NovelGridworlds: A Benchmark Environment for Detecting and Adapting to Novelties in Open Worlds

Shivam Goel*
Tufts University
Medford, USA
Shivam.Goel@tufts.edu

Gyan Tatiya*
Tufts University
Medford, USA
Gyan.Tatiya@tufts.edu

Matthias Scheutz
Tufts University
Medford, USA
Matthias.Scheutz@tufts.edu

Jivko Sinapov
Tufts University
Medford, USA
Jivko.Sinapov@tufts.edu

ABSTRACT

As researchers are developing methods for detecting and accommodating novelties that will make AI agents more robust to unknown sudden changes in the "open worlds", there is an increasing need for benchmark environments that allow for the systematic evaluations of the proposed AI techniques. We present "NovelGridworlds", an *OpenAI Gym* environment framework for developing and evaluating AI agents that can detect and adapt to unknown sudden novelties in their environments. Based on a rich taxonomy of novelties illustrated in 4 different tasks with 12 novelties in each, we propose evaluation metrics for evaluating both planning systems and learning systems that can handle novelty and illustrate the metrics with results from simulations of both types of AI agents.

KEYWORDS

Learning and Adaptation, Novelty, Non-stationary MDP, Reinforcement Learning, Planning, Benchmarking Environments

1 INTRODUCTION

Artificial agents operating in real-world applications need the ability to be robust to unknown sudden changes in their operating environment. Yet, most research in AI has been based on "closed-world assumptions", i.e., that all task-relevant information, at least the involved types of objects and the types of attributes they have, is known ahead of task performance. In contrast, "open worlds" may contain novelties that are unknown during algorithm design and can, therefore, not be accounted for ahead of time. There are increasing efforts for developing AI agents to adapt to novelties in open worlds, using techniques such as reinforcement learning (RL) [7, 12], planning-based approaches [8, 9], and hybrid approaches [14, 16].

When developing algorithms for open worlds, researchers need to use domains to train and evaluate their agents, and the domains are likely tailored to the agent's algorithms and thus do not allow for a fair comparison across different approaches. What is needed is an evaluation environments capable of benchmarking different types of AI agents without introducing a bias towards one or another method. Moreover, evaluation environments with cross-platform support for different programming languages and support for various sub-domains of AI are not readily available. To address this problem, we present a "gym environment" called *NovelGridworlds*[1]

The rest of the paper is organized as follows. We start by providing background on the existing environments used for benchmarking such agents in Section 2.1, followed by providing basics of the planning agents and the reinforcement learning agents in Section 2.2. We then describe the base tasks in the NovelGridworlds domain in Section 3, followed by a detailed taxonomy of novelty with examples of the same implemented in NovelGridworlds. In Section 4, we present the agents used with our domain, followed by the proposed evaluation metrics, and the results in Section 4.3. We finish with a discussion of our accomplishments and possible future improvements in Section 5 and Section 6, respectively.

Our contributions are thus (1) a benchmarking environment for evaluating and prototyping multiple AI agents that can detect and handle novelties; (2) a rich taxonomy of novelties and illustration of 4 tasks with 12 novelties in each of the tasks; and (3) evaluation metrics to evaluate both planning agents as well as reinforcement learning agents with baseline results using both the agents on some tasks and novelties.

2 BACKGROUND

2.1 Related Work

In the recent past, there is a growing interest in the AI community to develop agents that can adapt to changes in open-world environments. Khetarpal et al. [7], presents a review of methods used to adapt to non-stationarity in environments using reinforcement learning methods. Muhammad et al. [9] propose a cognitive architecture using open world symbolic planning to solve novelties in an open-world environment. Jamshidi et al. [5] present an architecture that utilizes machine learning techniques to find Pareto-optimal configurations and limit the vast search space to such configurations to make planning feasible. Klenk et al. [8] propose HYDRA, which uses a domain-independent planner to play the game, and learns a model to update it when novelties get introduced in the game. Sarathy et al. [16] propose SPOTTER, which uses reinforcement learning techniques to find the operators to solve the tasks when symbolic planning fails to solve them. There has been an

*The first two authors contributed equally.

[1]The open-source NovelGridworlds repository is available at https://github.com/gtatiya/gym-novel-gridworlds.

enormous reliance on the choice of domains to experiment upon in all of these works. Hence, it becomes necessary to develop domains best suited for the problems at hand.

The development of environments and benchmarks has helped progress the research in various fields of AI [3, 11, 13, 21, 23]. Alex et al. [23] present GLUE, a benchmarking environment platform to bridge learning with natural language understanding. Jiang et al. [6] present WordCraft, an environment to develop commonsense agents. This environment also incorporates natural language corpora to improve common sense among agents. Nichol et al. [11], present a benchmarking environment to evaluate the effectiveness of transfer learning and few-shot learning techniques in the context of Reinforcement Learning. To bridge the gap for the development of algorithms with a combination of planning and learning, PDDLGym [21] offers an environment to combine PDDL (Planning Domain Definition Language) with OpenAI Gym [1] environments.

However, environments with cross-platform support for various programming languages and support for various sub-domains of AI are not that easy to find. Moreover, NovelGridworlds provide tools to inject several novelties in all the base tasks. To best of our knowledge there is no such environment that provides novelty injection tool.

2.2 Notations & Preliminaries

In sequential decision making problems, the agent has to perform decision making at each time step to reach a set of goal states g from a start state s_o. However, in solving the problems in open-worlds, there can be unexpected changes in the environments, which can deter the agent from reaching the goal state. As elucidated in [9], we borrow the definition of novelty as something that the agent has not experienced before, nor it can derive it using some known experience. We define these unexpected changes as novelties. Ideally, the agents should have the capabilities to still reach the goal state even if novelties arise in the environments. In this paper, we extend this definition by providing a taxonomy of the classes of novelties and present examples of each of them implemented in our NovelGridworlds domain.

2.2.1 Planning. To define a planning problem, we define $\mathcal{L}$, as a first-order language that contains atoms $p\,(t_1, \ldots, t_n)$ and their negations $\neg p\,(t_1, \ldots, t_n)$, where t_i can be either variables or constants. We define a planning domain in the language $\mathcal{L}$ as $\Sigma = (\mathcal{S}, \mathcal{A}, \tau)$, in which $\mathcal{S}$ represents the set of states, $\mathcal{A}$ represents the set of finite actions, and τ is the transition function between the states and actions [17]. A typical planning problem is defined as $\mathcal{P} = (\Sigma, S_0, G)$, in which S_0, is the set of starting states and G is the set of goal states. The agent begins in one of start states S_0, and finds a plan π to reach one of the goal state in G. Thus the plan $\pi = \left[a_1, a_2, \ldots a_{|\pi|}\right]$ is the solution to the planning problem $\mathcal{P}$ where $a_i \in \mathcal{A}$ is the action. Each action has a set of preconditions and effects. The set of preconditions needs to be true for the action to be executed successfully in the environment. Table 1 shows examples of how actions are defined in planning domain. Note that these actions are only for demonstration and can be defined using a different set of pre-conditions and effects.

Objects	physical: agent, tree-log, air world: loc01, loc02, loc03, ... direction: north, south, east, west
Predicates	at(physical, world) adjacent(world, world, direction) orientation(physical, direction)
Break	
Pre-Conditions	(and (at agent ?world01) (orientation agent ?direction01) (adjacent ?world01 ?world02 ?direction01) (at ?physical01 ?world02))
Effects	(and (not (at ?physical01 ?world02)) (increase (inventory ?physical01) 1))
Move-forward	
Pre-conditions	(and (at agent ?world01) (orientation agent ?direction01) (adjacent ?world01 ?world02 ?direction01) (at air ?world02))
Effects	(and (at agent ?world02) (not (at agent ?world01)))

Table 1: Examples of the *break* and *move-forward* action as represented in the planning domain.

2.2.2 Reinforcement learning. A RL problem is generally formalized using a Markov Decision Process (MDP) that is represented by a tuple $\langle S, A, \tau, R, \gamma \rangle$. At every time step t, the RL agent receives a state representation $S_t \in \mathcal{S}$ and explores an unknown environment by taking an action $A_t \in \mathcal{A}(s)$. A reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ is provided based on the action taken by the agent to reach the next state 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 [22].

3 ENVIRONMENT

NovelGridworlds is an OpenAI Gym environment which is implemented in python and currently has four different tasks. Each environment is a discrete grid world domain. The size of the default grid world is set to 10, which the user can vary. Each block contains an entity or object. The contents of the blocks can vary according to the tasks. However, in each environment, there are a few common block items. The entire arena is enclosed by a *wall*, which is unbreakable. Each environment has a block occupied by a *crafting-table*, a block occupied by the *agent*, and free space occupied by *air*. The agent can move in the blocks containing air. At each time step, the agent maintains an inventory of the collected items. A diagrammatic representation of the grid world is shown in figure 2. The diagrammatic representation is not specific to a particular task. It only shows the *agent*, *wall*, *crafting-table* and *air*. It also demonstrates how a novel entity appears in the environment. The agent is given a discrete set of actions, out of which 4 actions are common

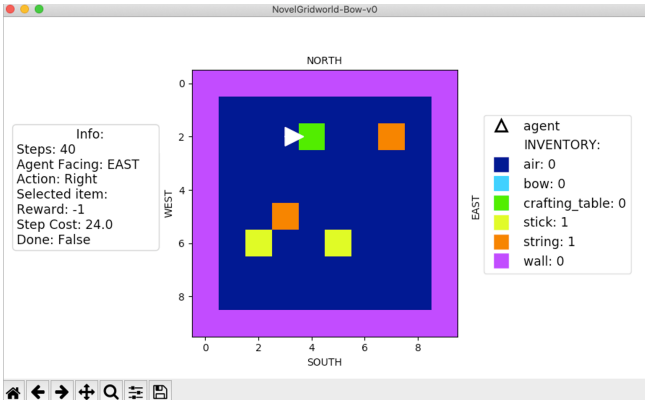
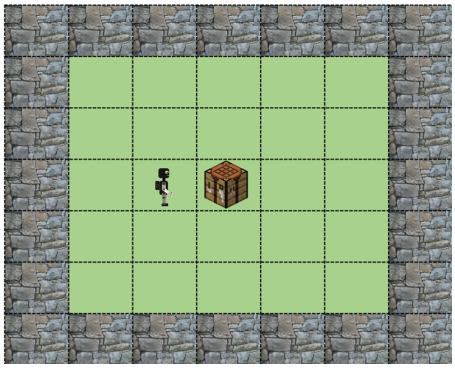
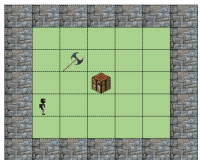


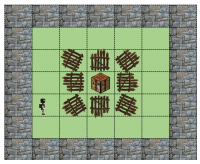
Figure 1: The figure shows a screenshot of one of the states of the NovelGridworld-Bow-v0 of size 10×10. The agent is represented as a triangle, and it is facing the crafting table (shown in green). Two of the three sticks (yellow) and strings (orange) are in gridworld and one of each is in the agent's inventory. The spaces in blue are the air blocks, and the agent can freely traverse in those spaces. The entire arena is surrounded by walls represented in purple. The image's left side shows the steps, agent's current direction, last action taken, reward, step cost, and selected item. On the right side of the image, the agent's inventory is displayed.



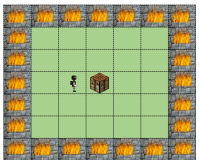
(a) Default (No novelty)



(b) Axe



(c) Fence



(d) Fire-wall

Figure 2: The Figures show a diagrammatic representation [2] of the NovelGridworlds domain and representation of some of the novelties. From subfigure (a), it can be seen that a wall surrounds the entire arena of the grid world, and the agent is facing a crafting-table. The green space represents the air, and the agent can move through that space to navigate inside the arena. In subfigure (b), a new object (axe) appears in the environment. In subfigure (c), a new object (fence) surrounds the crafting table to obstruct it from the agent. In subfigure (d), the walls are replaced by fire-walls.

to all the environments, namely *move-forward*, *turn-left*, *turn-right* and *break*. Some of the grid world blocks are occupied by breakable objects. The agent when one block next to these objects uses *break* action, the item falls into agent's inventory. For example, if the agent uses *break* action in front of the *tree-log*, the *tree-log* falls into the agent's inventory. The environment also returns an action cost for each action taken by the agent. This is designed by keeping in mind a planning agent to optimize the plan based on the action cost. In each task, the agent needs to craft objects using multiple recipes, which requires performing certain steps in some sequence. We describe each task in detail in the sub-sections below.

3.1 Task Descriptions

3.1.1 ***NovelGridworld-Bow-v0***. In this task, the agent has to achieve the goal of successfully crafting a *bow*. Apart from the *crafting-table*, *wall*, *air*, and *agent*, three blocks in the grid world are occupied by *string* and three blocks are occupied by *stick*. In addition to the default actions common to all the domains (described earlier), this task has an additional action called *craft-bow*. The *recipe* for creating a bow requires three *stick* and three *string*. After collecting all the required ingredients from the environment, the agent needs to navigate to the *crafting-table* to craft a *bow*. Upon successful execution, the *bow* appears in the agent's inventory, and the task is complete. A screenshot of this environment can be seen in figure 1.

3.1.2 ***NovelGridworld-Bow-v1***. As opposed to the previous task (3.1.1), in this task, the agent has to collect raw ingredients from the environment and craft the intermediaries to craft the *bow*. This task is an advanced version of the previous task. In addition to the default grid world items, the blocks in this grid world are occupied by *tree-log* and *wool*. The default action space is augmented by *extract-string*, *craft-plank*, *craft-stick* and *craft-bow*. The agent can use the action *craft-plank* to convert 1 *tree-log* into 4 *plank*, and use *craft-stick* to make 4 *stick* from 2 *planks*. The agent can use *extract-string* in front of *wool* to extract 4 *string* from the *wool*. The task's final goal is to craft a *bow*, and the agent needs to be in front of the *crafting-table* with 3 *stick* and 3 *string* to complete the task.

3.1.3 ***NovelGridworld-Pogostick-v0***. In this task, the agent has to craft a *pogostick*. The grid world in this task has *tree-log*, *tree-tap*, *stick*, *plank*, in addition to the default items. The default action space is augmented by *extract-rubber* and *craft-pogostick*. One of the *tree-log* has the *tree-tap* placed in front of it. The agent needs to extract *rubber* from that unique *tree-log*. The agent needs 4 *stick*, 2 *plank*, and 1 *rubber* to craft a *pogostick*. The agent needs to be facing the *crafting-table* to craft the *pogostick*. The task is complete when a *pogostick* gets added to the agent's inventory.

3.1.4 ***NovelGridworld-Pogostick-v1***. It is an advanced version of the task in 3.1.3. In addition to the default items, the grid world in this task has some blocks occupied with *tree-log*. The agent's default action space is expanded by *place-tree-tap*, *extract-rubber*,

[2] To reduce clutter, we only show the default items in the environment. Hence, the diagrammatic representation only shows the *agent*, *crafting-table*, *wall*, and *air*.

craft-plank, *craft-stick*, *craft-tree-tap*, and, *craft-pogostick*. In order to complete the task of successfully crafting the *pogostick*, the agent needs to craft intermediaries. The agent needs 1 *tree-log* to craft 4 *plank*, 2 *plank* to craft 4 *stick*, and 5 *plank* and 1 *stick* to craft a *tree-tap*. The *tree-tap* can be placed in any of the *tree-log* to extract *rubber*. The recipe to craft the *pogostick* requires 4 *stick*, 2 *plank*, and 1 *rubber*. To craft the *tree-tap* and *pogostick*, the agent needs to be facing the *crafting-table* with the necessary ingredients. Once the agent gets the *pogostick* in its inventory, the task is complete. Table 2 summarizes the actions and blocks in each task.

3.2 Novelty Taxonomy

As described in Section 2.2, a planning agent and a RL agent both have access to a finite set of states $\mathcal{S}$, set of finite actions $\mathcal{A}$, and only exclusive to a planning agent, access to the transition function τ. Any change in these sets would be considered novel to the AI agent. Therefore, to cover extensive novel scenarios, we classify the novelties by changing any or all of these sets. As described below, we divide the novelties into three broad classes.

3.2.1 Object Novelty. In this class of novelty, a novel object appears in the environment. This object is completely unknown to the agent's knowledge of the world. In other words, the observation space $x \in \mathcal{S}$ does not come from the distribution of the agent's current perception of the world, and hence in this class of novelty, state $\mathcal{S}$ changes. To illustrate an example, let us consider the *NovelGridworld-Pogostick-v1* task as described in Section 3.1.4. In the post-novelty scenario, a novel object called *axe* appears in the environment. The agent's current perception of the world has no notion of *axe* and hence it is considered as a novelty. The agent has to learn to interact with the *axe* to figure out its usage in goal completion.

3.2.2 Attribute Novelty. In this class of novelty, the attributes of the existing items in the environment changes. We define this class of novelty using the definition of transition function $\tau : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, as described in Section 2.2. In the post-novelty scenario, the environment changes such that $\tau' : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}'$ in which $\mathcal{S}'$ is the new state transition from the state action pair. Hence, similar state action pairs do not result in the same states as in the pre-novelty scenario. To illustrate this novelty, let us take an example of the *NovelGridworld-Pogostick-v1* task as described in Section 3.1.4. In the post-novelty scenario, if the *tree-log* provides 2 *tree-log* into the agent's inventory as opposed to 1, then we call it a attribute change of the *tree-log*. Therefore, the agent encounters a new state $\mathcal{S}'$ from known state $\mathcal{S}$ taking known action $\mathcal{A}$.

3.2.3 Action Novelty. In this class of novelty, the action space $\mathcal{A}$ of the environment changes. In other words, the behavior of action as known to the agent is not the same in the post-novelty scenario. To illustrate this class of novelty let us consider the agent's action space to be a set of three actions, namely, *move-forward* which takes the agent one step forward in the direction the agent is facing; *turn-left*, which turns the agent 90° counter-clockwise; and *turn-right*, which turns the agent 90° clockwise. In the post-novelty scenario, if the agent moves one block backward on using the action *move-forward* or if the agent turns 90° clockwise on using the action *turn-left*, then the agent's perception of the actions will change and hence, we call this as action novelty. We provide specific cases of each class of novelty later in the Section 3.3

We further divide each class of novelty into three sub types, *Beneficial*, *Detrimental*, and *Irrelevant*.

3.2.4 Beneficial. This type of novelty is helpful for the agent to complete the task at hand. Hence, the agent needs to learn to utilize this novelty in task completion.

3.2.5 Detrimental. This type of novelty is further catastrophic in task completion. In other words, the agent needs to learn to mitigate this novelty to complete the task efficiently.

3.2.6 Irrelevant. This type of novelty is irrelevant to the task. In other words, the novelty can be completely ignored by the agent.

Actions and Blocks in Tasks		
Environment	**Actions**	**Blocks**
Default	Move-forward, turn-left, turn-right, break, select-<block>.	air, wall, crafting-table
NovelGridworld-Bow-v0	craft-bow.	string, stick
NovelGridworld-Bow-v1	craft-bow, extract-string, craft-plank, craft-stick.	tree-log, wool, string, stick
NovelGridworld-Pogostick-v0	extract-rubber, craft-pogostick.	tree-log, tree-tap, stick, plank
NovelGridworld-Pogostick-v1	extract-rubber, craft-plank, craft-stick, craft-tree-tap, craft-pogostick.	tree-log, tree-tap, stick, plank

Table 2: Table shows the actions and blocks specific to each of the implemented environments. Note that "Default" denotes the actions and blocks common to all the environments.

Novelty Type			
Novelty Class	**Beneficial**	**Detrimental**	**Irrelevant**
Object	axe	fence	spring, arrow
Attribute	break increase, extract-increase	extract-decrease, fire-wall	-
Action	chop, jump	action remap	-

Table 3: Classification of all novelties based on the class and type.

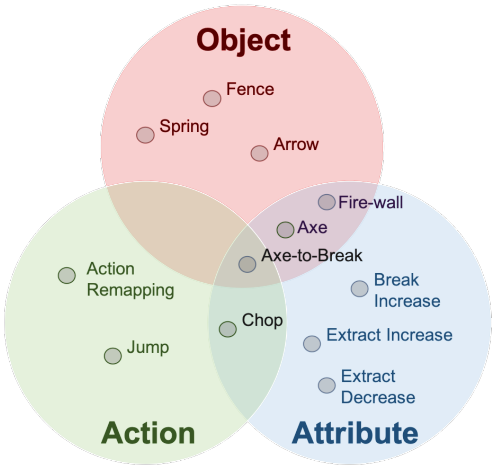


Figure 3: Classification of the novelties implemented in the NovelGridworlds. The novelties in this diagram are implemented across all the domains presented in the paper.

3.3 Novel Domains

In the present version of NovelGridworlds, we implement 12 [3] exclusive novel scenarios; a detailed list is shown in Table 3. Figure 3 shows the classification of each novelty based on the classes they fall under.

3.3.1 Fence. This novelty falls under the category of *Object* novelty. A new object called *fence*, appears in the environment. The *fence* encloses some or all of the existing items in the environment to obstruct the agent's access to those items. This novelty, as shown in Table 3, is classified as a detrimental novelty. In the ideal scenario, the agent should learn to *break* the fence to access the items in the environment.

3.3.2 Arrow & Spring. This novelty falls under the category of *Object* novelty. A new item called *arrow* or *spring* appears on some grid spaces in the environment. An *arrow* appears in *Bow-v0*, and *Bow-v1*, and *spring* appears in *Pogostick-v0* and *Pogostick-v1*. Both these novelties are classified as *irrelevant* novelties. Hence, they have nothing to do with the task and the agent should learn to ignore them.

3.3.3 Break increase. This is an *Attribute* novelty, in which the attribute of the existing item in the environment changes. In the pre-novelty scenario, if the agent breaks an object, the number of quantity of that object in the agent's inventory would be incremented by 1. However, in *break increase* novelty, after breaking an object, the agent would receive 2 quantities of that object in its inventory instead of 1. This is a *beneficial* novelty as the agent can collect more objects with lesser number of actions and finish the task sooner as compared to pre-novelty scenario.

3.3.4 Extract increase/decrease. This is an *Attribute* novelty, in which the attribute of the existing item in the environment changes. In this novelty, there is a change in the number of objects the agent gets when it executes *extract* action. This novelty can be beneficial as well as detrimental. If there is an increase in the number of objects the agent gets, it would be a beneficial novelty, whereas in case of decrease in the number of objects, it would be a *detrimental* novelty.

3.3.5 Action remapping. This novelty falls under the class of *Action* novelty, and is further classified as *detrimental*. In this novelty, the effects of the actions as known to the agent are not same anymore. For example, *turn-left* does not turn left anymore but make the agent move forward. We randomly shuffle the action effects with each other to remap actions among the set of available action in each task. This novelty is classified as a detrimental novelty, and the agent should learn to handle such a scenario. In a real-world scenario, this can be analogous to a system getting hacked, and the autonomous agent needs to figure out a way to get out of such a situation.

3.3.6 Jump. This novelty falls under the class of *Action* novelty, and is further classified as *beneficial*. In this novelty, a new action called *jump* is added to the action space of the agent. This action makes the agent jump two blocks forward in the direction it is facing. This novelty is a beneficial novelty because it helps the agent to move faster in the environment.

3.3.7 Axe. This novelty falls under two classes, *Object*, and *Attribute*. In this novel scenario, a new object called *axe* appears in the environment, and if the agent uses it, the number of quantities received from breaking something in the environment doubles. Hence, this novelty introduces a new object and also changes the attributes of the environment. This novelty is further classified as beneficial, and ideally, the agent should learn to exploit this novelty to its advantage.

3.3.8 Fire wall. This novelty falls under two classes, *Object*, and *Attribute*. In this novel scenario, the material of the wall enclosing the arena changes to *fire-wall*, and if the agent goes near it, it dies, and the episode terminates. This novel scenario is further classified as a detrimental novelty since the agent needs to be extra careful in going near the *fire-wall*.

3.3.9 Chop. This novelty falls under two classes, *Action* and *Attribute*. In this novelty, a new action called *chop* is added to the agent's action space. Upon using this action in front of the breakable objects in the environment, the agent gets double the number of items previously received. This novelty falls under the classification of beneficial novelty. Ideally, the agent should discover this novel action and learn to double the number of items.

3.3.10 Axe to break. This novelty falls under all three categories, *Object*, *Action*, and *Attribute*. In this novelty, a new object call *axe* appears in the environment, and the break action only works if the agent is holding an *axe*. This novelty is detrimental to the agent's perspective since it cannot complete the task unless it discovers this novelty. This novelty is not included in the table 3, because it is falls under two categories of detrimental and beneficial and also under three classes.

[3] To prevent redundancy, we have clubbed some novelties (extract increase, decrease and arrow, spring) into one description.

4 EXPERIMENTAL RESULTS

4.1 Agent Descriptions

To demonstrate our environments, we ran experiments using two different types of agents on some of the base tasks and novel tasks.

4.1.1 Planning . The planning agent was implemented using an extension of cognitive Distributed Integrated Affect Re-flection Cognition (DIARC) architecture [18, 19]. The planning agent parses the JSON received from the environment and generates a problem and a domain PDDL file. MetricFF [4] planner is used to find a plan to solve the task. The plan is executed using an action executor implemented in the architecture. At each time step the agent updates its memory based on the information it receives from the environment. We used the same architecture as described in [9]. The system maintains a living, episodic and a universal memory to infer novelties in the environment. The architecture uses heuristic algorithms as proposed in [17] to adapt to the novel situations.

4.1.2 Reinforcement Learning (RL) Agent. We created a reinforcement learning agent [4] with LIDAR sensors. The sensors send 8 beams at 45 degree increments to each reachable object in the environment. Each beam returns a euclidean distance from the agent's current location to the object. The agent also has a vector representing the current inventory. For the sake of simplicity, the *crafting-table* was made an unbreakable object. The reward function was also shaped to make learning easier. A reward of +10 was given if the agent collected one of the ingredients for the task completion. A reward of +50 was given for achieving the goal, and −1 for each time step otherwise. For training the RL agent, we used the Proximal Policy Optimization (PPO2) [20] implementation in the open-source Stable Baselines package [15], with default hyperparameters. We chose an off the shelf implemented RL algorithm to demonstrate how users can easily plug in existing algorithms with our environment.

4.2 Evaluation Metrics

In this section, we describe our proposed evaluation metrics for measuring performance of novelty-centric agents in both planning systems and learning systems.

4.2.1 Base task performance metrics. These metrics evaluate the performance of the agents on the base task (pre-novelty scenario). In both the planning and reinforcement learning agents, we measure the number of time steps the agent takes to complete the base task. Specifically for the RL agent, we also measure the percentage of times the agent achieves the final goal. In the case of the planning agent, the agent always succeeds in reaching the goal in the pre-novelty environment.

4.2.2 Novelty metrics. These metrics evaluate the performance of the agent after the novelty is introduced. The metrics are reliant on two criterion, namely detection and adaption. Hence we define three metrics to evaluate the performance of the agent in the novelty scenario.

[4]The code that trains the environment using the RL agent using stable-baselines is available at https://github.com/goelshivam1210/adaptive_agent. It also has the necessary PDDL files and other data to reproduce results presented in the paper.

- Metric M1 (Reaction Performance): This metric is defined as the ratio of the cumulative action cost that the agent accumulates in the post-novelty scenario to the action cost accumulated in the pre-novelty scenario. In other words, if the ratio is less than 1, the performance is improved; if the ratio is greater than 1, the performance is reduced and if the ratio is 1, the performance is same.
- Metric M2 (Steps): This metric records the number of steps taken by the agent to solve the task in the post-novelty scenario.
- Metric M3 (Detection Rate): This metric records the binary value of whether the agent reports the novelty or not. This metric is only specific to the planning architecture.

To demonstrate the RL agent's performance, we plot the learning curves. We divide the learning curve into two sections; section 1 presents the agent's learning performance in the pre-novelty scenario and section 2 represents the agent's performance in the post-novelty scenario. We present the test results of the agent over a course of 10 trials. In each trial, we save 20 models uniformly distributed across the training regime. We perform a test on each model, by running it on the corresponding environment for 25 episodes, each with a maximum of 200 steps. We compute the cumulative reward in every episode and report the cumulative mean reward of all the 25 episodes. This score averaged across 10 trials is plotted against the number of time steps the agent is trained. To do fair comparison to actually know whether the RL agent learned to solve the task, we also report the success rate of achieving the goal task. To compute the success rate, we run 25 tests on the final models of each training regime and report the percentage of times the agent successfully reaches the goal state.

4.3 Results

4.3.1 Planning Agent. We evaluated the planning agent on all the 4 tasks, and 3 novelty scenarios in each task.

(1) Fence
(2) Arrow/Spring
(3) Axe

Table 4 shows the performance of the planning agent. We ran the agent for 10 episodes. In each episode the agent was given the environment dynamics, and a goal to solve. Keeping the goal same across the episodes, we introduce the desired novelty in the 6^{th} episode. After the 6^{th} episode, all the subsequent episodes were loaded with the same novel environment. At each time step in an episode, the agent gets a step cost for the action taken from the environment. The cumulative step cost is used to compute the metric M1. The agent was able to achieve the goal in all the episodes. The table shows the mean and standard deviation of five non-novel episodes and five novel episodes. The planning agent we evaluate achieved a 100% detection rate in all the novelty scenarios. Hence, the metric M3 for the planning agent was a perfect score.

It is important to note that the metric M1 for fence novelty in all the environments is greater than 1, which clearly shows that the agent needs to spend a higher cost to break the fences to achieve the goal. Since the fence is a detrimental novelty, the agent accumulates more cost while handling the novelty. The metric M2, in this case

NovelGridworld-Bow-v0				
Metrics	**Default**	**Fence**	**Arrow/Spring**	**Axe**
M1	1.00 ± 0.00	1.80 ± 0.15	1.11 ± 0.135	1.02 ± 0.04
M2	87.6 ± 16.8	123.0 ± 18.0	90.6 ± 15.1	98.0 ± 15.9
NovelGridworld-Bow-v1				
M1	1.00 ± 0.00	1.82 ± 0.24	1.00 ± 0.02	0.99 ± 0.02
M2	36.8 ± 6.5	54.2 ± 4.8	37 ± 5.4	35.4 ± 4.0
NovelGridworld-Pogostick-v0				
M1	1.00 ± 0.00	1.16 ± 0.08	0.97 ± 0.03	0.99 ± 0.04
M2	127.8 ± 9.0	122.0 ± 19.8	105.6 ± 15.7	114.8 ± 4.0
NovelGridworld-Pogostick-v1				
M1	1.00 ± 0.00	1.34 ± 0.06	1.07 ± 0.05	0.99 ± 0.02
M2	94.2 ± 12.9	105.2 ± 14.5	100.8 ± 22.58	85.0 ± 15.4

Table 4: Table shows the results of the performance of the planning agent on all the four tasks. In each task, it shows pre-novelty (Default) scenario and 3 post-novelty (Fence, Arrow/Spring, Axe) scenarios.

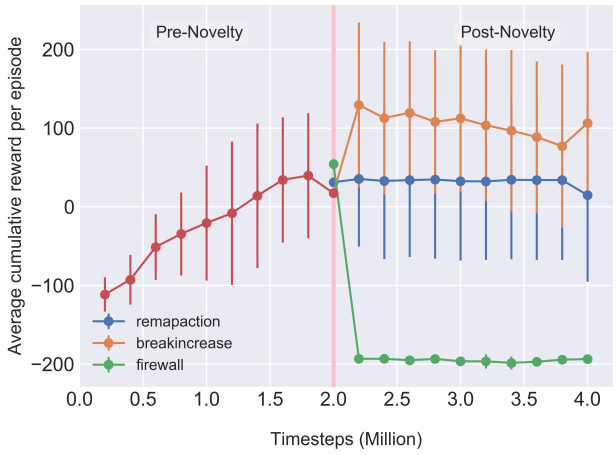


Figure 4: Figure shows the learning performance of the RL agent on the task of *NovelGridworld-Bow-v0*. The pink line in the center (2 Million) separates the performance of the agent in the pre-novelty scenario to the agent's performance in cases of novelty.

also shows that the agent needs more steps than the default case to finish the task.

It can be seen from Table 4, that the agent seeks to improve its performance in the *axe* novelty. However, the improvement is negligible. A possible reason for negligible improvement can be that the agent takes more steps to complete the tasks in novel cases (since it needs to navigate to the axe and hold it). Hence, it incurs more step cost than it improves by using the axe. Moreover, we noticed that in some of the episodes the agent did not even use the axe to benefit from using it. In the case of the irrelevant novelty (arrow/spring), the agent's performance is nearly the same as the default case, as expected.

4.3.2 Reinforcement learning agent. The RL agent was evaluated on the *NovelGridworld-Bow-v0* task. We evaluate the performance on 3 novelty scenarios in this task. The agent was evaluated on these novelties for demonstration purposes and users are encouraged to develop methods to handle more complex novelties as implemented in the environment.

(1) Action remapping
(2) Fire wall
(3) Break increase

Figure 4 shows the performance of the RL agent on the base task (shown in red) and in the three novelty scenarios, namely *break-increase* (shown in orange), *fire-wall* (shown in green), and *action remapping* (shown in blue). The figure was plotted as described earlier in Section 4.2 . It can be seen from Figure 4, that the RL agent performs well in solving the task after being trained for around 1.8 million time steps. Row 1 of Table 5, shows the success rate (percentage of times the agent reaches the goal state) of the default task after 1.8 million time steps of training. Its encouraging to see that the agent improves its performance in the post novelty scenario in the case of *breakincrease* novelty (orange line in Figure 4). This was expected since it is a beneficial novelty. However, if we notice in Table 5, the success rate of the agent drops considerably. One of the possible reasons for increase in the total rewards but decrease in the performance can be that the agent tries to collect resources and hoard reward without completing the final task. We also notice a sharp decline in agent's performance when the *fire-wall* novelty is introduced. *Fire-wall* novelty is a detrimental novelty and the task becomes quite hard to solve, hence we see that the agent never recovers in this novel scenario. Table 5, also reflects the poor performance of the agent in the *fire-wall* novelty.

Action remapping novelty however, showed almost similar performance when compared to the pre-novelty performance. Provided that its a *detrimental* novelty, the behavior is surprising. A possible reason for this behavior can be that the action space is too small (5 discrete actions), and the actions are randomly shuffled. Therefore, in some trials the agent might have got lucky and only 2 actions might have inter-changed. We discovered this, when we inspected the agent's performance on each trial, and found out that it dipped considerably (to almost -150 in the reward curve), when all the actions were remapped. However, it did bounce back to original performances in around 200, 000 environment interactions. Table 5, however, shows the decline of 12% in the performance of the agent.

The RL agent was not tailored explicitly to handle novelties. Hence, we could not evaluate it on all novelties. However, RL agents do not have access to the transition dynamics and learn from interactions, hence some level of non-staionarity such as, action remapping and break increase was handled by the RL agent. The addition of a new item in the agent's state space is particularly hard, especially when the state space representation of the RL agent is symbolic. Hence, we did not evaluate the agent on *object* novelties, and leave it as an open challenge for the RL community

5 DISCUSSION

The architecture of NovelGridworlds is designed for cross-platform applicability and cross-agent compatibility. As shown in Section 4,

NovelGridworld-Bow-v0	
Novelty	**Success (%)**
Default	92.0 ± 20.8
Action remapping	80.00 ± 40.00
Fire wall	0.00 ± 0.00
Break increase	29.2 ± 19.6

Table 5: Table shows the results of the performance of the reinforcement learning agent on the base task of *NovelGridworld-Bow-v0* and three novelty scenarios. The values denote the mean and standard deviations of the success rate of solving the final task. The values were calculated at 1.8 M timestep for the default case and 4 M time step for all the novelty scenarios.

NovelGridworlds is capable of prototyping algorithms both in the planning domains and machine learning domains. It is a fast and lightweight simulation with a socket connection [5] for communication between the agent and the environment. Support for socket connection provides an efficient way to connect the environment with various agents, implemented using different programming languages. We demonstrate this ability by evaluating the environment using agents developed in Java as well as Python.

Another striking feature of the NovelGridworlds is the verbose information about the state of the world. This information can be exploited by Natural Language Processing (NLP) algorithms to perform novelty reasoning. Hence, combining NLP algorithms with the RL and Planning frameworks could be an important research direction that can be pursued using our environment. Our framework is also flexible in adding multiple novelties to the same task, thereby making the task harder and generating a curriculum. This functionality provides the capability of training a curriculum for reinforcement learning agents [10].

Reinforcement learning algorithms deal with sample inefficiency, and many researchers work in the direction of pre-training using human demonstrations [2]. To that extent, we provide keyboard interface support in our environment and provide support for recording trajectories, which can be further used to perform learning from demonstrations. We also provide code examples of wrappers for limiting action space, implementing various observation spaces essential in RL algorithms training.

6 CONCLUSION & FUTURE WORK

We presented *NovelGridworlds*, a robust open-source benchmarking environment for developing and evaluating agents capable of handling novelties in open worlds, together with a rich taxonomy of how novelties in this domain are characterized. Results from running and evaluating both planning and learning agents in different tasks show the utility of *NovelGridworlds* for comparing and quantifying different approaches to open-world AI.

Moving forward, we plan to include multi-agent support that will allow for training and benchmarking multi-agent RL and planning algorithms. And while the current version only allows for static novelties (i.e., novelties that remain the same throughout a task run), future versions of the system will also include dynamic novelties that can change at any time during task performance. This will enable even more challenging evaluation settings that can test an agent's ability to quickly react to novelties.

[5]We have detailed out all the special features and how to use them in the readme of our environment which can be found at this link: https://github.com/gtatiya/gym-novel-gridworlds/blob/master/README.md

ACKNOWLEDGMENTS

This work was funded in part by DARPA grant W911NF-20-2-0006.

REFERENCES

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. arXiv:arXiv:1606.01540

[2] Gabriel V. de la Cruz, Yunshu Du, and Matthew E. Taylor. 2019. Pre-training with Non-expert Human Demonstration for Deep Reinforcement Learning. *The Knowledge Engineering Review* 34 (2019), e10. https://doi.org/10.1017/S0269888919000055

[3] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. 2020. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219* (2020).

[4] Jörg Hoffmann and Bernhard Nebel. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. 14 (2001), 253–302.

[5] Pooyan Jamshidi, Javier Cámara, Bradley Schmerl, Christian Käestner, and David Garlan. 2019. Machine learning meets quantitative planning: Enabling self-adaptation in autonomous robots. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 39–50.

[6] Minqi Jiang, Jelena Luketina, Nantas Nardelli, Pasquale Minervini, Philip HS Torr, Shimon Whiteson, and Tim Rocktäschel. 2020. WordCraft: An Environment for Benchmarking Commonsense Agents. *arXiv preprint arXiv:2007.09185* (2020).

[7] Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. 2020. Towards Continual Reinforcement Learning: A Review and Perspectives. *arXiv preprint arXiv:2012.13490* (2020).

[8] Matthew Klenk, Wiktor Piotrowski, Roni Stern, Shiwali Mohan, and Johan de Kleer. [n.d.]. Model-Based Novelty Adaptation for Open-World AI. ([n. d.]).

[9] Faizan Muhammad, Vasanth Sarathy, Gyan Tatiya, Shivam Goel, Saurav Gyavali, Mateo Guaman, Jivko Sinapov, and Matthias Scheutz. 2021. A Novelty-Centric Agent Architecture for Changing Worlds. In *Proceedings of the 2021 International Conference on Autonomous Agents & Multiagent Systems*.

[10] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. 2020. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research* 21, 181 (2020), 1–50.

[11] Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. 2018. Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720* (2018).

[12] Sindhu Padakandla, KJ Prabuchandran, and Shalabh Bhatnagar. 2020. Reinforcement learning algorithm for non-stationary environments. *Applied Intelligence* (2020), 1–17.

[13] Fabio Pardo. 2020. Tonic: A Deep Reinforcement Learning Library for Fast Prototyping and Benchmarking. *arXiv preprint arXiv:2011.07537* (2020).

[14] Xiangyu Peng, Jonathan C Balloch, and Mark O Riedl. 2021. Detecting and Adapting to Novelty in Games. (2021).

[15] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. 2019. Stable Baselines3. https://github.com/DLR-RM/stable-baselines3.

[16] Vasanth Sarathy, Daniel Kasenberg, Shivam Goel, Jivko Sinapov, and Matthias Scheutz. 2020. SPOTTER: Extending Symbolic Planning Operators through Targeted Reinforcement Learning. *arXiv preprint arXiv:2012.13037* (2020).

[17] Vasanth Sarathy and Matthias Scheutz. 2018. MacGyver problems: Ai challenges for testing resourcefulness and creativity. *Advances in Cognitive Systems* 6 (2018), 31–44.

[18] Paul W Schermerhorn, James F Kramer, Christopher Middendorff, and Matthias Scheutz. 2006. DIARC: A Testbed for Natural Human-Robot Interaction.. In *AAAI*, Vol. 6. 1972–1973.

[19] Matthias Scheutz, Thomas Williams, Evan Krause, Bradley Oosterveld, Vasanth Sarathy, and Tyler Frasca. 2019. An overview of the distributed integrated cognition affect and reflection DIARC architecture. *Cognitive architectures* (2019), 165–193.

[20] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[21] Tom Silver and Rohan Chitnis. 2020. PDDLGym: Gym Environments from PDDL Problems. In *International Conference on Automated Planning and Scheduling (ICAPS) PRL Workshop*. https://github.com/tomsilver/pddlgym

[22] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[23] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461* (2018).