Proceedings of the Ninth Annual Conference on Advances in Cognitive Systems Submitted 9/2021; published 11/2021

OpenMIND: Planning and Adapting in Domains with Novelty

David J. Musliner MUSLINER@SIFT.NET
Michael J. S. Pelican MPELICAN@SIFT.NET
Matthew McLure MMCLURE@SIFT.NET
Steven Johnston SJOHNSTON@SIFT.NET
Richard G. Freedman RFREEDMAN@SIFT.NET
Corey Knutson CKNUTSON@SIFT.NET
Smart Information Flow Technologies (SIFT), Minneapolis, MN 55401 USA

ABSTRACT

We describe OpenMIND, a goal-oriented agent that plans to achieve its goals, executes its plans, and revises its goals and planning models on the fly when novel, unexpected situations arise. This general idea is not new; our contributions lie in the development of several domain-independent model-modification and goal-reasoning heuristics that have proven effective in handling novelty, based on blind evaluations with quantitative metrics. The metric results show that OpenMIND is reliably able to detect novelties and then take advantage of novel opportunities and avoid or work around novel hindrances using a small collection of domain-independent techniques. Overall, for evaluation trials that presented novel challenges and opportunities across two different domains, OpenMIND adapted to novelties to achieve 80% or more of the performance achieved by a baseline agent on pre-novelty tasks in that domain.

1. Introduction

Our objective is to develop a discrete-action task-oriented agent that plans to achieve its goals, executes its plans in open worlds and monitors their results, and effectively revises its goals and planning models on the fly when novel, unexpected situations arise. This is an ambition of wide interest to those building physical and virtual cognitive systems that can survive and succeed in open worlds. This general idea is not new (*e.g.*, Klenk et al. (2013); Cox (2016); Cox & Dannnenhauer (2017)). Our contributions lie in the development of several domain-independent model-modification and goal-reasoning heuristics that have proven effective in handling novelty, based on **blind evaluations with quantitative metrics**. We have implemented these heuristics and adaptation techniques in the context of our OpenMIND autonomy architecture and they have been rigorously evaluated in two independently-developed domains (with a third domain upcoming).

The particular focus of our work is how cognitive systems can be designed and built to detect, overcome, and even exploit, novelties encountered in open world domains. We define "novelty" as significant changes to the agent's world that are not represented in the agent's initial model of how the world starts and changes over time, whether that initial model has been learned from previous

experience or engineered from specifications or examples. Although this definition encompasses many types of novelty at different levels of domain models (*e.g.*, object features, action transitions Boult et al. (2021)), we limit ourselves to detectable, sudden, persistent changes to the agent's environment (Langley (2020)). These novel changes persist, so the agent may be able to improve its performance if it adapts its behavior after the novelty occurs.

Although several systems have been developed to address the challenges of novelty, we describe a unique combination of heuristics organized by a hypothesis-testing framework that has been evaluated at a new level of rigor through cross-domain, single-blind evaluation. Funded by the DARPA SAIL-ON program, this work is part of a multi-team effort to explore ways that autonomous agents can be robust to a wide variety of types of novelty, in a scientifically sound research approach based on uniform metrics and single-blind evaluations (Langley (2020)). The research groups developing domain simulations that include novelty are separate from the research groups developing agents, and only very limited exchange of novelty examples is allowed. Agents must detect and respond to novel aspects that arise during their interactions with the domain simulations, and their performance on both detection and response is measured over many different interactions and different novelties. Performance evaluations are conducted every six months by the domain developers, using a variety of "unrevealed" novelty examples that have not been shown to the agent developers. After each evaluation, one or more of the novelty examples has sometimes been revealed to help agent developers make progress; subsequent evaluations include new novelties of related and unrelated types. Now past its 18th month, the SAIL-ON program has spurred significant advances in domain-independent methods for robustness in the face of truly novel domain elements.

This research is not concerned with **entirely** novel domains where the agent must learn how to perform a task from scratch with no prior knowledge (*i.e.*, this is not a *tabula rasa* problem). Here we are concerned with an agent that is already skilled in its intended domain, through training or hand-crafting, but faces some novelties that arise on the fly and either interfere with its normal operations or present opportunities for improvement. This scenario matches much more closely with real-world situations where a well-developed, highly trained/tuned autonomous system might be deployed, yet still encounter novel situations.

To set the stage for our focus on domain-independent novelty handling, we begin by briefly describing three domains from the SAIL-ON program, each presenting different challenges to OpenMIND. We then describe in detail how OpenMIND uses domain-independent hypothesis reasoning to manipulate its goals and planning models, yielding successful adaptations to novelties previously unknown to both the agent and its creators. In the Experiments and Results section, we present quantitative evaluations based on results that the simulator developers obtained when running our OpenMIND agent in their environment. The metric results show that OpenMIND is reliably able to detect novelties and then take advantage of novel opportunities and avoid or work around novel hindrances. We then briefly review related work, acknowledging that OpenMIND's architecture is not unique, but builds on a long history of research on agents, learning, and heuristics for domain-independent problem solving. We conclude by reiterating that our contributions are in domain-independent heuristics for handling novelty and a scientifically-sound evaluation, and note an interesting problem with this type of blind evaluation: the bug-or-novelty conundrum.

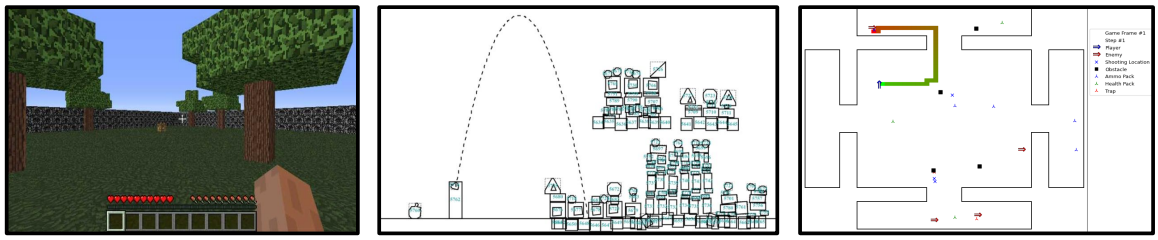

Figure 1: Views of the Polycraft, Sciencebirds, and Vizdoom domains.

2. Domains

To demonstrate how OpenMIND's novelty-handling reasoning is domain-independent, we are applying the system to three different domains during the SAIL-ON program. Illustrated in Figure 1, these domains pose varying challenges and have widely diverse forms of novelty, and we have enhanced OpenMIND with both domain-dependent and independent methods for each. During the third-party evaluations, the basic unit of evaluation is a "trial" of a hundred or more "games" to be played by OpenMIND. In each trial, the agent is faced with repeated modest variations of a challenge (game) it must solve in a limited time or number of moves. The initial configuration of each game is drawn from one of two distributions of variations. In the first distribution, "pre-novelty" games conform with the expectations of the agent. In the second distribution, "novel" games contain some significant variation from the world for which the agent was designed or trained. The point at which the trial shifts from the pre-novelty distribution to the novel distribution is randomly determined. The agent receives a score at the end of each game and may receive other intermediate score information, depending on the domain. The agent can learn from game to game, but it begins each trial in its original "pre-novelty" form (in our case, with the hand-crafted knowledge we built into OpenMIND, but no knowledge learned from playing games in novelty-bearing trials). More details on the evaluation process are in Section 4.

The following sections give brief descriptions of the domains and some of the (revealed) novelties involved. We develop and test OpenMIND using the games provided by the domain developers, and, when possible, we also invent our own novelties to test OpenMIND features. The domain developers evaluate our agent every six months using novelties that have not been previously revealed to us.

2.1 Polycraft

Developed by a team at University of Texas, Dallas (UTD (2020)), Polycraft is a Minecraft mod that can itself simulate many open-world domains. This domain, the one we started with, is perhaps best suited to domain-independent symbolic reasoning about novelty, since the underlying simulation is so broadly flexible and the action space is highly discretized and implicitly parameterized (roughly corresponding to a limited set of mouse clicks and keystrokes whose effects vary depending on the player's environment). For the SAIL-ON Phase 1 "POGO" challenge, Polycraft simulates a world in which an agent must chop down trees to obtain wood logs and thence wooden planks, build and

use a tree-tap to obtain rubber, and assemble those resources into a pogo stick using the Minecraft "crafting table." In addition to "non-novelties" such as different locations for objects such as trees and the crafting table, various revealed novelties have included "fence blocks" that obstruct access to trees (and must be removed), "axes" that can chop down trees more efficiently but are expensive to build from scratch, and "jungle" trees that make rubber more efficiently than normal trees. Actions in Polycraft have individual costs, and also can report failure if they are invoked incorrectly (*e.g.*, if an agent tries to move to an inaccessible location).

2.2 Sciencebirds

Building on the popular Angry Birds game, Australian National University (ANU) developed Sciencebirds to simulate a world where the agent must use a slingshot to shoot birds at pigs resting in an environment full of other objects such as wood blocks, ice blocks, etc. Renz et al. (2015) organize an annual AI challenge to develop an agent that can play Sciencebirds. Score points are accumulated both for killing pigs and breaking some of the other objects. Beyond the "non-novel" different numbers and types of birds, pigs, and other objects in different configurations, revealed novelties have included objects with different appearance that are nevertheless to be treated as pigs or birds, rotations of the screen coordinates, changes in object value, "health," speed, etc.

2.3 Vizdoom

Our most recent domain is Vizdoom, modified from an original vision-oriented first-person shooter game Wydmuch et al. (2018), and provided in symbolic-sensing form to the SAIL-ON program by Washington State University (WSU). Vodnala (2019) explains that Vizdoom will be part of an IQ test for AI. In this domain, OpenMIND controls an agent that moves about the various rooms, as shown in Figure 1, shooting enemies repeatedly and navigating to scattered health and ammunition packs. The symbolic version from WSU provides the wall coordinates, player and enemy coordinates/orientation, and locations of objects, so no vision processing is needed. While Polycraft includes teleport movement commands that allowed us to avoid route planning, Vizdoom requires 2D route planning and determining suitable locations to shoot from (given limited orientation angles and obstacles that block shots). Accordingly, we have added a fairly simple set of algorithms for those functions. Novelties in Vizdoom include things like enemies moving faster than normal, more health packs in the environment than normal, etc. OpenMIND has not yet been independently evaluated in the Vizdoom domain.

3. Architecture

As illustrated in Figure 2, the OpenMIND architecture has much in common with other planning and execution agents situated in dynamic worlds (*e.g.*, Kuipers et al. (2017); Musliner et al. (1993); Bonasso et al. (1996); Albus et al. (2002)). However, it faces the uncommon challenge of encountering novel obstacle and opportunities in three virtual environments created independently by three different teams and having its performance objectively evaluated in all three.

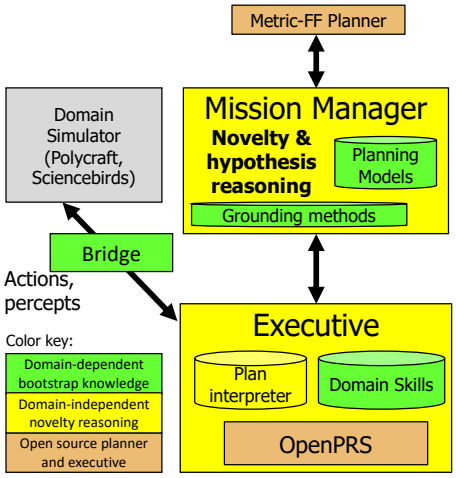

Figure 2: OpenMIND's Mission Manager reasons about novelties to exploit or overcome them. The other elements of the OpenMIND architecture closely resemble their counterparts in other situated agent architectures.

The two major components of the OpenMIND architecture are the OpenMIND Executive and the OpenMIND Mission Manager (OMMM)[1]. The Executive manages the reliable execution of a plan sequence by checking preconditions, evaluating success/failure conditions, and, in a few cases, combining low-level simulator interactions into a single plan action. OMMM uses symbolic models of actions, transitions, and world states to build plans to achieve specific goals. OMMM has the additional, more challenging task of detecting evidence of novelty in its interactions with the domain, proposing hypotheses that explain the novel observations, adapting its planning models to compensate, and evaluating the success of its efforts.

For example, when operating in the Polycraft POGO domain, OpenMIND repeatedly achieves a single main task (creating a pogo stick) while learning to compensate for novelties in the domain, including novel perceptions, novel items in the world, novel actions it can take, *etc*. Some novelties can impede the agent's original plans for building a pogo stick, while other novelties offer opportunities for improvement. For example, in one novel situation the agent awakens in the domain to find there is an unknown object, an "axe," lying on the ground. Having never seen an axe before, and with no semantic description of what an axe is good for (*i.e.*, it could be a "foobar"), the challenge is for the agent to discover that an axe makes breaking trees into logs much easier, and thus improves its score on the task of making a pogo stick.

OMMM solves this learning challenge by forming explicit *testable hypotheses* about what might be different in the world, and how it might correspondingly change its action models to compensate (see Figure 3). In the axe example, OMMM forms several hypotheses that suggest that using the axe in new ways in its various actions, including breaking trees, is a good idea. These correspond to

1. A domain-dependent Bridge component supports the Executive and OMMM by translating the native representations of the different simulators into a standard, symbolic representation.

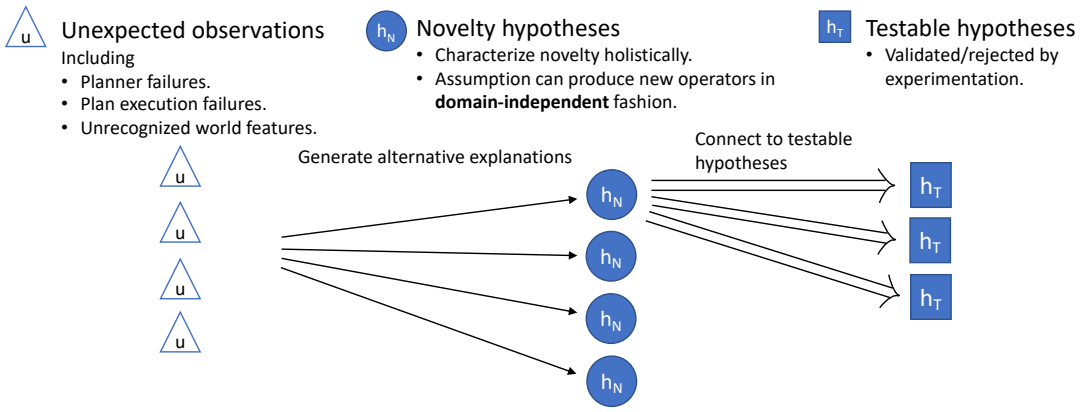

Figure 3: OpenMIND reasons about novelty hypotheses to adapt its planning models.

planning model changes that the agent will temporarily use for *experiments*, to see if the hypotheses are accurate and helpful. As it tries various new actions, OMMM observes the cost and success or failure of doing those actions, compares those to its baseline expectations about the world, and recognizes when the break-tree-with-axe action yields a lower cost. This *validates* the hypothesis, now representing a *quantum of valuable learned information*: OMMM will retain that validated hypothesis and the corresponding planning model changes in its future activities, using the axe any time it is available.

3.1 The OpenMIND Executive

The OpenMIND Executive robustly executes a sequence of parameterized actions (the plan) sent to it by OMMM. The Executive ensures reliable execution of the actions by checking preconditions against the current world state and handling failure conditions returned from the domain simulation. In most cases, a single planner operator translates to a single action sent to the domain simulation. In some cases, the Executive manages a sequence of actions sent to the simulation and may dynamically choose simulation commands based on world state. We have implemented the Executive with the OpenPRS (Open Procedural Reasoning System) (Ingrand et al. (1992)) kernel to use its reliable, fact-triggered procedure execution.

When the Executive encounters unexpected world states, action failures, or unmet preconditions, it initiates reasoning about possible novelties. OpenMIND is focused on *mission-relevant* novelty and explicitly checks world state related to the preconditions of planned actions. For example, crafting a pogo stick requires four sticks and OpenMIND will plan an action to produce four sticks from two planks. If the planned action unexpectedly produces a different number of sticks, the Executive will notice the violation of anticipated world state when it checks aspects of the state relevant to action preconditions. Interestingly, if the planned action unexpectedly produces *five* sticks (which satisfies the precondition for producing a pogo stick), the Executive will still detect the deviation from the expected state of the world, and OpenMIND will report novelty and replan, just as if the action produced too few sticks. This design choice reflects OpenMIND's bias towards

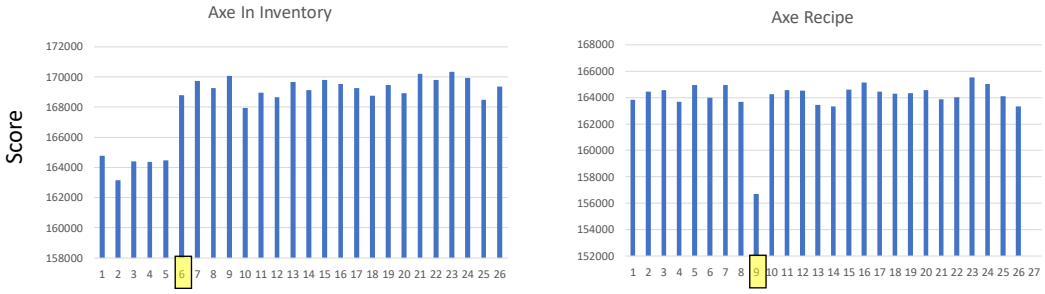

Figure 4: OpenMIND's scores over multiple games within two different trials show how it learns to use axes that are already available, and also learns not to make an axe from scratch (because it costs too much). Here, yellow boxes highlight the game in which novelty is introduced: an axe in inventory or the recipe for making an axe.

novelty detection. In the event there are sufficient sticks to make a pogo stick, OMMM will report novelty, and a replan will very likely re-derive the rest of the usual pogo stick plan.

Action failures also trigger signals from the Executive to OMMM to initiate the investigation of potential novelty. Although the Executive checks action preconditions, there are other sources of action failures in the SAIL-ON domains, including intended novelties. Some domains are sufficiently dynamic that a precondition may be violated by the expected evolution of the simulation state through environmental processes or the actions of other agents. For example, in some Polycraft scenarios, there is a rival "pogoist" who may harvest a tree that OpenMIND plans to use. In Vizdoom, the targets are actively moving. In some domains, action outcomes are sufficiently uncertain that actions may fail even when performed under the best conditions. For example, the control inputs for Sciencebirds are discretized in such a way that the launch angle and velocity of a projectile cannot be specified precisely enough to reliably hit a static target. And, in rare cases, there are small bugs or irregularities in the simulation which cause actions to fail. The Executive does not have the ability to reason about the underlying causes of these failures or distinguish their root causes. It lumps them together as action failures and sends a message to OMMM indicating the failure, relaying the new current state of the world, and requesting a new plan.

3.2 OpenMIND Mission Manager (OMMM)

OMMM constructs the action sequences that will achieve OpenMIND's goals. Under nominal conditions, this is a straightforward task, complicated slightly by the need to handle multiple heterogeneous domains and their simulators. When confronted by novelty, presented as action execution failure or unexpected elements in the sensed world state, OMMM must characterize the novelty, construct candidate plans that may overcome or exploit the novelty, and assess the quality of the execution of those candidate plans. To structure these tasks, we have chosen a framework based on domain independent hypotheses which describe the encountered novelty and algorithmically produce planning model changes to generate plans that handle the novelty. In this paper, we focus on five hypothesis types (described in Section 3.2.2) which we believe are surprisingly effective in

```
(:action craft____r4-polycraft_wooden_pogo_stick-default
 :parameters
  (?crafting-table - mc-crafting-table-type ?unk-loc - pcw-loc)
 :precondition
  (and
   (name ?unk-loc ?crafting-table)
   (>= (in-inventory polycraft__sack_polyisoprene_pellets) 1)
   (>= (in-inventory minecraft__planks) 2)
   (>= (in-inventory minecraft__stick) 4))
 :effect
  (and
   (decrease (in-inventory polycraft__sack_polyisoprene_pellets) 1)
   (decrease (in-inventory minecraft__planks) 2)
   (decrease (in-inventory minecraft__stick) 4)
   (increase (in-inventory polycraft__wooden_pogo_stick) 1)))
```

Figure 5: At runtime, OMMM translates simulator information into scenario-specific domain models. This is a possible instance of an action to construct a pogo stick derived from Polycraft output.

combination, and generally useful in problem domains where cognitive systems must handle novel changes to their environments.

3.2.1 Planning Models

OMMM maintains its knowledge about world state, state transitions, and goals as PDDL problems, domains, and plans[2]. In the nominal (no novelty) case, OMMM can generate a fully specified PDDL domain from its *a priori* knowledge of the domain, including the expected domain objects, applicable predicates, and default world transitions (including actions). Domain knowledge can be provided at agent build time or at run-time. For example, in Polycraft, the recipes for crafting actions are provided by the simulator at the beginning of a scenario. OMMM translates these recipes into domain transitions (see Figure 5).

Whenever the Executive is at a loss for its next action, for example when the scenario has just started, a planned action has failed, or the Executive has successfully executed all of the actions in a plan, it requests a new plan from OMMM. Each plan request includes a complete, symbolic snapshot of the world state in the form of `domain-fact` statements (see Figure 6). First, OMMM analyzes the incoming list of facts to identify any novel elements in the world state. Then, it constructs a PDDL problem instance, using the fact list, its current goal, and the effects of any currently active hypotheses.

Finally, OMMM invokes Metric-FF (or other PDDL-compliant planner) to solve its planning problem. There are no guarantees a plan will be possible within a reasonable amount of time, so OMMM must handle several possible outcomes of invoking the planner. The simplest case is when

2. Specifically, OMMM uses PDDL 2.1 with numeric fluents and limited derived predicates (Fox & Long (2003)).

```
((:type :plan-request)
 (:plan-name t0-g1-p12)
 (:reason "plan finished with game not over")
 (:factlist (. (domain-fact (selected-item polycraft.wooden_pogo_stick))
               (domain-fact (in-inventory polycraft.wooden_pogo_stick 1))
               [...]
               (domain-fact (name "20,4,20" minecraft.crafting_table))
               (domain-fact (isaccessible "20,4,20" true))
               (domain-fact (near minecraft.crafting_table)) .)))
```

Figure 6: The bridge and Executive translate the domain-dependent representation of the domain simulators into a list of `domain-fact` statements, included in the `plan-request`.

```
(define (problem t0-g0-p1-problem)
 (:domain pcw-domain)
 (:objects
   loc10_4_10 - pcw-loc
   loc18_4_26 - pcw-loc
   [...]
 (:init
   (= (in-inventory polycraft__bag_polyisoprene_pellets) 0)
   (= (in-inventory polycraft__sack_polyisoprene_pellets) 0)
   (= (in-inventory polycraft__wooden_pogo_stick) 0)
   (= (in-inventory minecraft__planks) 0)
   [...]
   (name loc3_4_17 minecraft__log)
   (variant loc3_4_17 oak)
   (axis loc3_4_17 y)
   (isaccessible loc3_4_17 true)
   [...]
   (near minecraft__log)
   (crafting-table-p minecraft__crafting_table))
 (:goal (= (in-inventory polycraft__wooden_pogo_stick) 1)))
```

Figure 7: OMMM constructs a PDDL problem instance from the world state, its goal, and active hypotheses.

```
(BREAK-BLOCK MINECRAFT__LOG LOC27_5_27)
(CRAFT____R5-MINECRAFT_PLANKS-DEFAULT)
(CRAFT____R6-MINECRAFT_STICK-DEFAULT)
(BREAK-BLOCK MINECRAFT__LOG LOC23_4_15)
(CRAFT____R5-MINECRAFT_PLANKS-DEFAULT)
(CRAFT____R3-POLYCRAFT_TREE_TAP-DEFAULT MINECRAFT__CRAFTING_TABLE LOC20_4_20)
(BREAK-BLOCK MINECRAFT__LOG LOC18_4_26)
(PLACE-TAP LOC8_4_2)
(EXTRACT-RUBBER LOC8_4_2)
(CRAFT____R5-MINECRAFT_PLANKS-DEFAULT)
(CRAFT____R6-MINECRAFT_STICK-DEFAULT)
(CRAFT____R4-POLYCRAFT_WOODEN_POGO_STICK-DEFAULT MINECRAFT__CRAFTING_TABLE LOC20_4_20)
```

Figure 8: OMMM uses Metric-FF to solve planning problems it has formulated from domain knowledge, current world state, and hypothesized novelties. This plan, if executed successfully, constructs a pogo stick in Polycraft.

```
(CHECK-ASSERTION (NAME "8,4,2"  MINECRAFT.LOG) CRITICAL)
(CHECK-ASSERTION (IN-INVENTORY POLYCRAFT.TREE_TAP 1) CRITICAL)
(PLACE-TAP "8,4,2" )
(CHECK-ASSERTION (TAP-PLACED-ON-TREE-AT "8,4,2" ) CRITICAL)
(EXTRACT-RUBBER "8,4,2" )
(CHECK-ASSERTION (IN-INVENTORY MINECRAFT.LOG 1) CRITICAL)
(CRAFT R5-MINECRAFT_PLANKS-DEFAULT)
(CHECK-ASSERTION (IN-INVENTORY MINECRAFT.PLANKS 5) CRITICAL)
(CRAFT R6-MINECRAFT_STICK-DEFAULT)
(CHECK-ASSERTION (IN-INVENTORY MINECRAFT.STICK 7) CRITICAL)
(CHECK-ASSERTION (IN-INVENTORY MINECRAFT.PLANKS 3) CRITICAL)
(CHECK-ASSERTION (IN-INVENTORY POLYCRAFT.SACK_POLYISOPRENE_PELLETS 1) CRITICAL)
(CRAFT R4-POLYCRAFT_WOODEN_POGO_STICK-DEFAULT MINECRAFT.CRAFTING_TABLE "20,4,20")
```

Figure 9: OMMM sends an execution sequence to the Executive that includes checks of critical world state and actions to be translated into the domain-specific simulator. This sequence includes the last four actions to construct a pogo stick in Polycraft.

the planner returns a goal-satisfying plan promptly. In that case, OMMM translates the plan into a sequence of actions that can be consumed and executed by Executive (see Figure 9). If the planner deems the problem unsolvable, OMMM begins its hypothesis reasoning cycle to determine whether there is a novelty causing its failure and search for a way to overcome the difficulty.

3.2.2 Hypothesis Reasoning Cycle

When OMMM detects novelty, it begins its hypothesis reasoning cycle to characterize the nature of the novelty and effectively respond to it. In many cases, the novelty creates opportunities for OpenMIND to achieve its goals more quickly or with less expense. In other cases, the novelty poses a threat to goal achievement that OpenMIND must overcome.

Novelty handling begins with detection, generally in the form of an unexpected observations (see Figure 3). In this paper, we consider detections in the form of action execution failures, planner failures, and the observation of previously unknown items or features. Note that each of these events can be detected in a domain independent fashion. In response to an unexpected observation, OMMM creates an instance of a hypothesis class. The instance is placed in a queue for consideration. When the hypothesis is active, OMMM uses it to modify its planning model, *e.g.*, by adding preconditions to an operator. While a hypothesis is in effect, OMMM is actively evaluating its validity and assigning credit/blame to active hypotheses.

To illustrate hypothesis handling, we describe the life cycle of several domain independent hypothesis classes which we have found to be very effective in SAIL-ON domains.

- Bad Args – One or more Bad Args hypotheses may be created in response to the detection of an execution failure, particularly when the failed action is considered to be very reliable. The notion is that the unexpected failure of an operator may be caused by unobserved or misunderstood feature of one of the arguments. For example, in Polycraft, OpenMIND may try to collect a log from a particular tree which for some reason is immune to the "break block" action. When the Bad Args hypothesis is active, OMMM will dynamically modify the PDDL operator for `break-block` to check for blacklisted parameters (Figure 10). If the OpenMIND's plan execution succeeds with the model modifications, the hypothesis will be considered confirmed and remain active.

- Remove Novelty – In scenarios with repeated negative outcomes such as planner failures and action failures, in the presence of detected novelty, OMMM will consider a Remove Novelty hypothesis. This hypothesis class represents the theory that the new element in the domain is interfering with planner or action success and that OpenMIND's goals would be best served by removing as much of the novelty as possible before resuming pursuit of the goal. This domain independent approach to novelty requires some domain dependent knowledge in the form of "grounding methods" that operationalize novelty removal for that domain. For example, in Sciencebirds, the grounding method for novelty removal may shoot powerful birds at novel scenario elements.

- Do Anything – When more specific hypotheses fail to find a way past negative outcomes, OMMM can apply a Do Anything hypothesis, based on the observation that in goal achieve-

```
(:action break-block :parameters (?item - mc-item ?loc - pcw-loc)
 :precondition
 (and
   (name ?loc ?item)
   (isaccessible ?loc true)
   (not (crafting-table-p ?item))
   (not (break-block-bad-args ?item ?loc)))
 :effect
 (and
   (increase (in-inventory ?item) 1)
   (not (name ?loc ?item))))
```

Figure 10: When a Bad Args hypothesis is in effect, OpenMIND will create a new version of the subject operator that supports parameter blacklisting.

ment domains doing nothing is almost never the right thing. As with Remove Novelty hypotheses, a domain dependent grounding method must be provided to make the Do Anything hypothesis actionable. For example, in Polycraft the grounding method instructs OMMM to use the `break-block` operator on a randomly selected item.

- Preferred Operator – The Preferred Operator hypothesis seeks to find new opportunities presented by novelties. When a novel element appears in the scenario, OMMM will create Preferred Operator hypotheses for operators in its domain model. As it tests each of these hypotheses in turn, it modifies its PDDL models to prefer operators that can make use of the novelty in some way. For example, as described above, when an "axe" appears in the Polycraft domain the agent will form a new operator use the axe to break a tree into logs, and create a preferred operator hypothesis to encourage the planner to try using that action whenever an axe is available. If using the axe yields logs at lower cost than the normal use-hand operator, the preferred operator hypothesis is validated and will be used in the future. The left plot in Figure 4 illustrates this case. Preferred Operator hypotheses are confirmed and retained when they result in better scenario outcomes.

- Useful Prefix Goal – The detection of a novel item or feature will also trigger the creation of Useful Prefix Goal hypotheses to obtain an example of the novel object. When a Useful Prefix Goal hypothesis is active, the usual domain goal is replaced in the planning model by the goal to obtain the novel element. For example, when the agent receives a recipe for how to make an axe, but no axe is already present, it will create a hypothesis that having a new goal to create the axe at the start of the game would be a good idea, so that it can be used by the hypothesized preferred operator (also created, as above) to make logs at lower cost. In some situations, however, the process of making the axe costs so much that it overwhelms the value of using the axe to make the pogo stick. In that case, the agent will recognize that its overall score for the game is lower than expected, and will reject the useful prefix goal hypothesis and not try making the axe again. The right plot in Figure 4 illustrates this case.

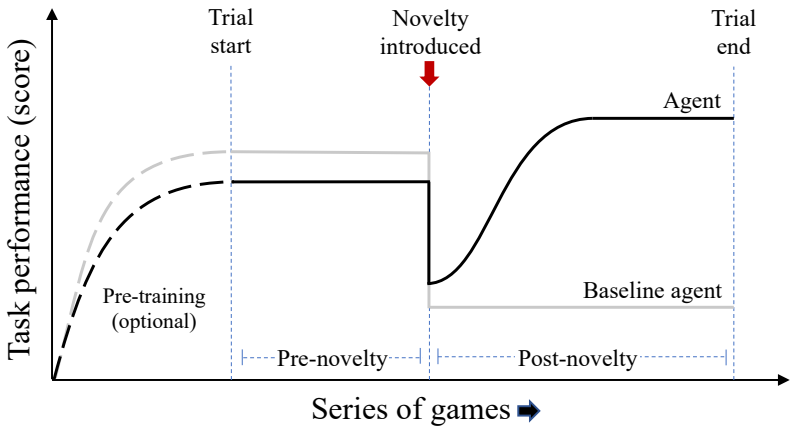

Figure 11: The format of a trial in a SAIL-ON evaluation.

4. Experiments and Results

In SAIL-ON, OpenMIND is evaluated in multiple domains by the developers of each domain. As agent developers, we are blind to the details of evaluation process, to better ensure that the agent design is not influenced by the specific novelties chosen for the evaluation and so that novelties can be reused across evaluations to assess improvements in the agent over time. Multiple agents from different SAIL-ON performers may be evaluated in the same domain, and all agents have been evaluated multiple times, at six-month intervals. The domain developers have published a handful of novelties for the agent developers, in part to debug the mechanics of the evaluation process. Domain developers also provide a baseline agent - a domain-specific agent that is intended to perform the task while being oblivious to novelty.

The evaluation of an agent consists of a battery of *trials*, typically on the order of one hundred. The novelties for these trials are chosen to cover multiple categories spanning three difficulties (easy, medium, and hard) and three types of novelty: classes, properties, and representations[3]. Many trials are performed for each novelty, to provide some statistical confidence.

The domain providers use various methods to design and adjust the difficulty levels, including changing scales of effects (e.g., a few fence blocks surrounding only some trees is easy, while several layers of fence blocks surrounding every tree is hard). The difficulty may also be based on how much effort is needed to explore and understand the novelty. For example, in Polycraft, the easy version of the revealed axe novelty (a new class of object) provides an axe lying around in the world (which is an opportunity to improve score), while the hard axe novelty only provides the axe recipe, which is actually a decoy or red herring type of novelty, since making an axe uses so many actions that it costs more than having the axe saves while breaking logs for the actual pogo stick task.

To help explain the SAIL-ON evaluation metrics, Figure 11 depicts the elements and structure of a trial. Each trial consists of a series of games. The agent retains its knowledge from game to game,

3. In future evaluations, SAIL-ON will expand the range of novelty types.

but is reset to its initial tabula-rasa state between trials. Each trial is optionally preceded by a training period to allow the agent to reach its asymptotic (non-novel) task performance before trial start, but this is irrelevant for OpenMIND, which is expected to initialize at its asymptotic (best) level of performance. At trial start, games are randomly drawn from the non-novel distribution of games for the domain. At some variable point after the trial starts, games begin to be only drawn from the novel distribution, marking the end of pre-novelty and the beginning of the post-novelty phase. During the post-novelty phase, the SAIL-ON agent is intended to adapt to the novelty, eventually reaching its asymptotic performance given the new conditions, whereas the baseline agent, not designed to handle novelty, is not expected to adapt.

The metrics fall into two categories: detection metrics and adaptation metrics. Detection metrics characterize how accurately the agent is able to detect when novelty has been introduced. These metrics are various functions of false positives (games before the novelty has been introduced in which the agent falsely reports that novelty has been introduced), and false negatives (games after novelty has been introduced, in which the agent fails to report that novelty has been introduced). True positives are games after novelty has been introduced in which the the agent correctly reports that novelty has been introduced. A correctly detected trial is a trial which contains no false positive games and at least one true positive game. The three central SAIL-ON detection metrics are (1) the average number false positives in correctly detected trials (FN_CDT; smaller is better), (2) the percentage of trials that are correctly detected trials (CDT_%; larger is better), and (3) the percentage of trials that contain at least one false positive (smaller is better; FP_%). These metrics trade off. An agent that is highly sensitive is more likely to report novelty in general, resulting in some "false" or "artificial" reporting. If this false reporting first occurs in a trial after the introduction of novelty that would otherwise go undetected, then it will result in some artificial true positives in a trial which contains no false positives, potentially artificially driving up CDT. However, this potentially also increases FN_CDT, depending on the gap between the true introduction of novelty and the artificial report. Such a sensitive agent is also more likely to incur false positives, since the false report may come before the true introduction of novelty.

Adaptation metrics characterize how well the agent performs its task in the face of novelty; *i.e.*, how well it adapts to the novelty. The primary adaptation metric in the SAIL-ON program is Novelty Reaction Performance (NRP). NRP is the ratio of the agent's asymptotic post-novelty score on the task to the baseline agent's mean pre-novelty score on the task. It captures the degree to which an agent is able to recover to normal performance levels by trial end (although compared to the baseline agent, not itself).

$$NRP = \frac{score_{agent,post,asymptote}}{score_{baseline,pre,mean}}$$

Note that the baseline agent, tuned to the non-novel domain, may well initially outperform the adaptive agent, but the adaptive agent is expected to outperform the baseline, non-adaptive agent after some period of novelty. This may be because the non-adaptive agent fails to take advantage of opportunities or loses score by failing to adapt to interfering novelties that prevent pre-novelty normal behaviors from succeeding. For example, a non-adaptive agent using the Polycraft teleport commands cannot succeed in the hard fence domains, when all trees are blocked.

Two additional adaptation metrics compare the agent's post-novelty performance to baseline agent's post-novelty performance (as opposed to comparing to the baseline's *pre-novelty* performance, like NRP). The Overall Task Performance Improvement (OTPI) measures the ratio between the area under the agent's post-novelty performance curve and the total (summed) area under the post-novelty performance curves of both agent and baseline. The Asymptotic Task Performance Improvement (ATPI) measures a similar ratio, between the agent's asymptotic post-novelty performance and the sum of the asymptotic performances of both agent and baseline. For both of these metrics, a value greater than 50% reflects that the agent is outperforming the baseline agent in the face of novelty. The primary difference is that OTPI is more sensitive to the speed with which the agent adapts.

4.1 Two Evaluations

For the last two rounds of evaluation— month 12 (M12) and month 18 (M18) of the SAIL-ON program— our OpenMIND agent has been evaluated in two domains, Polycraft POGO and Sciencebirds. Since we started building OpenMIND for just the Polycraft domain and then started extending it for Sciencebirds at month 6, our agent was more advanced in some ways in Polycraft than Sciencebirds.

By M12, we had developed the entire hypothesis reasoning cycle described above, as well as versions of each of the novelty-handling hypotheses, with grounding methods as needed for the Polycraft domain. For example, the remove-novelty hypothesis is grounded in Polycraft by the knowledge that BREAK-BLOCK can make a novel block disappear from the environment. Similarly, in Sciencebirds, shooting a bird at an object (pig or not) can make it disappear, so that novelty hypothesis applied easily to Sciencebirds and was working at M12.

Some of the novelty-hypotheses do not need domain-dependent grounding methods. For example, the Bad Args hypotheses do not require any domain-dependent elements, they apply to any PDDL operator that has arguments (as most typically do). So, for Polycraft at M12 the Bad Args hypothesis could determine that a "ghost axe" that we injected into the sensor data (but was not really present in the simulation) could not actually be picked up, and the agent should stop trying. Similarly, in Sciencebirds, at M12 the Bad Args hypothesis could determine that an invulnerable pig could not be eliminated, and the agent should target a different pig.

Both M12 evaluations were impacted by several bugs in both the domain simulations and OpenMIND's behaviors and interface. For example, in Polycraft it was found that some blocks, when being broken near a tree-tap, would disappear from the agent's sensor data because the broken block landed on top of the tree-tap, removing it from the single 2D level sensed by the agent. UTD removed runs where that behavior was detected by our agent (and reported as novelty— disappearing blocks!), so that bug did not affect the results data shown below. In OpenMIND's, an M12 bug in our agent made it randomly report novelty on non-novel objects occasionally, so that *the agent surprised us* by detecting novelties for which we had built no detection mechanisms. ANU was able to derive methods to detect this pattern in our data and also eliminated those runs from revised M12 results.

Differences in our agent from M12 to M18 included numerous OpenMIND bug fixes and workarounds for domain simulation bugs, as well as improvements in both domain-dependent and

Table 1: Results on adaptation metrics

	NRP		OTPI		ATPI	
	M12	M18	M12	M18	M12	M18
Polycraft	81.7%	81.0%	50.5%	68.8%	56.1%	76.3%
Sciencebirds	84.0%	182.8%	50.9%	54.6%	50.7%	54.5%

Table 2: Results on detection metrics

	FN_CDT		CDT_%		FP_%	
	M12	M18	M12	M18	M12	M18
Polycraft	2.4%	0.6%	82.0%	88.5%	4.6%	0.0%
Sciencebirds	27.8%	23.0%	68.3%	27.2%	12.6%	0.6%

domain-independent functions. For example, we extended the Bad Args hypotheses to handle situations where not just one argument is responsible for a bad outcome, but some subset of the PDDL's operator's bound arguments are to blame and should be avoided. So far we only have homegrown tests that illustrate this working in Polycraft, but it should apply in Sciencebirds (and may have during the blind evaluations, we still don't know). In Sciencebirds, we also improved the colormap classifier, aiming algorithms, and perception-translation detection.

For M18, we also developed a new Do Anything fallback behavior when no novelty is perceived (so Remove Novelty and several other hypotheses cannot apply) and yet all plans seem to mysteriously fail to execute correctly, or the agent thinks it has achieved all the goals and yet the simulation doesn't agree the game should be over (*e.g.*, if a normal-looking Sciencebirds block is actually being treated as a pig that needs to be shot). In this situation, since the domains include no other source of change (so far), the agent *must* try something to handle the imperceptible obstruction. This new hypothesis is readily grounded in Sciencebirds (shoot at any object), since the action space is more limited. In Polycraft at M18 it was roughly equivalent to the remove-novelty grounding (break-block on any accessible object), but has since been expanded to support the broader action-space.

4.2 Results

In Table 1, we report our mean (across all trials) NRP, OTPI, and ATPI scores for M12 and M18 in these two domains. OpenMIND clearly performed very well on both domains across all three metrics. The NRP scores show that OpenMIND could overcome novelty and recover to more than 80% of baseline agent pre-novelty performance on average in all four conditions. Similarly, the OTPI and ATPI scores show that OpenMIND outperformed the baseline agent in the face of novelty on average under all conditions and sometimes outperformed by a wide margin, such as on the month 18 Polycraft trials.

In Polycraft, our results showed a marginal drop (<1%) in NRP from month 12 to month 18, but it coincided with a much more significant jump in OTPI and ATPI (>18% and >20%, respectively). These results indicate that in this domain, our agent's post-novelty performance remained consistent with respect to the baseline's pre-novelty performance while building a significant advantage over

the baseline agent's post-novelty performance. This suggests that the set of novelties for this domain in the month-18 evaluation was more challenging with respect to adaptation than was month-12, which sheds a more positive light on our relatively consistent NRP from month 12 to month 18.

In Sciencebirds, we saw significant gains in NRP from month 12 to month 18, with our agent in month 18 exceeding by 82.8% the pre-novelty performance level of the baseline by trial end, on average. The significance of this achievement is tied to the nature of novelties; if they were all purely obstacles, this would represent full recovery (with respect to baseline, at least), but some novelties may be opportunities, for which this level of NRP would not be impressive. Our agent's OTPI and ATPI only rose by roughly 4% and 3% respectively from month 12 to month 18 in this domain, implying that while OpenMIND 's ability to adapt improved in this domain, it was a more modest improvement than the jump in our NRP would suggest; we likely owe some of this jump to an easier batch of novelties in the month-18 evaluation than in month 12 for the Sciencebirds domain.

Table 2 shows our results in the Sciencebirds and Polycraft domains on the three detection metrics listed above. All metrics moved in the preferred direction in the Polycraft domain between month 12 and month 18. In Sciencebirds, our agent was sensitive to false detection of novelties per the discussion of trade-offs above. We eliminated those false detections almost entirely by month 18, resulting in a reduced CDT (not preferred) alongside a reduced FP_% (preferred) and minor reduction in FN_CDT (preferred). A subsequent analysis by our Sciencebirds evaluators revealed that, when ignoring detections that showed signs of being artificial, our adjusted CDT actually remained virtually constant between month 12 and month 18, indicating that our ability to truly detect novelties did not suffer in the same period in which we almost eradicated false reports.

5. Related Work

As noted earlier, experienced readers will recognize many common themes in our OpenMIND agent architecture and its functions. Here we touch on only a few related works, highlighting key inherited ideas and differences in our approach. Our technical contributions lie primarily in the domain-independent computational methods we have developed for modifying agent goals and planning models, leveraging the information available from detected novelty aspects, while remaining flexible and applicable in a wide variety of situations. In a sense, we're researching ways to implement general novelty-handling strategies that are closely related to the extensive literature on creative problem solving strategies (for both humans and machines, *e.g.* Fobes (1993)).

5.1 Multi-Layered Agent Architectures

One distinguishing aspect of our work on OpenMIND has been the focus on remaining open to novelty at all levels of the world representation. For example, the OpenMIND Executive can handle novel features of the world (*e.g.*, trees are now perceived with a bark-type). In the real world, this would correspond to the agent acquiring a new sensing modality, such as a Mars Rover cannibalizing an older, disabled rover to acquire a new infrared sensor. In contrast, many other plan executives need to have the complete set of perceived features declared at compile or load-time (usually to efficiently index their responses). Similarly, OMMM can reason about and manipulate its goals

explicitly, including creating new goals based on never-before-seen objects and features (*e.g.*, chop all trees with (bark-type rough)).

5.2 Adapting Planning Models

Arora et al. (2018) present a useful overview of a wide variety of learning methods applied to planning models, both online and offline. Their section on Surprise Based Learning relates most closely to the SAIL-ON challenges and our approach, which generalizes prior work on detecting violated expectations and modifying planning models accordingly. Perhaps the largest distinction between those efforts to learn planning models and ours is that our work has been assessed in blind evaluations against novel domain elements the system creators never knew about, lending credence to the claim that they were truly novel and unexpected.

6. Conclusions

We have developed the OpenMIND agent architecture to remain as open as possible to novelty in its environment: the system can handle new, never-before-seen perceptions (features of the world, objects), new actions, new goals, and new world dynamics. In single-blind evaluations of OpenMIND's ability to detect and respond effectively to novelty, initial performance results are encouraging. The SAIL-ON program has spurred significant advances in domain-independent methods for robustness in the face of truly novel domain elements, and the work is ongoing. We will be enhancing OpenMIND to handle more types of novelty in each of the three domains it has been applied to so far, and the domain developers will conduct several more evaluations. Future challenges including generalizing some of the system's more recent domain-dependent capabilities that were required to perform in new domains, such as route-planning and targeting, to create more domain-independent functions for future applications.

6.1 Lessons Learned?

Readers may wish for a series of ablation studies or other experiments to determine which of OpenMIND's capabilities are responsible for its performance improvements, but therein lies the rub of the scientifically-valid evaluation process. We have not yet been given the full set of M12/M18 evaluation novelties, and so we cannot perform those ablation studies or know conclusively which knowledge/behavior improvements led to which performance improvements. Furthermore, it is not clear what such studies would really show: would they illustrate that our domain-independent novelty-handling methods correctly anticipated the precise set of novelties the domain-developers created, or that they have a more-general validity?

6.2 Bug or Novelty?

While we have touted the scientific soundness of blind evaluation on novelties, in practice these evaluations have revealed an unexpected conundrum: on unrevealed novelties, there's no perfectly clear way to distinguish between bugs in our agent, bugs in the domain simulations, and intended novelties. For example, the simulation creators may think that their simulation is producing a novel

world state, but there may be a bug so that it is not; then in the evaluation, the assessors expect our agent to report novelty but it does not. They can tell us that it had a "false negative," but not give us the domain to try to reproduce the problem and diagnose the flaw. In **more than one case** on revealed novelties, the simulator has actually been broken so that the novelty was not being created at all.

Conversely, the simulator creators might be surprised that our agent is reporting novelty in a supposedly non-novel domain. For example, UTD reported that for some non-novel Polycraft scenarios our agent didn't make the pogo stick. After some debugging, we realized that our agent was doing the best it could, but there were simply not enough resources (trees) in the starting situations of those scenarios to make the goal possible. UTD agreed, found the problem in their game creation code, and the issue was resolved with no change to our agent.

The bugs can be on either side, or both sides, and that's part of the conundrum! In yet another bug-vs-novelty situation, our agent performed unexpectedly well at detecting novelty in Sciencebirds in our first blind evaluation. The evaluation showed that the agent reported novelty on many types of novelty for which it had no detection mechanisms at all, so we were quite surprised. However, on deeper inspection of long-duration logs for *non-novel* trials, we found that the perception system was rarely and semi-randomly detecting spurious false novelty. These random "fake" detections would apparently also trigger during the blind evaluation of novel trials, giving us an improved score despite the fact that the novelty detection was really an artifact of an OpenMIND flaw.

Acknowledgements

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001120C0041. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA.

We would like to thank all the SAIL-ON performers for their collaborative efforts on the program, particularly the UTD, ANU, and WSU teams who have worked hard to make the evaluations of OpenMIND and other agents possible.

References

Albus, J., et al. (2002). *4D/RCS version 2.0: A reference model architecture for unmanned vehicle systems*. Technical Report NIST Interagency/Internal Report (NISTIR) - 6910, NIST.

Arora, A., Fiorino, H., Pellier, D., Etivier, M. M., & Pesty, S. (2018). A Review of Learning Planning Action Models. *Knowledge Engineering Review*, *33*. From `https://hal.archives-ouvertes.fr/hal-02010536`.

Bonasso, R. P., Kortenkamp, D., Miller, D., & Slack, M. (1996). Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical AI*.

Boult, T. E., et al. (2021). Towards a unifying framework for formal theories of novelty. *Proc. National Conf. on Artificial Intelligence* (pp. 15047–15052). From `https://ojs.aaai.`

org/index.php/AAAI/article/view/17766.

Cox, M. T. (2016). A model of planning, action and interpretation with goal reasoning. *Advances in Cognitive Systems 4*.

Cox, M. T., & Dannnenhauer, Z. A. (2017). Perceptual goal monitors for cognitive agents in changing environments. *Advances in Cognitive Systems 5*.

Fobes, R. (1993). *The creative problem solver's toolbox*. Solutions Through Innovation.

Fox, M., & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*.

Ingrand, F. F., Georgeff, M. P., & Rao, A. S. (1992). An architecture for real-time reasoning and system control. *IEEE Expert*, (pp. 34–44). From http://www.laas.fr/~felix/publis/publis.html.

Klenk, M., Molineaux, M., & Aha, D. W. (2013). Goal-driven autonomy for responding to unexpected events in strategy simulations. *Computational Intelligence*, *29*. From http://matthewklenk.com/papers/CI-12-GDA.pdf.

Kuipers, B., Feigenbaum, E., Hart, P., & Nilsson, N. (2017). Shakey: From conception to history. *AI Magazine*, *38*, 88–103.

Langley, P. (2020). Open-world learning for radically autonomous agents. *Proc. National Conf. on Artificial Intelligence*.

Musliner, D. J., Durfee, E. H., & Shin, K. G. (1993). CIRCA: A cooperative intelligent real-time control architecture. *IEEE Trans. Systems, Man, and Cybernetics*, *23*, 1561–1574.

Renz, J., Ge, X., Gould, S., & Zhang, P. (2015). The Angry Birds AI competition. *AI Magazine*, *36*, 85–87. From https://ojs.aaai.org/index.php/aimagazine/article/view/2588.

UTD (2020). PolyCraft World. https://polycraft.utdallas.edu/. Retrieved September 6, 2021.

Vodnala, S. (2019). IQ test for artificial intelligence systems. https://news.wsu.edu/2019/12/12/iq-test-artificial-intelligence-systems/. Retrieved September 6, 2021.

Wydmuch, M., Kempka, M., & Jaśkowski, W. (2018). Vizdoom competitions: Playing doom from pixels. *IEEE Transactions on Games*.