruction, allowing no response and providing no useful information the agent could use to avoid that event in the future. Novel transitions might occur only in a distant region of the state space that the agent never encounters in practical situations. Therefore, we consider conditions that ensure an agent has an opportunity (and reason) to learn about an environment transformation. We say that a transformation is learnable if it changes the environment in the following ways:

Relevance: Affects an agent's performance

Noticeability: Causes different observations

Controllability: Rewards policies differently

To formally define these qualities, we require the following definitions and assumptions. A trajectory τ is a linked list where each element is either the empty trajectory $\emptyset$ or a tuple (a, o, τ'). Here, a is the agent's selected action if it is the agent's turn, or yield if it is not; o is the subsequent observation received; τ' is the remaining trajectory. We define an extension of a trajectory τ with new action a_n and observation o_n as a copy of that trajectory with a new final element $(a_n, o_n, \emptyset)$:

$$extension(\emptyset, a_n, o_n) \equiv (a_n, o_n, \emptyset)$$
$$extension(\tau = (a, o, \tau'), a_n, o_n) \equiv$$
$$(a, o, extension(\tau', a_n, o_n))$$

The length of a trajectory τ is defined as 0 for $\emptyset$, and 1 + the length of the subtrajectory for all others:

$$length(\emptyset) \equiv 0$$
$$length(\tau = (a, o, \tau')) \equiv 1 + length(\tau')$$

Due to nondeterminism, an agent α with policy π and initial state s may experience any one of multiple possible histories τ of any given length. We define an agent's decision-making policy

Key: Variables for PDDL+ Domain Constructs	
F is the set of functions V is the set of values for a numeric range T is the existing set of types P is the existing set of predicates A is the existing set of actions E is the existing set of events D is the existing set of derived predicates U is the existing set of processes	C is the existing set of possible conditions, where a condition is a, possibly negated, predicate or fluent statement: $\|C\| = 2 \cdot ((3 \cdot \|F\|) + \|P\|)$. The 3 in $(3 \cdot \|F\|)$ reflects that each fluent condition may have a $<$, $>$, or $=$ comparison (we do not assume nested fluent conditions, which while possible, results in an infinite number of conditions). C^{ex} is the set of existing preconditions and effects across all existing actions and events.

Key: Transformation-specific Parameters

n: preconditions; m: effects; k: arguments for new predicate; j: upper bound on conditions

ID	Transformation Function Signature	Number of Instances
1	changeEffectProbability(actionOrEvent, effect, value)	$\|(A+E) \times C \times V\|$
2	changeTypeParent(typeChild, typeParent)	$\|T \times T\|$
3	addEffect(actionOrEvent, effect, probability)	$\|(A+E) \times C \times V\|$
4	addPrecondition(actionOrEvent, precondition)	$\|(A+E) \times C \times V\|$
5	addNewAction(name, parameters, preconditions, effects)	$\|\binom{C}{n} \times \binom{C}{m}\|$
6	addNewEvent(eventName, parameters, pre, eff, freqDist)	$\|\binom{C}{n} \times \binom{C}{m}\|$
7	addNewPredicate(newPredicateName, arguments)	$k\|T\|$
8	addFunction(newFluentName, existingType)	$\|T\|$
9	addNewType(newTypeName, typeParent)	$\|T\|$
10	removePreconditionOrEffect(action/event, condition)	$\|(A+E) \times C^{ex}\|$
11	removeAction(action)	$\|A\|$
12	removeEvent(event)	$\|E\|$
13	removeFluent(fluent)	$\|F\|$
14	removeType(typeName)	$\|T\|$
15	addDerivedPredicate(name, pred expression)	$2 \cdot \|\binom{C}{j}\| \cdot 2 \cdot (j-1)!$
16	changeFluentEffectVal(action/event, prec/effect, val)	$\|(A+E) \times C \times V\|$
17	changeFrequencyDistributionEvent (event, newValue)	$\|E\|$
18	removeDerivedPredicate(derivedPredicateName)	$\|D\|$
19	removeProcess(processName)	$\|U\|$
20	addConstant(name, type)	$\|T\|$
21	addProcess(processName)	1
22	changeProbability(eventName, probability)	$\|E\|$
23	addProcessCondition(processName, condition)	$\|U\| \times \|C\|$
24	addProcessChange(processName, change)	$\|U\|$

Table 2. Environment Transformations.

$\pi : \tau \rightarrow A$ as a mapping from the space of trajectories to an action to take at the end of that

trajectory. The probability $p(\tau|\tau^P, s, \sigma, \pi, \alpha)$ of an agent α experiencing a particular trajectory τ using policy π starting in state s with history τ^P in environment σ is given by:

$$\begin{aligned}
&p(\emptyset|\tau^P, s, \sigma, \pi, \alpha) = 1 \\
&p(\tau = (a, o, \tau')|\tau^P, s, \sigma = (S, A, O, Ag, Tu, \gamma, \omega), \pi, \alpha) \\
&\quad = \sum_{s' \in S} \begin{array}{l} \gamma(s, \pi(\tau^P), s', Tu(s)) \\ * \, \omega^M(s', o, \alpha) \\ * \, p(\tau'|extension(\tau, a, o), s', \sigma, \pi, \alpha) \end{array}
\end{aligned}$$

We assume that an agent is motivated by some environment-independent performance measure that is known or observable to it. This function, *Performance:* $T \times S \rightarrow \mathbb{R}$ is some mapping from the space of possible trajectories T and actual states traversed to the real numbers; this function is calculated over the full sequence of states and actions in the agent's past trajectory. The space of final states of an environment σ, $S_F(\sigma) \subset S$ is defined as the set of states which have no outgoing transitions:

$$\begin{aligned}
&S_F(\sigma = (S, A, O, Ag, Tu, \gamma, \omega)) \equiv \\
&\quad \{s \in S \mid \forall a \in A, s' \in S : \gamma(s, a, s') = 0\}
\end{aligned} \tag{2}$$

We define the **E**xpected **P**erformance of a **P**olicy (EPP) in an environment based on a past trajectory τ^P, state history $\mathbf{s}$, and policy π as the performance of the past trajectory in any final state, or otherwise as the average expected performance of the policy over each possible next state and observation as given by recursion over the next state and updated trajectory.

$$\begin{aligned}
&EPP(\tau^P, \pi, \mathbf{s} = (s_0 ... s_n \in S_F(\sigma)), \alpha, \sigma) \equiv \\
&\quad Performance(\tau^P, s)
\end{aligned}$$

$$\begin{aligned}
&EPP(\tau^P, \pi, \mathbf{s} = (s_0 ... s_n \notin S_F(\sigma)), \alpha, \\
&\qquad \sigma = (S, A, O, Ag, Tu, \gamma, \omega)) \equiv \\
&\sum_{s' \in S, o \in O} \begin{pmatrix} \gamma(s_n, \pi(\tau^P), s', \alpha) \\ * \, \omega(s', o, \alpha) \\ * \, EPP(extension(\tau^P, \pi(\tau^P), o), \\ \qquad \pi, concat(\mathbf{s}, s'), \alpha, \sigma) \end{pmatrix}
\end{aligned} \tag{3}$$

Formally, an environment transformation (σ, σ^M, α) is relevant if there is some policy π that produces different expected performance starting in some state s_0 in σ and σ^M.

$$Relevant(\alpha, \sigma, \sigma^M) \equiv$$
$$\exists \pi, s_0 : EPP(\emptyset, \pi, s_0, \alpha, \sigma) \neq EPP(\emptyset, \pi, s_0, \alpha, \sigma^M)$$

An environment transformation is noticeable if, from any starting state, there is some agent policy π that results in different trajectories in the modified environment σ^M than the original σ. Generally, this means that there is some set of trajectories T such that an agent will almost always experience a trajectory in T using π in σ^M, but almost never in σ. In any fully observable, deterministic environment, every environment transformation that maintains full observability and determinism is necessarily noticeable: the observation function cannot be changed without also changing the transition function, and any changed transition is immediately apparent.

$$\begin{aligned} Noticeable(\alpha, \sigma = (S, A, O, Ag, Tu, \gamma, \omega), \sigma^M) \equiv \\ \exists n, \pi, T : \\ \forall \tau \in T : length(\tau) = n \\ \wedge \sum_{s \in S^M} \sum_{\tau \in T} p(\tau | \emptyset, s, \sigma^M, \pi, \alpha) > 1 - \epsilon \\ \wedge \sum_{s \in S} \sum_{\tau \in T} p(\tau | \emptyset, s, \sigma, \pi, \alpha) < \epsilon \end{aligned}$$

An environment transformation (σ, σ^M, α) is considered controllable if there is some policy π' that performs better for α in the modified environment σ^M than the optimal policy π^* for the original environment σ.

$$\begin{aligned} Controllable(\alpha, \sigma, \sigma^M) \equiv \\ \exists \pi^* \, \forall \pi, s \in S : \\ EPP(\emptyset, \pi, s, \alpha, \sigma) \leq EPP(\emptyset, \pi, s, \alpha, \sigma) \\ \wedge \; \exists \pi' \, \forall s \in S^M : \\ EPP(\emptyset, \pi^*, s, \alpha, \sigma') \leq EPP(\emptyset, \pi', s, \alpha, \sigma') \end{aligned}$$

We strive to approximate these conditions so as to efficiently find learnable environment transformations.

5. Related Work

Unlike work that seeks to define novelty relative to an agent's experiences (e.g., Muhammad et al. (2021); Boult et al. (2021); Gamage et al. (2021), we intentionally avoid any reference to an agent's knowledge. Such frameworks are useful because they can identify how prepared an agent is for new challenges. However, they cannot usefully describe how environment challenges differ in a way that cuts across different agents and different knowledge representations. For historical and language reasons, both currently use the term "novelty", but in different ways.

Boult et al. (2021) defined a theory of open world novelty where novelty occurs when an agent experiences an environment sufficiently different from its prior experiences. This is fundamentally different from our definition in several ways; first, in our framework, novelty exists independent of any given agent's experiences or knowledge, for reasons given above. Second, our framework does not define novelty as occurring at a point in time, but rather as encompassing the time-independent difference between two environments. This means that novelty can exist without any changes to the observation or state space. While valuable for considering what experiences could be new to a particular agent, two agents cannot truly be compared on the "same" novelty unless they have identical prior experiences; such comparisons are a key feature of our framework.

Langley (2020)'s work provides a set of broad requirements for a theory of environmental change that can help explain and measure progress in open-world learning. He suggested that such a theory would propose a formalism for environments and transformations on them. Our framework provides both, and so can be consider an example "theory" in this regard that describes environmental changes and provides specific language for characterizing how environment changes vary.

Current benchmark domains [Goel et al. (2021); Xue et al. (2022); Kejriwal & Thomas (2021); Balloch et al. (2022)] test an agent's ability to respond to novelty using carefully constructed scenarios by domain experts. These testbeds provide novelties of varying difficulties and type. Almost all of these novelties are *learnable* (relevant, noticeable, and controllable) due to the human design process, yet remain limited in quantity and coverage. To our best knowledge, our testbed is the first to systematically provide infinite novelties that are not hand curated by domain experts. Instead our system randomly generates new environments and then filters them according to our learnability criteria: relevance, noticeability, and controllability.

Wiggins (2006)'s framework for creative systems in AI describes novelty as a property of a creative output which previously did not exist. While our description of environment transformation novelties makes no restriction on the generating process, it is related: part of the purpose of our framework is to describe constraints on the generation of pairs of environments pre- and post-transformation, where the later one has properties that did not previously exist in the earlier one. Thus we are broadly consistent with Wiggins' definition.

MacGyver problems (Sarathy & Scheutz, 2018) are those where the goal for an agent is unreachable with the agent's current domain model. The agent must update its model (e.g. by learning a new macro action) in order to solve the problem. When novelty occurs in the environment and an agent's goal is no longer achievable, such a scenario can be considered a MacGyver problem. Not all novelties are MacGyver problems, since a novel environment may still permit reaching the original goal. Extending our framework to generate MacGyver problems would be possible by adding an additional filter that, given a starting state and goal, selects a new environment where that goal, from the same starting state, is no longer achievable.

6. Empirical Results

6.1 Experimental Setup

In order to test the capabilities of our novelty generator, we designed a set of experiments to determine if the generated novelties are relevant. As a test-bed, we used the *data-network* domain from

the IPC competition. Since we were pulling from the IPC competition, we were provided with a domain file and 20 problem files. At this time our experimental setup only evaluates relevance; we leave noticeability and controllability for future work. The experimental procedure follows multiple steps:

Step 1: Read in the domain file for the domain, creating an objectified representation of the domain in python.

Step 2: Search over all the possible 1-step alterations on the domain that affect the transition model. This provides a total of 136 novelties for the *data-network* domain. There is some initial filtering during this generation stage that begins to guide the generation towards relevant novelties. The filtering at this stage looks only for new effects (using transformation 3 from Table 2) such that the arguments to the predicate or fluent being added can be matched to existing parameters in the action or event to which the effect is being added. Since we exclude no possible alterations during the generation process, we avoid bias' that come from human perceptions.

Step 3: After the novelties are generated, create a set of 10 problems in the domain. These problems are created from one of the given problem files of the IPC domain. We add in an additional set of init clause facts to the problem such that the novel action or event is executable from the initial state. This is done to make sure that the novel action has every chance to be taken so we have the best chance of witnessing if the novelty is relevant.

Step 4: With these generated problems, run the planning and simulation agent to test the novelty. The simulation always runs in the modified domain, however we generate 2 separate plans: one using the modified domain and one using the unmodified domain. We run both plans through the simulator, and collect the metric data. This metric value is either the defined metric for the domain, or if no metric is defined we use the number of actions executed. This default metric is used to prioritize shorter plan traces in environments where action costs are not defined. If the metric cost between the plan using the modified domain and the one using the unmodified domain differ for any of the 10 created problems, we say the novelty is relevant.

Step 5: After determining relevance (Step 4), we perform a verification test as well as data collection on the relevant novelties. Since the problems used during the determination process were biased, we now use the provided sample problems to verify the relevance on the novelty and acquire meaningful data. Consequently, we perform step 4 again, using the given problems instead of the generated problems, saving the difference in metric score for the plans made using the modified and unmodified domain. Thus, we represent the relevance of the novelty by the average difference in metric cost seen across the given problems. These scores can be seen in Figure 1.

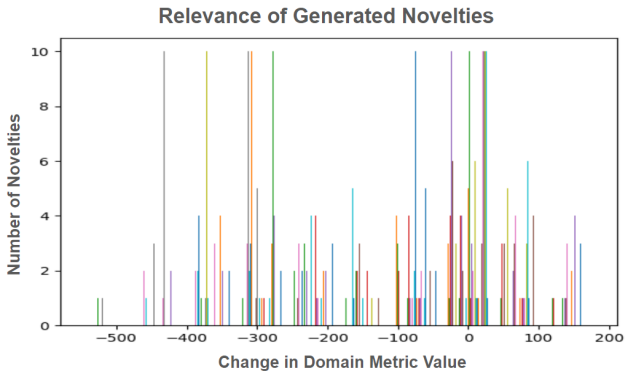

Figure 1. Showcasing the impact of novelties on the Data-Network domain from the IPC competition. The impact is measured by the change in metric value between the modified and unmodified domain

6.2 Discussion on Results

Analyzing Figure 1 gives us a few insights into the performance of our novelty generator. Firstly, we can see there is a wide variety of impact caused by the generated novelties. For context, the average metric cost of the unmodified domain was 647. There were a few novelties that made the given problems trivial to solve, a good number that made the problems easier, a lot that had a small impact, and a few that made the problems slightly harder. While this distribution is specific to this domain, it still showcases the variety of impacts provided by our generator.

Secondly, while its hard to see in this graph, none of the novelties has 0 impact on the verification problems. This means that the novelties outputted by our generator all had at least some impact on an agent acting in the domain. It should be noted however, that there are likely many relevant novelties that our generator is currently not finding.

Thirdly, in this paper we only report results on relevance of novelties generated. Future work is needed to provide tests for controllability and noticeability. Currently, the domains we are using are fully observable, so all novelties are noticeable.

7. Conclusions

We introduce the first automated, domain-independent *novelty-generator* capable of generating infinite environment-based novelties for relational, state-action environments. Our formalism defines *learnable novelties* as those that are **relevant** to an agent, **noticeable**, and **controllable**. We provide an experimental setup to automatically generate novel environments to obtain new learnable novelties independent of human bias. Using the *data-network* IPC domain, we evaluated a planning agent on 136 automatically generated novelties, showing that all of these novelties were relevant.

While we believe important steps have been taking with generating relevant novelties, there are paths forward still to explore. While testing relevance, our current methods focused on novelties that affected the transition model. We would like to expand our filtering methods to other novelty types. Besides relevance, we would also like to collect data on the controllability of the generated

novelties. Controllability in short is the ability of the agent to make use of the novelty introduced in the domain. We would also like to expand our testing to more domains, which requires a larger representation language. Currently, the language used by the novelty generator can not represent all PDDL domains, and that is a future goal.

References

Balloch, J., Lin, Z., Hussain, M., Srinivas, A., Wright, R., Peng, X., Kim, J., & Riedl, M. (2022). Novgrid: A flexible grid world for evaluating agent response to novelty. *arXiv preprint arXiv:2203.12117*.

Boult, T., et al. (2021). Towards a unifying framework for formal theories of novelty. *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 15047–15052).

Gamage, C., Pinto, V., Xue, C., Stephenson, M., Zhang, P., & Renz, J. (2021). Novelty generation framework for ai agents in angry birds style physics games. *2021 IEEE Conference on Games (CoG)* (pp. 1–8). IEEE.

Goel, S., Tatiya, G., Scheutz, M., & Sinapov, J. (2021). Novelgridworlds: A benchmark environment for detecting and adapting to novelties in open worlds. *International Foundation for Autonomous Agents and Multiagent Systems, AAMAS*.

Kejriwal, M., & Thomas, S. (2021). A multi-agent simulator for generating novelty in monopoly. *Simulation Modelling Practice and Theory*, *112*, 102364.

Langley, P. (2020). Open-world learning for radically autonomous agents. *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 13539–13543).

McLure, M. D., & Musliner, D. J. (2022). A changepoint method for open-world novelty detection. *IGARSS 2022-2022 IEEE International Geoscience and Remote Sensing Symposium* (pp. 5329–5332). IEEE.

Muhammad, F., Sarathy, V., Tatiya, G., Goel, S., Gyawali, S., Guaman, M., Sinapov, J., & Scheutz, M. (2021). A novelty-centric agent architecture for changing worlds. *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems* (pp. 925–933).

Sarathy, V., & Scheutz, M. (2018). Macgyver problems: Ai challenges for testing resourcefulness and creativity. *Advances in Cognitive Systems*, *6*, 31–44.

Wiggins, G. A. (2006). A preliminary framework for description, analysis and comparison of creative systems. *Knowledge-Based Systems*, *19*, 449–458.

Xue, C., Pinto, V., Zhang, P., Gamage, C., Nikonova, E., & Renz, J. (2022). Science birds novelty: an open-world learning test-bed for physics domains.