Physics-Based Novel Task Generation through Disrupting and Constructing Causal Interactions

Chathura Gamage[1,*], Matthew Stephenson[2], and Jochen Renz[1]
[1]School of Computing, The Australian National University, Canberra, Australia
[2]College of Science and Engineering, Flinders University, Adelaide, Australia

***Abstract*—In response to the growing demand for AI systems that can operate the physical world, there has been an increasing interest in enhancing their physical reasoning capabilities. Equally crucial is the ability to handle unseen novel situations, as such situations frequently arise in real-world environments. To facilitate the development of AI systems with those abilities, researchers have developed testbeds with specialized tasks to evaluate agents' adaptation to novelty in physical environments. In this paper, we propose a method for generating physics-based tasks with incorporated novelties to assess agents' novelty adaptation capabilities. The tasks are defined as causal sequences of physical interactions between objects, and novelties are strategically introduced to disrupt existing causal relationships and construct new ones. This approach ensures that agents must adapt to the effects of novelties to perform those tasks, enabling confident measurement of their novelty adaptation capabilities using task performance. Moreover, our methodology eliminates the need for manual task creation, unlike existing novelty-centric testbeds. The proposed method is demonstrated and evaluated using 12 physical scenarios in the Angry Birds domain. The evaluated metrics include generation time, physical stability, intended solvability, intended unsolvability, and accidental solvability of the tasks, and they yielded favourable results compared to the literature.**

***Index Terms*—Novelty Generation, Physics-Based Tasks, Open-World Learning, Physical Reasoning.**

I. Introduction

THE physical reasoning capabilities of AI systems are crucial for their successful operation in real-world environments. However, merely possessing the ability to reason about physics is insufficient in dealing with the complexities of the real world. In our daily lives, characterized by an open-world environment, we frequently encounter novel situations that have not been encountered before. Hence, it becomes imperative for AI systems in open physical worlds to not only possess physical reasoning capabilities but also the capacity to handle novel situations effectively. Within the research domains such as Open World Learning (OWL), efforts are being made to equip AI systems with the necessary abilities to adapt efficiently and proficiently to novel situations [1], [2]. Moreover, researchers are developing testbeds to facilitate the development, experimentation, and evaluation of these AI systems [3], [4].

The concept of novelty for an AI agent entails a transformation within the environment that influences the agent's understanding of the environment itself [2], [4]. When an AI agent encounters an unexpected change in the environment it was trained on, it typically struggles to perform adequately. Here we term the tasks that incorporate these novelties as novel tasks, while tasks without any novelties as normal tasks. To facilitate the development of AI systems capable of handling novelties in physical environments, researchers have utilized physics-simulating environments to construct testbeds encompassing both normal and novel tasks [3]. When evaluating an AI system's performance in the presence of novelties, it is crucial that the novel tasks present meaningful challenges to the AI, rather than merely acting as nuisances that can be disregarded by the agent. To achieve this, we believe the novelty should significantly impact the task's solution, necessitating the AI to adapt its actions accordingly. However, generating such novel tasks automatically in physics-based environments remains a challenge due to their inherent complexity. Despite recent developments in this research area, there is a lack of existing works focusing on the automatic generation of such novel tasks.

This work introduces a procedural task generation method designed to create physics-based tasks that incorporate novelties. Our approach involves generating tasks in pairs, consisting of both normal tasks and novel tasks, to align with the standard OWL evaluation protocol [5]. We propose a systematic approach that extends a recently introduced physics-based task-generation method, which defines physical scenarios as causal sequences of physical interactions between objects involved in the solution of the scenario [6]. Through the introduction of novelties, the established causal interactions in the normal task's interaction sequence are disrupted, while new causal interactions are constructed in the novel task's interaction sequence. This approach ensures that the introduced novelties impact the task's solution, thereby necessitating the AI system to make appropriate adjustments in response. By generating tasks that incorporate novelties in this manner, we enable a robust evaluation of the AI system's ability to adapt to novel tasks.

In this study, we showcase the application of our approach using the physics-based puzzle game, Angry Birds. The choice of this game is driven by its widespread use in physical reasoning [7], [8] and OWL research [9], [10], along with its realistic physics simulation environment that aligns with our objectives. We commence by providing an overview of related work, situating our contribution within the existing literature, and formulating the theory of designing tasks to confidently evaluate agents' novelty adaptation. Additionally,

*Corresponding author. Email: chathura.gamage@anu.edu.au

we define the scope of novelties considered in this study. Then, we discuss the causal sequence-based task-generation method employed in this research and present our proposed extensions to enable novelty-centric task generation in physical environments. We then present the step-by-step process of generating tasks as pairs of normal and novel tasks for a given physical scenario. Lastly, we evaluate the generated tasks using a comprehensive set of metrics. These metrics encompass the assessment of the runtime of the generator, the physical stability of the objects in the tasks, the solvability of the tasks using intended solutions, the intended unsolvability of the tasks using other solutions, the shift of the solution between normal and novel tasks, and the accidental solvability of the tasks using unintended solutions. Using the evaluation results, we showcase the capabilities of our methodology in generating sophisticated tasks that adhere to our design theory, thus facilitating sound assessments of OWL AI systems.

II. Related Work

In this section, we provide a comprehensive overview of the relevant literature in the fields of physics-based task generation, novelty-centric domains, and novelty generation.

A. Physics-Based Task Generation

Here we explore the two main research areas that extensively focus on physics-based task generation: game content development for physics-based video games and physical reasoning benchmarks and testbeds.

In the realm of physics-based games, researchers have extensively explored Angry Birds [11] and Cut the Rope [12] for generating game content. Angry Birds, in particular, has gained significant attention within the procedural content generation research community and has been investigated from various angles. These investigations have included the development of techniques for generating physically stable structures [13], converting hand-drawn sketches into stable game levels [14], dynamically adjusting the difficulty of game levels [15], generating deceptive levels for AI agents [16], and utilizing prompt engineering to create prompts for generating physically stable structures [17]. A recent and intriguing approach to physics-based task generation, specifically for Angry Birds, involves defining physical scenarios through causal sequences of physical interactions between objects [6]. In this method, a scenario is defined by its solution, which is represented as a sequence of physical interactions among the objects involved. Tasks are generated based on this scenario definition, adhering to the specified interaction sequence. This approach's significant advantage lies in well-defined tasks using the underlying physics mechanics, enabling systematic evaluations of AI systems' physical reasoning capabilities and identifying specific areas where agents may encounter challenges using the generated tasks. Building upon these advantages, our work adapts and extends this task generation mechanism to incorporate novelties.

Physics-based benchmarks and testbeds have emerged as crucial resources within the research community, serving as tools for evaluating the physical reasoning capabilities of AI agents. While those environments contain various types of physical reasoning tasks based on images [18], videos [19]–[21], and actions [7], [22]–[24], our work primarily aligns with task creation methods in action-based environments. In action-based environments, agents are required to take actions within the physical environment to complete tasks successfully. Task generation in such environments heavily relies on handcrafted task templates created by developers. When generating tasks, these templates undergo slight variations, such as altering object positions or introducing distraction objects. However, the manual creation of these task templates is a tedious and labour-intensive process that demands domain expertise, adherence to physics laws, ensuring stability under gravity, and designing with enough flexibility for variations. In contrast, our proposed methodology for the task generator relies on a textual input that serves as the task definition, comprising an intuitive sequence of physical interactions. This eliminates the need for pre-created templates and allows for a more streamlined and efficient task generation process. Furthermore, this approach enables the systematic classification of tasks to different physical scenarios based on their associated physical interactions, facilitating more comprehensive evaluations of agents across different classes of physical scenarios.

B. Novelty-Centric Domains and Novelty Generation

Novelty has been defined by researchers from various perspectives. Some define it in terms of an agent's model of the external world, where novelties are situations that violate implicit or explicit assumptions in an agent's model [1] or situations that surpass their cognitive capabilities based on past experiences [25]. Others define it with respect to the environment, viewing it as transformations of elements within the environment [2]. To facilitate advancements in OWL research, researchers have developed several novelty-centric domains. These include GNOME [4], based on the board game Monopoly, NovGrid [26] and NovelGridWorlds [27], which are grid-based environments like MiniGrid [28], and NovelCraft [29], based on Minecraft. Additionally, there are physics-based novelty-centric domains such as CartPole Novelty [30], which incorporates novelties into the CartPole domain, and ScienceBirds Novelty [9], which introduces novelties into Angry Birds.

For these novelty-centric domains, researchers have also proposed various approaches for generating novelties [4], [31]. For instance, a novelty generation framework specifically designed for Angry Birds introduces a method to generate novelties that can be 'detected' by AI agents [10]. In that framework, users can define an objective of introducing novelty, such as reducing the passing rate of a level by 5%, through changes in the friction of a specific block type. The framework employs an iterative generate and test approach, systematically varying properties of novel objects and randomly placing them within tasks until the novelty objective is achieved. The primary focus of this framework is on generating detectable novelties with no guarantee of adaptability to those novelties. However, in the context of OWL, the emphasis is placed more on novelty adaptation rather than mere detection [3].

The ultimate expectation from a system is its ability to adapt to novel situations rather than merely recognizing that there is a novelty while failing to respond effectively. In our work, we explicitly ensure the adaptability of the generated tasks and ensure that the effect of the novelty has to be considered when adapting (i.e., tasks with novelties must be solvable, and the novelty's effect must be considered in task-solving), allowing us to conduct rigorous evaluations of AI agents' capabilities in handling novelties.

NovPhy [3] is a recently developed testbed that utilizes the ScienceBirds Novelty framework. Similar to the objective of generating tasks in our work, NovPhy aims to evaluate agents' performance in physical reasoning tasks under the influence of novelties. NovPhy consists of tasks for five physical reasoning scenarios, such as rolling objects, falling objects, and sliding objects. The testbed encompasses tasks created by combining each physical scenario with eight novelties. Task generation in NovPhy follows a template-based approach, similar to the one in previously discussed physics-based testbeds, where a hand-crafted template is modified by adjusting object positions and adding new objects as distractions to generate new tasks. Due to this template-based approach, the task generation method in NovPhy also exhibits inefficiencies as discussed earlier. Our approach eliminates the necessity of template implementation, opting instead for a textual description of tasks using a defined grammar. This shift not only reduces the manual labour involved in implementing novelties and task templates but also facilitates systematic agent evaluations as the physical interactions involved in solving those tasks are well-defined and known to the evaluator.

III. Angry Birds and Novelty Scope

In this section, we discuss our test domain, Angry Birds, and define the scope of novelties considered in our study.

A. Angry Birds

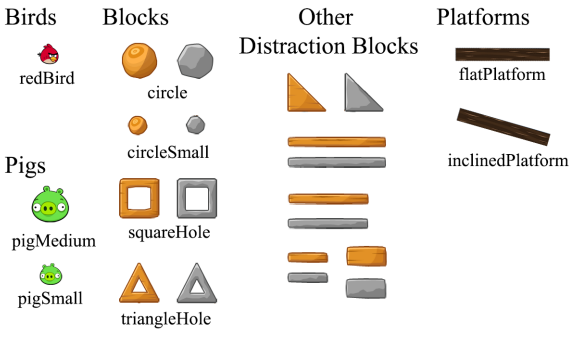

Fig. 1. The game objects in Angry Birds used in this study. The platform objects may vary in size, while the remaining objects maintain a fixed size. The brown-coloured blocks represent wood blocks, whereas the grey blocks correspond to stone blocks. The stone blocks possess higher mass, increased friction, and reduced bounciness compared to wood blocks. The blocks shown as Other Distraction Blocks are those that are not directly involved in the interactions required to solve the task. Instead, they serve as distractions for the agents by being placed in the task space.

Angry Birds is a 2D physics-based game where players aim to destroy pigs by launching a given number of birds using a slingshot. Figure 1 depicts the game objects from Angry Birds utilized in our study. A game level typically comprises a slingshot, birds, blocks, pigs, and platforms. Birds, blocks, and pigs are dynamic objects that follow Newtonian physics principles. These objects also possess health points that decrease upon collisions, leading to destruction when their health points reach zero. On the other hand, platforms remain static and are unaffected by external forces.

For our research, we utilize Science Birds [32], a research clone of Angry Birds implemented in Unity, incorporating the Box2D physics engine. To better showcase or methodology for the original Angry Birds environment, minor adjustments were made to the Science Birds framework to better align its behaviour with that of the original game. The two most impactful changes were a reduction in the object fragility by increasing their health points and improved object dynamics by fine-tuning various physics parameters. This 'enhanced' version of Science Birds can be downloaded at the following link[1], which includes a complete list of changes from the original Science Birds.

B. Novelty Scope

The abundance of novelties that can be created in a physical environment is a result of the limitless diversity and continuity present in the physical world. With an extensive variety of objects, properties, interactions, and dynamics, the potential for modifying or combining these elements knows no bounds, leading to an infinite array of possible novelties. For instance, one can introduce novelties by altering physical properties such as mass or friction along a continuous spectrum, introducing novel objects with varying shapes and sizes, or devising new interaction mechanisms between objects. The sheer magnitude of possibilities renders it impractical to exhaustively enumerate and study all novelties within a physical environment. Consequently, in this study, we focus on a limited set of intriguing novelties selected from this vast expanse of possibilities.

In alignment with our focus on physical reasoning, our examination is restricted to physics-based novelties, particularly those that apply forces to objects. This choice stems from their realistic nature and frequent occurrence in physical environments. For instance, within the NovPhy testbed, among the eight introduced novelties, six entail the application of forces. These include a Fan that applies horizontal forces to objects, a Magnet with attraction and repulsion forces, an Air Turbulence that exerts vertical forces on objects, and a novelty that alters the gravitational force. In abstract form, we consider four force-applying novelties: RightForce, LeftForce, UpForce, and DownForce, which apply forces in the respective directions of Right, Left, Up, and Down. In the context of a task, these novelties can be positioned within the task space, applying force in a specified region towards the specified direction. To observe how these novelties are operationalized in the tasks, refer to Figure 5, which illustrates some tasks generated using the proposed methodology.

[1] https://github.com/ChathuraT/science_birds_novelty_generator

IV. Theory for Designing Tasks

In this section, we formulate the theory that we use to design tasks in a physical environment, enabling us to confidently evaluate the novelty adaptation capabilities of an AI agent. In order to facilitate the discussion we define the below terms first:

- A **physical interaction** is a dynamic engagement involving two or more objects within a physical environment. It is characterized by tangible effects such as collisions, applied forces, or any other physical engagement influencing the behaviour of the involved objects.
- The **causality between interactions** represents the cause-and-effect relationship within the dynamic engagements among objects. It elucidates how the influence of one interaction (cause) gives rise to subsequent interactions (effect) within the physical environment.
- A **physical scenario** is a structured representation of a physics-based environment, defined by a specific problem or objective. It is characterized by the potential physical interactions among objects within the environment that are arranged in a sequence based on their causality, which can lead to the solving of physical tasks derived from the scenario.
- A **physical task** is a specific configuration of objects within the physical environment, derived from a corresponding physical scenario. It is solvable by adhering to the sequence of causal physical interactions defined in its scenario, leading to the solving of the scenario's specified problem or objective.

We refer to tasks without novelties as normal tasks and tasks with novelties as novel tasks. In our study, we focus on scenarios where only a single novelty is present in a novel task. This aligns with the existing evaluation criteria in the state-of-the-art OWL evaluation protocol [5]. To confidently assess an agent's adaptation to novelties in a physical environment, we propose the following criteria:

1) The agent is first presented with a normal task, followed by a novel task. This setup closely replicates real-world situations where novelty unexpectedly emerges in a familiar environment. This is also the standard evaluation setting for agents in OWL research [5], which has also been adapted in testbeds such as NovPhy [3].
2) The novel task should be created solely by introducing a novelty to the normal task. This ensures that any change in the task solution is purely a result of the effect of the introduced novelty.
3) There must be a distinct change in the task solution when transitioning from the normal task to the novel task. This requires the agent to explicitly respond to the influence of the introduced novelty, rather than simply ignoring it, allowing for an accurate assessment of the agent's adaptation capabilities based on task performance results.

Based on the above criteria, we deduce the following task design requirements:

- Each physical scenario comprises a pair of tasks, a normal task, and a novel task.
- For a physical scenario, two solutions are defined: one for the normal task (S_{nor}) and one for the novel task (S_{nov}).
- The normal task and the novel task are identical, except that the novel task includes a novelty.
- S_{nor} and S_{nov} must satisfy the following conditions:
  - S_{nor} and S_{nov} are not identical.
  - S_{nor} works without the application of novelty but becomes ineffective when novelty is applied.
  - S_{nov} is ineffective without the application of novelty but works when novelty is applied.

We formulate the above requirements as our theory for designing tasks for novelty adaptation evaluation in physics-based environments as follows:

We define the solution of a task as a causal sequence of physical interactions between the involved objects that lead to task completion. A solution interaction sequence effectively solves the task when the interactions unfold in the defined order. The causal relationships between interactions adhere to a temporal sequence, where the occurrence of each interaction triggers the subsequent interaction, and the preceding interaction serves as the cause for the subsequent one. Let $S_{\text{nor}} = i_1, i_2, i_3, \ldots, i_n$ represent the solution interaction sequence for the normal task. Let $S_{\text{nov}} = i'_1, i'_2, i'_3, \ldots, i'_n$ represent the solution interaction sequence for the novel task. The conditions for task design are as follows:

1) $S_{\text{nor}} \neq S_{\text{nov}}$.
2) When novelty is not applied (i.e., in the normal task):
   - $\forall i \in S_{\text{nor}}$: the interaction i is functional and results in successful task completion using S_{nor}.
   - $\exists i'_m, i'_n \in S_{\text{nov}}$ such that the causality between i'_m and i'_n is disrupted, interrupting the interaction sequence and hence preventing the normal task from being solved using S_{nov}.
3) When novelty is applied (i.e., in the novel task):
   - $\forall i' \in S_{\text{nov}}$: the interaction i' is functional and results in successful task completion using S_{nov}.
   - $\exists i_n, i_m \in S_{\text{nor}}$ such that the causality between i_n and i_m is disrupted, interrupting the interaction sequence and hence preventing the novel task from being solved using S_{nor}.
4) A single novelty, applied to the novel task, should be capable of disrupting the causality between interactions i_n and i_m, while simultaneously constructing the causality between interactions i'_n and i'_m.

V. Grammar and Defining Scenarios

In this section, we present an existing grammar that can be used to define physical scenarios and propose extensions to this grammar to incorporate novelties into physical scenarios. We then demonstrate how this grammar enables defining physical scenarios with novelties, followed by defining a set of sample scenarios for this study.

A. Grammar for Angry Birds

A grammar with four essential components has been proposed in [6] to facilitate the description of objects, interactions,

restricted interactions, and object layouts within a physical environment. These grammar components are known as object grammar, interaction grammar, restriction grammar, and layout grammar, and they are detailed in Table I. The object, interaction, and restriction grammars are used in defining physical scenarios, while the layout grammar is used during the generation process to establish constraints between objects in the layout.

To effectively describe the effect of novelties, we propose the addition of two new grammar components: disruption grammar and construction grammar (Table II). In this work, we consider two distinct ways that a novelty can impact causal interactions between physical objects: by disrupting existing causal interactions and by introducing new causal interactions. These two grammar components, disruption grammar and construction grammar, are designed to describe these effects. The grammar terms are defined in accordance with the four abstract novelties considered in this study: RightForce, LeftForce, UpForce, and DownForce.

For example, a disruption term such as, *notOnRightForce([roll(obj1)(obj2)(right)])([fall(obj1)(obj3)])* represents that when there is a RightForce novelty, object1 rolling on object2 to the right does not cause object1 to fall onto object3. This indicates that the causality between the rolling interaction and the subsequent falling has been disrupted by the RightForce novelty. On the other hand, a construction term such as *onRightForce([roll(obj1)(obj2)(right)])([hit(obj1)(obj3)(left)])* represents that when a RightForce novelty is present, object1 rolling on object2 to the right causes object1 to hit object3 from the left side. This indicates that the causality between the rolling interaction and the subsequent hitting has been constructed by the RightForce novelty.

TABLE I
GRAMMAR TERMS OF THE FOUR GRAMMAR COMPONENTS PROPOSED IN [6]. THE PARAMETERS *a* AND *b* REPRESENT OBJECTS, *d* REPRESENTS A DIRECTION, AND *l* REPRESENTS A LOCATION.

Object Grammar	Game Objects Represented
bird	redBird
pig	pigSmall∨pigMedium
rollableBlock	circleSmall∨circle
fallableBlock	circleSmall∨circle∨squareHole∨triangleHole
slidableBlock	squareHole∨triangleHole
horizontalSurface	flatPlatform
inclinedSurface	inclinedPlatform
surface	flatPlatform∨inclinedPlatform
Interaction Grammar	**Description**
hit(*a*)(*b*)(*d*)	*a* collides with *b* from direction *d* of *b* *d* ∈ *D*, *D* = {*left, right, above, below, any*}
roll(*a*)(*b*)(*d*)	*a* rolls on *b* towards direction *d* *d* ∈ *D*, *D* = {*left, right*}
fall(*a*)(*b*)	*a* falls towards *b*
slide(*a*)(*b*)(*d*)	*a* slides on *b* towards direction *d* *d* ∈ *D*, *D* = {*left, right*}
bounce(*a*)(*b*)(*d*)	*a* bounces off *b* towards direction *d* *d* ∈ *D*, *D* = {*left, right, above, below*}
destroy(*a*)(*b*)	*a* destroys *b* in the collision with *b*
Restriction Grammar	**Description**
cannotHit(*a*)(*b*)(*d*)	*a* cannot collide with *b* from direction *d* of *b* *d* ∈ *D*, *D* = {*left, right, above, below, any*}
cannotFall(*a*)	*a* cannot fall in its motion
Layout Grammar	**Description**
inDirection(*a*)(*b*)(*d*)	*a* is in direction *d* of *b* *d* ∈ *D*, *D* = {*left, right, above, below*}
onLocation(*a*)(*b*)(*l*)	*a* is on top of *b* at location *l* *l* ∈ *L*, *L* = {*left, centre, right*}
locatedFar(*a*)(*b*)(*d*)	*a* is located far from *b* in direction *d* of *b* *d* ∈ *D*, *D* = {*left, right, above, below*}
touching(*a*)(*b*)(l)	*a* is touching *b* at location *l* of *b*, *l* ∈ *L* *L* = {*upperLeft, centreLeft, lowerLeft*}
pathObstructed(*a*)(*b*)(*d*)	there is an obstacle in the path between *a* and *b*, in the direction *d* to *b* *d* ∈ *D*, *D* = {*left, right, above, below, all*}
liesOnPath(*a*)(*b*)	*a* lies on *b*'s moving path

TABLE II
GRAMMAR TERMS FOR DISRUPTION AND CONSTRUCTION GRAMMARS DESCRIBING IMPACTS TO CAUSAL INTERACTIONS. THE PARAMETERS *p* AND *q* REPRESENT INTERACTIONS.

Disruption Grammar	Description
notOnRightForce(*p*)(*q*)	*p* does not cause *q* when a RightForce is present
notOnLeftForce(*p*)(*q*)	*p* does not cause *q* when a LeftForce is present
notOnUpForce(*p*)(*q*)	*p* does not cause *q* when an UpForce is present
notOnDownForce(*p*)(*q*)	*p* does not cause *q* when a DownForce is present
Construction Grammar	**Description**
onRightForce(*p*)(*q*)	*p* causes *q* when a RightForce is present
onLeftForce(*p*)(*q*)	*p* causes *q* when a LeftForce is present
onUpForce(*p*)(*q*)	*p* causes *q* when an UpForce is present
onDownForce(*p*)(*q*)	*p* causes *q* when a DownForce is present

B. Defining Scenarios

A physical scenario defined by the user using the above-discussed grammar serves as the input to the proposed generation process in this work. As detailed in the theory of task design in Section IV, each physical scenario consists of a pair of tasks: a normal task and a novel task. Thus, when defining a scenario, the user must devise two solution sequences of physical interactions: one for normal tasks (S_{nor}) and the other for novel tasks (S_{nov}). The object grammar and interaction grammar presented earlier are utilized to define these sequences of interactions between objects. Subsequently, the restriction grammar is utilized to specify any constraints relevant to the tasks, such as preventing one object from hitting another. This is primarily done to prevent the tasks from being solvable using alternative solutions rather than the intended one, an idea discussed in the literature, as seen in [33]. The newly introduced disruption and construction grammar are then used to specify the impact of introducing novelty on selected pairs of interactions. In S_{nor}, the novelty should disrupt the causality of a pair of interactions, rendering S_{nor} ineffective when the novelty is introduced. Conversely, in S_{nov}, the novelty should construct the causality of a pair of interactions, enabling S_{nov}, to become effective with the introduction of the novelty. Table III illustrates 12 example physical scenarios defined for demonstration purposes in this study. The step-by-step process for defining a scenario is outlined below using Scenario 1 from Table III as an example:

1) Begin by conceptualizing a scenario and identifying the objects required, defining them using the object grammar.
   For example, in Scenario 1, we consider a scenario where a bird must hit a rollable object placed on a

surface, causing it to roll and subsequently hit and destroy a pig. In this case, the objects required would include a *bird*, a *rollableBlock*, a *surface*, and a *pig*.

2) Establish two potential solutions for the normal tasks (S_{nor}) and novel tasks (S_{nov}). These solutions are represented as sequences of physical interactions between the objects leading to the destruction of the pigs in the task. Utilize the interaction grammar to define these sequences, ordering them according to the causality of the interactions.
   In our example, S_{nor} involves a bird hitting a rollable block from the block's left side, causing it to roll to the right on an inclined surface. Subsequently, the rollable block falls towards the pig, striking the pig from above and leading to the pig's destruction. In S_{nov}, the bird similarly hits the rollable block from the block's left side, resulting in it rolling to the right on a horizontal surface and hitting the pig from the left side, thereby causing the pig's destruction. For the interaction sequences written using the grammar for this example, please refer to Scenario 1 in Table III.
3) Specify any interactions between objects that need to be restricted using the restriction grammar.
   In the example scenario, we aim to prevent the bird from directly hitting the pig from any direction, thereby ensuring that the task cannot be solved by directly launching the bird at the pig instead of using the rollable object to destroy the pig. To achieve this, we include the restriction [cannotHit(*bird*)(*pig*)(*any*)].
4) Incorporate the effect of the novelty into the definitions. Firstly, select a novelty intended to be introduced into the scenario. Then, for S_{nor}, identify two potential interactions whose causality should be disrupted by the novelty and define them using the disruption grammar. Conversely, for S_{nov}, identify two potential interactions whose causality should be constructed by the same novelty and define them using the construction grammar. In the example, for S_{nor}, it was determined that the causality between the interactions of the rollable block rolling and its falling can be disrupted when the RightForce novelty is introduced. Therefore, the disruption term [notOnRightForce([roll(..)(..)(..)])([fall(..)(..)])] is added. Conversely, for S_{nov}, the causality between the interactions of the rollable block rolling and hitting the pig potentially can be constructed only when a RightForce novelty is applied. Thus, the construction term [onRightForce([roll(..)(..)(..)])([hit(..)(..)(..)])] is included.

As mentioned earlier, Table III showcases 12 example physical scenarios defined for demonstration purposes. Scenarios five through eight demonstrate four scenarios, each featuring one of the four novelties applied to the same solution sequences. Scenarios nine through twelve demonstrate the application of novelties to disrupt and construct causalities of different interactions within the same solution sequences. Please refer to Figure 5 for the generated tasks corresponding to these scenario definitions, providing a visualization of these scenario definitions.

VI. Task Generation Process

The task generation process utilizes a defined scenario as input and generates feasible tasks (i.e., game levels) that can be solved using the interaction sequences defined in the scenario. This process comprises two phases: a qualitative phase followed by a simulation phase. During the qualitative phase, the unbounded generative space is narrowed down to a manageable limited space. Subsequently, guided by the results of the qualitative phase, simulation-based techniques are employed to precisely determine the layouts of objects. In this section, we discuss these two phases.

A. Qualitative Phase

In the initial phase of the generation process, we utilize the qualitative techniques discussed in [6]. Here, we provide a brief overview of these qualitative techniques. For a more comprehensive understanding, please refer to [6].

The process commences with a scenario definition as the input, defined as the normal and novel solution sequences. From the scenario definition, the necessary objects to be included in the scenario are determined. Subsequently, the layout constraints are inferred between these objects. The layout constraints are defined using the layout grammar presented in Table I, taking into account the interactions and restricted interactions specified in the scenario definition. The layout terms that can be inferred from the interaction and restriction predicates are provided in Table IV. When inferring layout constraints, redundant constraints between objects can be inferred depending on the interaction and restriction predicates present in the scenario definition. Therefore, such redundant constraints are eliminated from the inferred layout constraints. Additionally, the physical stability of the objects is tested by checking that all dynamic objects are in a stable configuration, with direct support from static objects underneath. If physically unsupported objects exist, new surfaces are introduced below them to establish stability. This ensures that all dynamic objects are initially placed in a stable configuration. Next, these inferred layout constraints are represented as a layout constraint graph among the objects. The layout constraint graph for Scenario 1 in Table III is illustrated in Figure 2.

The next step involves converting the generated layout constraint graph into a Qualitative Spatial Relationship (QSR) graph. In this process, objects are represented using a 5-point representation, utilizing the X and Y coordinates of their lower left (ll), upper left (ul), centre (c), lower right (lr), and upper right (ur) points. The layout constraint terms are then mapped into different QSRs derived from various QSR calculi in the 2D Euclidean space. This mapping is detailed in Table V. We exclude the layout terms pathObstructed and liesOnPath, as they are handled in the simulation phase of the generation. The resulting QSR relationships are represented as point relationships between the objects using the 5-point representation. Notably, a single layout term can be mapped into multiple QSRs, thereby allowing diversity in the generator's output.

TABLE III
DEFINITIONS OF THE EXAMPLE PHYSICAL SCENARIOS. A SCENARIO IS DEFINED USING TWO SOLUTION SEQUENCES OF PHYSICAL INTERACTIONS, ONE FOR THE NORMAL TASK (S_{NOR}) AND THE OTHER FOR THE NOVEL TASK (S_{NOV}) (SHOWN IN THE FIRST LINE AND SECOND LINE, RESPECTIVELY). IN THE DEFINITION, A SEQUENCE OF PHYSICAL INTERACTIONS (CAUSALITY DENOTED BY $>$, AND $\mathbin{>}$ REPRESENTS THE CAUSALITY THAT GETS AFFECTED BY THE NOVELTY) IS FOLLOWED BY A SET OF RESTRICTIONS, AND SUBSEQUENTLY, EITHER A DISRUPTION (FOR THE NORMAL TASK) OR A CONSTRUCTION (FOR THE NOVEL TASK) TERM. THE OBJECT GRAMMAR TERMS ROLLABLEBLOCK, FALLABLEBLOCK, SLIDABLEBLOCK, HORIZONTALSURFACE, AND INCLINEDSURFACE ARE ABBREVIATED AS RBLOCK, FBLOCK, SBLOCK, HSURFACE, AND ISURFACE, RESPECTIVELY. FOR OVERLOADED PARAMETER VALUES, ANY OF THE OVERLOADED VALUES CAN BE USED (E.G., HIT(*bird*)(*fBlock*)(*left*∨*above*) REPRESENTS THE BIRD CAN COLLIDE WITH THE *fBlock* FROM *left* OR FROM *above*).

Scenario	Scenario Definition
1	{[hit(*bird*)(*rBlock1*)(*left*)] > [roll(*rBlock1*)(*iSurface*)(*right*)] $\mathbin{>}$ [fall(*rBlock1*)(*pig*)] > [hit(*rBlock1*)(*pig*)(*above*)] > [destroy(*rBlock1*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]}, {[notOnRightForce(*[roll(rBlock1)(iSurface)(right)]*)(*[fall(rBlock1)(pig)]*)]}
	{[hit(*bird*)(*rBlock2*)(*left*)] > [roll(*rBlock2*)(*hSurface*)(*right*)] $\mathbin{>}$ [hit(*rBlock2*)(*pig*)(*left*)] > [destroy(*rBlock2*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]∧[cannotFall(*rBlock2*)]}, {[onRightForce(*[roll(rBlock2)(hSurface)(right)]*)(*[hit(rBlock2)(pig)(left)]*)]}
2	{[hit(*bird*)(*rBlock1*)(*left*)] > [roll(*rBlock1*)(*hSurface*)(*right*)] $\mathbin{>}$ [hit(*rBlock1*)(*pig*)(*left*)] > [destroy(*rBlock1*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]∧[cannotFall(*rBlock1*)]}, {[notOnDownForce(*[roll(rBlock1)(hSurface)(right)]*)(*[hit(rBlock1)(pig)(left)]*)]}
	{[hit(*bird*)(*rBlock2*)(*left*)] > [roll(*rBlock2*)(*iSurface*)(*right*)] $\mathbin{>}$ [fall(*rBlock2*)(*pig*)] > [hit(*rBlock2*)(*pig*)(*above*)] > [destroy(*rBlock2*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]}, {[onDownForce(*[roll(rBlock2)(iSurface)(right)]*)(*[fall(rBlock2)(pig)]*)]}
3	{[hit(*bird*)(*sBlock1*)(*left*)] > [slide(*sBlock1*)(*iSurface*)(*right*)] $\mathbin{>}$ [fall(*sBlock1*)(*pig*)] > [hit(*sBlock1*)(*pig*)(*above*)] > [destroy(*sBlock1*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]}, {[notOnRightForce(*[slide(sBlock1)(iSurface)(right)]*)(*[fall(sBlock1)(pig)]*)]}
	{[hit(*bird*)(*sBlock2*)(*left*)] > [slide(*sBlock2*)(*hSurface*)(*right*)] $\mathbin{>}$ [hit(*sBlock2*)(*pig*)(*left*)] > [destroy(*sBlock2*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]∧[cannotFall(*sBlock2*)]}, {[onRightForce(*[slide(sBlock2)(hSurface)(right)]*)(*[hit(sBlock2)(pig)(left)]*)]}
4	{[hit(*bird*)(*sBlock1*)(*left*)] > [slide(*sBlock1*)(*hSurface*)(*right*)] $\mathbin{>}$ [hit(*sBlock1*)(*pig*)(*left*)] > [destroy(*sBlock1*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]∧[cannotFall(*sBlock1*)]}, {[notOnDownForce(*[slide(sBlock1)(hSurface)(right)]*)(*[hit(sBlock1)(pig)(left)]*)]}
	{[hit(*bird*)(*sBlock2*)(*left*)] > [slide(*sBlock2*)(*iSurface*)(*right*)] $\mathbin{>}$ [fall(*sBlock2*)(*pig*)] > [hit(*sBlock2*)(*pig*)(*above*)] > [destroy(*sBlock2*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]}, {[onDownForce(*[slide(sBlock2)(iSurface)(right)]*)(*[fall(sBlock2)(pig)]*)]}
5	{[hit(*bird*)(*fBlock1*)(*left*∨*above*)] > [fall(*fBlock1*)(*pig*)] $\mathbin{>}$ [hit(*fBlock1*)(*pig*)(*above*)] > [destroy(*fBlock1*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]}, {[notOnRightForce(*[fall(fBlock1)(pig)]*)(*[hit(fBlock)(pig)(above)]*)]}
	{[hit(*bird*)(*fBlock2*)(*left*∨*above*)] > [fall(*fBlock2*)(*pig*)] $\mathbin{>}$ [hit(*fBlock2*)(*pig*)(*above*)] > [destroy(*fBlock2*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]} {[onRightForce(*[fall(fBlock2)(pig)]*)(*[hit(fBlock2)(pig)(above)]*)]}
6	{[hit(*bird*)(*fBlock1*)(*left*∨*above*)] > [fall(*fBlock1*)(*pig*)] $\mathbin{>}$ [hit(*fBlock1*)(*pig*)(*above*)] > [destroy(*fBlock1*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]}, {[notOnDownForce(*[fall(fBlock1)(pig)]*)(*[hit(fBlock)(pig)(above)]*)]}
	{[hit(*bird*)(*fBlock2*)(*left*∨*above*)] > [fall(*fBlock2*)(*pig*)] $\mathbin{>}$ [hit(*fBlock2*)(*pig*)(*above*)] > [destroy(*fBlock2*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]} {[onDownForce(*[fall(fBlock2)(pig)]*)(*[hit(fBlock2)(pig)(above)]*)]}
7	{[hit(*bird*)(*fBlock1*)(*left*∨*above*)] > [fall(*fBlock1*)(*pig*)] $\mathbin{>}$ [hit(*fBlock1*)(*pig*)(*above*)] > [destroy(*fBlock1*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]}, {[notOnUpForce(*[fall(fBlock1)(pig)]*)(*[hit(fBlock)(pig)(above)]*)]}
	{[hit(*bird*)(*fBlock2*)(*left*∨*above*)] > [fall(*fBlock2*)(*pig*)] $\mathbin{>}$ [hit(*fBlock2*)(*pig*)(*above*)] > [destroy(*fBlock2*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]} {[onUpForce(*[fall(fBlock2)(pig)]*)(*[hit(fBlock2)(pig)(above)]*)]}
8	{[hit(*bird*)(*fBlock1*)(*left*∨*above*)] > [fall(*fBlock1*)(*pig*)] $\mathbin{>}$ [hit(*fBlock1*)(*pig*)(*above*)] > [destroy(*fBlock1*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]}, {[notOnLeftForce(*[fall(fBlock1)(pig)]*)(*[hit(fBlock)(pig)(above)]*)]}
	{[hit(*bird*)(*fBlock2*)(*left*∨*above*)] > [fall(*fBlock2*)(*pig*)] $\mathbin{>}$ [hit(*fBlock2*)(*pig*)(*above*)] > [destroy(*fBlock2*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]} {[onLeftForce(*[fall(fBlock2)(pig)]*)(*[hit(fBlock2)(pig)(above)]*)]}
9	{[hit(*bird*)(*rBlock1*)(*left*)] > [roll(*rBlock1*)(*surface1*)(*right*)] $\mathbin{>}$ [hit(*rBlock1*)(*fBlock1*)(*left*)] > [fall(*fBlock1*)(*pig*)] > [hit(*fBlock1*)(*pig*)(*above*)] > [destroy(*fBlock1*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]∧[cannotFall(*rBlock1*)]}, {[notOnRightForce(*[roll(rBlock1)(surface1)(right)]*)(*[hit(rBlock1)(fBlock1)(left)]*)]}
	{[hit(*bird*)(*rBlock2*)(*left*)] > [roll(*rBlock2*)(*surface2*)(*right*)] $\mathbin{>}$ [hit(*rBlock2*)(*fBlock2*)(*left*)] > [fall(*fBlock2*)(*pig*)] > [hit(*fBlock2*)(*pig*)(*above*)] > [destroy(*fBlock2*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]∧[cannotFall(*rBlock2*)]}, {[onRightForce(*[roll(rBlock2)(surface2)(right)]*)(*[hit(rBlock2)(fBlock2)(left)]*)]}
10	{[hit(*bird*)(*rBlock1*)(*left*)] > [roll(*rBlock1*)(*surface1*)(*right*)] $\mathbin{>}$ [hit(*rBlock1*)(*fBlock1*)(*left*)] > [fall(*fBlock1*)(*pig*)] > [hit(*fBlock1*)(*pig*)(*above*)] > [destroy(*fBlock1*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]∧[cannotFall(*rBlock1*)]}, {[notOnDowntForce(*[roll(rBlock1)(surface1)(right)]*)(*[hit(rBlock1)(fBlock1)(left)]*)]}
	{[hit(*bird*)(*rBlock2*)(*left*)] > [roll(*rBlock2*)(*surface2*)(*right*)] $\mathbin{>}$ [hit(*rBlock2*)(*fBlock2*)(*left*)] > [fall(*fBlock2*)(*pig*)] > [hit(*fBlock2*)(*pig*)(*above*)] > [destroy(*fBlock2*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]∧[cannotFall(*rBlock2*)]}, {[onDownForce(*[roll(rBlock2)(surface2)(right)]*)(*[hit(rBlock2)(fBlock2)(left)]*)]}
11	{[hit(*bird*)(*rBlock1*)(*left*)] > [roll(*rBlock1*)(*surface1*)(*right*)] > [hit(*rBlock1*)(*fBlock1*)(*left*)] > [fall(*fBlock1*)(*pig*)] $\mathbin{>}$ [hit(*fBlock1*)(*pig*)(*above*)] > [destroy(*fBlock1*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]∧[cannotFall(*rBlock1*)]}, {[notOnLeftForce(*[fall(fBlock1)(pig)]*)(*[hit(fBlock1)(pig)(above)]*)]}
	{[hit(*bird*)(*rBlock2*)(*left*)] > [roll(*rBlock2*)(*surface2*)(*right*)] > [hit(*rBlock2*)(*fBlock2*)(*left*)] > [fall(*fBlock2*)(*pig*)] $\mathbin{>}$ [hit(*fBlock2*)(*pig*)(*above*)] > [destroy(*fBlock2*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]∧[cannotFall(*rBlock2*)]}, {[onLeftForce(*[fall(fBlock2)(pig)]*)(*[hit(fBlock2)(pig)(above)]*)]}
12	{[hit(*bird*)(*rBlock1*)(*left*)] > [roll(*rBlock1*)(*surface1*)(*right*)] > [hit(*rBlock1*)(*fBlock1*)(*left*)] > [fall(*fBlock1*)(*pig*)] $\mathbin{>}$ [hit(*fBlock1*)(*pig*)(*above*)] > [destroy(*fBlock1*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]∧[cannotFall(*rBlock1*)]}, {[notOnUpForce(*[fall(fBlock1)(pig)]*)(*[hit(fBlock1)(pig)(above)]*)]}
	{[hit(*bird*)(*rBlock2*)(*left*)] > [roll(*rBlock2*)(*surface2*)(*right*)] > [hit(*rBlock2*)(*fBlock2*)(*left*)] > [fall(*fBlock2*)(*pig*)] $\mathbin{>}$ [hit(*fBlock2*)(*pig*)(*above*)] > [destroy(*fBlock2*)(*pig*)]}, {[cannotHit(*bird*)(*pig*)(*any*)]∧[cannotFall(*rBlock2*)]}, {[onUpForce(*[fall(fBlock2)(pig)]*)(*[hit(fBlock2)(pig)(above)]*)]}

Additionally, the generator can overload each grammar term when defining scenarios, as exemplified in scenario six where the bird can hit the fBlock from either the left side or above. This capacity for choice enhances the generator's ability to produce a wide range of tasks. The QSR graph corresponding to scenario 1, derived from its layout constraint graph, is illustrated in Figure 3.

In the concluding step of the qualitative phase, the constructed QSR graph is examined to ensure its consistency,

TABLE IV
THE LAYOUT CONSTRAINTS THAT CAN BE INFERRED FROM THE INTERACTIONS AND RESTRICTIONS PROPOSED IN [6]. THE PARAMETER *-d* REPRESENTS THE OPPOSITE DIRECTION OF *d*. IN THE TERM CANNOTFALL(*a*), *p*, *q*, AND SO ON ARE USED TO DENOTE THE OBJECTS OVER WHICH *a* MOVES IN THE SPECIFIED ORDER, WHILE *lp*, *lq*, AND SO ON INDICATE THE LOCATIONS WHERE THOSE OBJECTS ARE CONNECTED, CREATING A CONTINUOUS PATH FOR *a* TO FOLLOW.

Interactions/Restrictions Predicate	Inferred Layout Constraints
hit(*a*)(*b*)(*d*)	liesOnPath(*b*)(*a*)∧inDirection(*b*)(*a*)(*-d*)
roll(*a*)(*b*)(*d*)	inDirection(*a*)(*b*)(*-d*)
fall(*a*)(*b*)(*d*)	locatedFar(*a*)(*b*)(*d*)
slide(*a*)(*b*)(*d*)	inDirection(*a*)(*b*)(*-d*)
bounce(*a*)(*b*)(*d*)	inDirection(*a*)(*b*)(*d*)
cannotHit(*a*)(*b*)(*d*)	pathObstructed(*a*)(*b*)(*d*)
cannotFall(*a*)	touching(*a*)(*p*)(*lp*)∧touching(*p*)(*q*)(*lq*), ...

TABLE V
THE QUALITATIVE SPATIAL RELATIONSHIPS INFERRED FROM THE LAYOUT CONSTRAINTS PROPOSED IN [6]. THE ABBREVIATED REPRESENTATION OF THE DIRECTIONS IN THE NOTATION FOLLOWS CONVENTIONAL CONVENTIONS.

Layout Predicate	Inferred QSRs
inDirection(*a*)(*b*)(*left*)	W(*a*,*b*)∨NW(*a*,*b*)∨SW(*a*,*b*)
inDirection(*a*)(*b*)(*right*)	E(*a*,*b*)∨NE(*a*,*b*)∨SE(*a*,*b*)
inDirection(*a*)(*b*)(*above*)	N(*a*,*b*)∨NE(*a*,*b*)∨NW(*a*,*b*)
inDirection(*a*)(*b*)(*below*)	S(*a*,*b*)∨SE(*a*,*b*)∨SW(*a*,*b*)
onLocation(*a*)(*b*)(*left*)	MeetDuringW(*a*,*b*)
onLocation(*a*)(*b*)(*centre*)	MeetN(*a*,*b*)
onLocation(*a*)(*b*)(*right*)	MeetDuringE(*a*,*b*)
locatedFar(*a*)(*b*)(*left*)	FarW(*a*,*b*)∨FarNW(*a*,*b*)∨FarSW(*a*,*b*)
locatedFar(*a*)(*b*)(*right*)	FarE(*a*,*b*)∨FarNE(*a*,*b*)∨FarSE(*a*,*b*)
locatedFar(*a*)(*b*)(*above*)	FarN(*a*,*b*)∨FarNE(*a*,*b*)∨FarNW(*a*,*b*)
locatedFar(*a*)(*b*)(*below*)	FarS(*a*,*b*)∨FarSE(*a*,*b*)∨FarSW(*a*,*b*)
touching(*a*)(*b*)(*upperLeft*)	MeetNW(*a*,*b*)
touching(*a*)(*b*)(*centreLeft*)	MeetW(*a*,*b*)
touching(*a*)(*b*)(*lowerLeft*)	MeetSW(*a*,*b*)

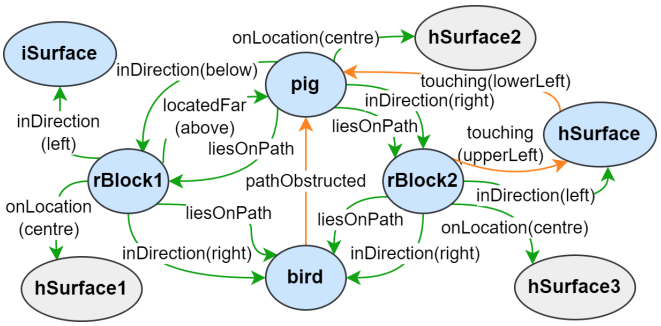

Fig. 2. The layout constraint graph of scenario one. Blue nodes represent objects defined in the scenario definition, and grey nodes represent objects introduced to maintain object stability under gravity. Green edges represent layout constraints inferred from interactions, while orange edges represent constraints inferred from restrictions defined in the scenario.

thereby verifying the existence of feasible spatial layouts for all objects. To accomplish this, a dimension graph-based approach is employed [34], wherein point-based constraints are projected independently into the X and Y dimensions, producing a pair of dimension graphs - one for the X dimension and another for the Y dimension. This approach effectively transforms the task of consistency checking into a graph cycle detection problem. If both the X and Y dimension graphs, based on a given set of spatial constraints, are free of cycles,

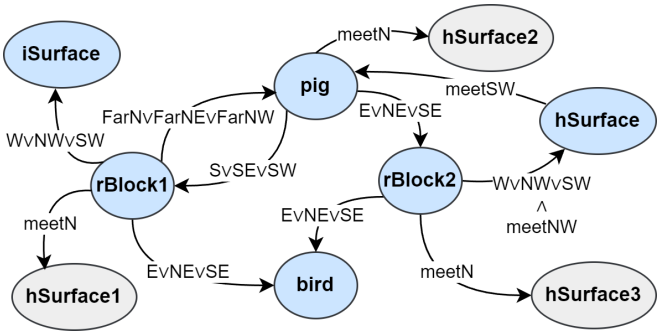

Fig. 3. The qualitative spatial relationship (QSR) graph of scenario one. Nodes represent objects, and edges represent QSR relationships. A directed edge from node *a* to *b* with a relationship *y* denotes that '*a* is in *y* of *b*'. For connections with multiple relationships (represented with ∨), any relationship out of them can be considered.

it signifies that generating a feasible layout for the objects that satisfies the constraints is possible. As mentioned earlier, the presence of choices in selecting QSR relations between objects allows for the generation of a pool of feasible dimension graph pairs for a given scenario, each representing a plausible layout that satisfies the specified constraints.

B. Simulation Phase

In physics-based environments, using qualitative methods alone may not fully capture the complexities imposed by physics, such as object motion and destruction. Therefore, simulation-based techniques have frequently been employed by researchers for generating content in such environments [16]. In the second phase of our task generation process, simulation-based techniques are used to obtain additional information, particularly to determine object trajectory paths (to satisfy the liesOnPath terms) and to verify task solvability with different actions. The steps of this phase are illustrated in Figure 4. Starting with a pool of dimension graphs obtained from the qualitative phase, the final outcome is a feasible pair of normal and novel tasks for the input scenario that adhere to our task design theory.

Initially, a pair of dimension graphs (representing X and Y dimensions) is randomly selected from the pool of dimension graph pairs. Subsequently, the point-based constraints represented by these dimension graphs are resolved using the standard forward-checking technique [35] employed in solving constraint satisfaction problems, resulting in the initial positions of the game objects. Finally, the game objects are positioned in the level space according to these determined positions.

The subsequent step involves configuring and instantiating the novelty. The specific novelty to be used is determined by referencing the disruption/construction terms in the scenario definition, which are directly associated with the intended novelty. The novelty is then adjusted in size and positioned appropriately within the level space, ensuring it lies between the two interactions to be disrupted in S_{nor} and the two interactions to be constructed in S_{nov}. If the current layout does not permit such placement of the novelty, the objects associated with one solution are shifted until a layout accommodating the desired

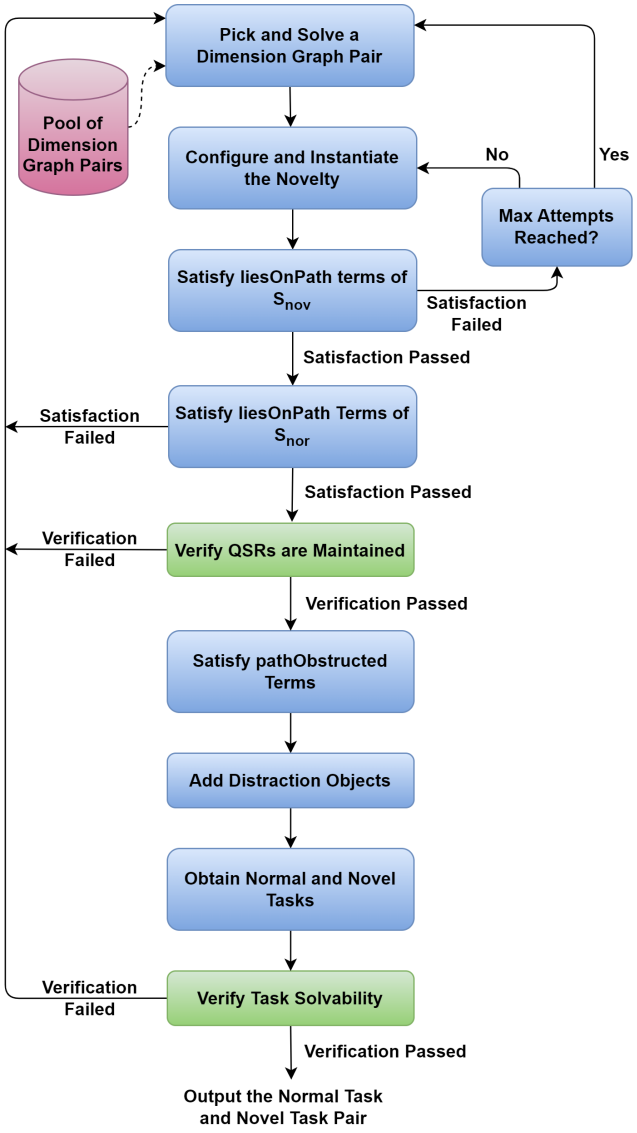

Fig. 4. Simulation phase steps of the generation process.

novelty placement is achieved. In this study, force-applying novelties are considered, and a default force magnitude is assigned to them. Then the novelty game object is instantiated in the game space with the determined size, position, and default force magnitude.

Following that, the liesOnPath terms pertaining to the S_{nov} interactions are addressed. This involves taking the action that triggers the S_{nov} interaction sequence, where the bird is shot at the object initiating the sequence. The trajectories of the objects associated with the liesOnPath terms are then observed. As Angry Birds offers a continuous range of shooting angles, we discretize the possible action space by considering only specific points of interest on the target object (e.g., ll, ur, and c) and simulating the shooting towards these points. To satisfy liesOnPath(a)(b), we adjust the position of the object a to a position that mostly intersects the trajectory of b for the tested shooting points. If satisfaction for liesOnPath cannot be achieved in this manner, particularly for the liesOnPath terms associated with objects influenced by the novelty, we systematically retry after reconfiguring the novelty (size, placement and force magnitude).

Next, the liesOnPath terms related to the S_{nor} interactions are satisfied. First, the introduced novelty is removed from the task. The process is similar to the one used for satisfying the liesOnPath terms associated with the S_{nov} interactions, involving simulating the action that initiates the S_{nov} interaction sequence, followed by observing and repositioning the relevant objects. After this step, we verify that the object layout still adheres to the spatial constraints established during the qualitative phase, guaranteeing that the layout facilitates the intended initial interactions. In the event that the liesOnPath constraints are not satisfied, or if QSR constraints are violated in the current layout, the simulation phase is reset by choosing a different pair of dimension graphs.

The next step involves satisfying the pathsObstructed terms by introducing platforms that obstruct the paths to the corresponding objects. Careful consideration is given to ensure that these obstacles do not interfere with the trajectories of other objects, which could disrupt the solution interactions of the task. Additionally, drawing inspiration from tasks used for evaluating AI agent performance in benchmarks [3], [7], we introduce a random number of superfluous objects at various locations to serve as distractions for the AI agents. We ensure that these added distractions do not alter the solution interaction sequence of the tasks. These distractions are intended to reduce the likelihood of agents exploiting spurious patterns instead of engaging in genuine physical reasoning and novelty adaptation.

Subsequently, normal and novel tasks are obtained. The task without the novelty, which is the current task at hand in the generation process, is the normal task. The novel task is obtained by incorporating the novelty into the normal task according to the previously determined configurations of the novelty. A final verification is then performed to ensure that these tasks are solvable using their intended solutions (i.e., the normal task by S_{nor} and the novel task by S_{nov}) while being unsolvable using their counterpart solutions. This verification entails simulating the bird's shooting action towards the intended target object, which is expected to instantiate the corresponding interaction sequence. The successfully verified tasks are then produced as the normal and novel task pairs of the input physical scenario.

VII. Results and Evaluations

In this section, we present the results of our proposed method and the evaluations conducted to measure its performance and adherence to our task design theory. Example generated tasks for the 12 physical scenarios considered in this work are shown in Figure 5, while Figure 6 demonstrates the variations of the tasks generated for a specific scenario, scenario 1. The limited variations observed in the tasks generated for the same scenario are in line with the task variations typically used in existing literature for assessing 'local generalization' in AI agents, such as in PHYRE [23] and Phy-Q [7]. While variations in the tasks in those works are often achieved by simple shifts in object locations, our

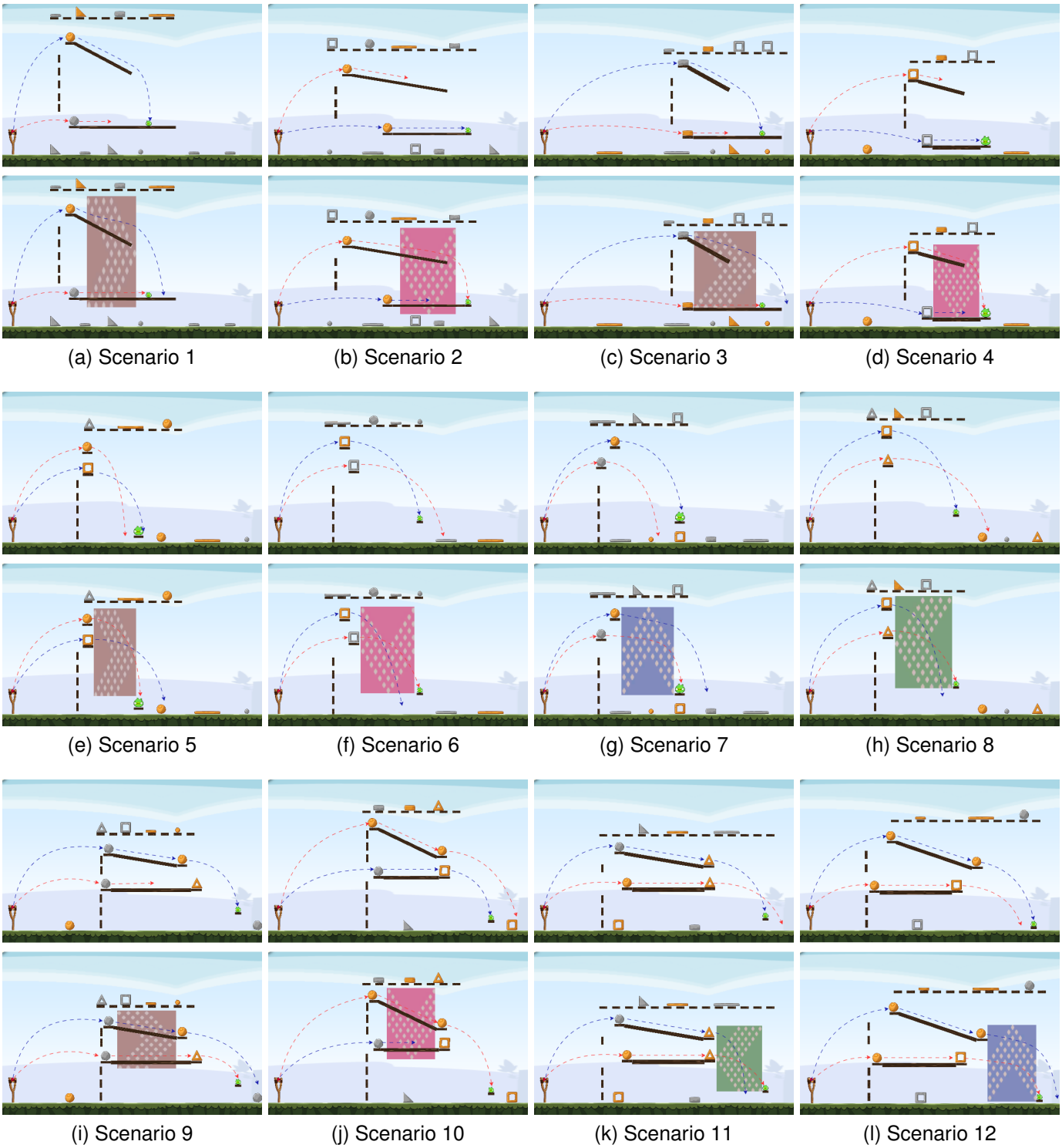

Fig. 5. Generated tasks for the defined example scenarios. Each subfigure includes the normal task (top) and the novel task (bottom). The RightForce, LeftForce, UpForce, and DownForce novelties are represented as brown, green, blue, and pink rectangles in the novel tasks, and their corresponding force is applied when an object is inside the novelty. Blue and red dotted arrows illustrate the object trajectories when S_{nor} and S_{nov} solutions are initiated, respectively.

approach expands this by incorporating changes in relative object positions, object materials, object sizes, and object types, providing a more diverse evaluation of AI agents' local generalization capabilities. Additionally, to promote 'broad generalization', our generation facilitates setups such as training agents on various scenarios with different interactions and subsequently evaluating them on different scenarios that exclusively feature interactions encountered during training, ensuring comprehensive learning and testing opportunities for the agents.

To evaluate the presented task generation methodology, we propose six metrics: generation time, physical stability of the

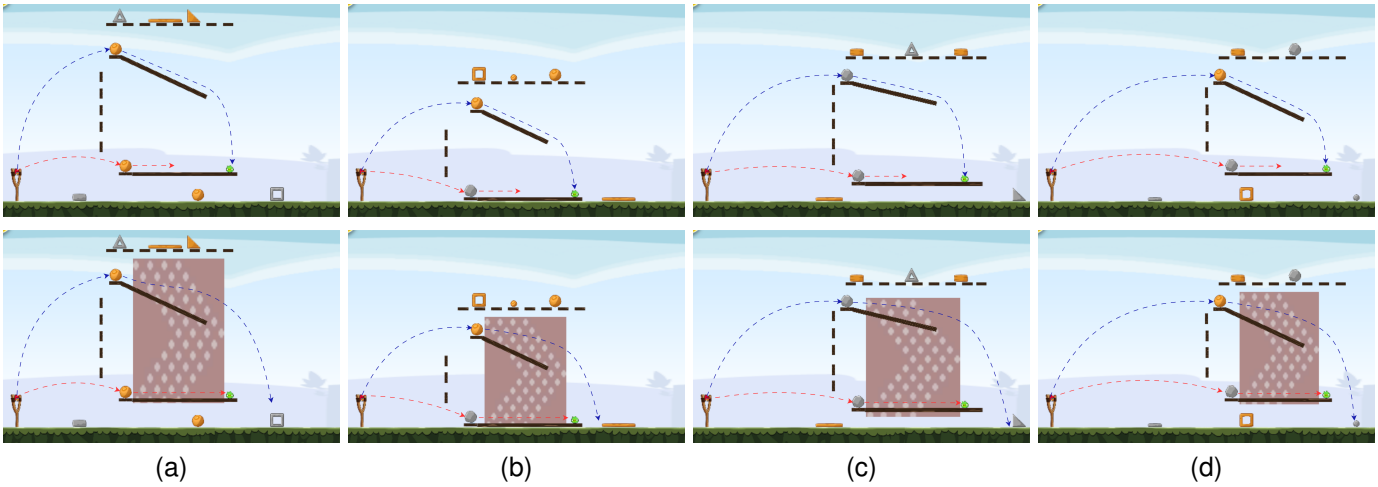

Fig. 6. Four tasks generated for scenario one. Each subfigure includes the normal task (top) and the novel task (bottom). Blue and red dotted arrows illustrate the object trajectories when S_{nor} and S_{nov} solutions are initiated, respectively.

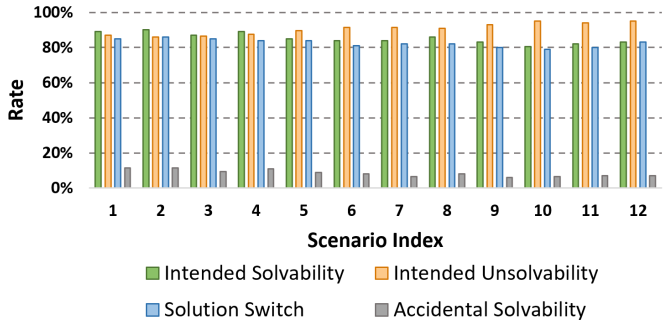

Fig. 7. The results of the intended solvability, intended unsolvability, solution switch, and accidental solvability evaluations for the 12 scenarios.

tasks, solvability of the tasks using the intended solutions, intended unsolvability of the tasks using other solutions, the satisfaction of solution switch from normal task to novel task, and accidental solvability of the tasks using unintended solutions. To evaluate the method, we generated 30 task pairs (normal and novel) for each of the 12 scenarios, totalling 360 task pairs.

A. Generation Time

To compare the efficiency of our task generator, we evaluated its runtime on a Windows 10 desktop equipped with an i9-9900KS CPU and 64GB RAM. The average time taken to generate a task pair for the 12 scenarios considered in this study was 12.92 seconds. In comparison, the physics-based task generation method presented in [6] consumed an average of 3.01 seconds to generate a task using the same infrastructure. It is important to note that the method in [6] only generates normal tasks without novelties, and each task contains a single sequence of interactions. In contrast, our approach considers two sequences of interactions, generates a feasible novelty in the process, and the measured time is for producing a pair of tasks. Furthermore, the domain-independent novelty generation method proposed in [31] requires approximately four hours to generate a feasible novelty. This result highlights the efficiency of our approach in producing tasks with novelties while maintaining a reasonable runtime.

B. Physical Stability

In the context of Angry Birds, as well as other prevalent physics-based testbeds such as [7], [22], [23], it is essential for all objects within a task to be stable under gravity at the outset. In this evaluation, we examined the physical stability of the generated tasks. Our generation process adheres to strict stability checks, ensuring that all objects are validated and stabilized under gravity during the generation of the layout constraint graph. Additionally, when introducing distraction objects, they are placed only in locations that can support their stability. Consequently, all tasks generated through our approach exhibited stable configurations under gravity at the beginning of each task.

C. Intended Solvability

To assess the intended solvability of the generated tasks, we evaluate their solvability using the defined sequence of physical interactions in the scenario definition. At the generation time, we record the action, which is the release angle of the bird, that initiates the solution interactions for each task. Subsequently, this action is executed during this evaluation process to verify the solvability of the tasks. A task pair is deemed solvable if the normal task can be solved using S_{nor}, and the novel task can be solved using S_{nov}. To obtain the intended solvability rate for a specific scenario, we calculate the percentage of solvable task pairs within that scenario. It is worth noting that we omit the final solvability verification step in the generation process for the tasks generated for this evaluation, as this step prevents the output of unsolvable tasks. Our main objective here is to assess how frequently the process can produce solvable tasks. The results, presented in Figure 7, illustrate that the intended solvability rate exceeds 81% for all the 12 example scenarios.

D. Intended Unsolvability

To gauge the intended unsolvability of the generated tasks, in contrast to the intended solvability measure, we examine the tasks' solvability using their counterpart solution. In other words, a task pair is considered to meet the intended unsolvability criterion if the normal task cannot be solved using S_{nov}, and the novel task cannot be solved using S_{nor}. To determine the intended unsolvability rate for a specific scenario, we calculate the percentage of task pairs within that scenario that adhere to this condition. Similar to the preceding evaluation, the final verification step is excluded for the tasks used in this assessment, as that verification screens out task pairs that do not meet this criterion. The results, depicted in Figure 7, show that the intended unsolvability rate surpasses 86% for all the 12 example scenarios. Additionally, a general trend is observed where an increase in intended unsolvability corresponds to a decrease in intended solvability.

E. Solution Switch

An essential criterion in our task designing theory is to ensure that there is a distinct change in the solution between normal and novel tasks. This is necessary to prevent agents from utilizing the same solution for both tasks. To quantitatively assess the extent to which this criterion is met, we introduce the solution switch measure. This measure is satisfied by task pairs that fulfil both the above-discussed intended solvability and intended unsolvability measures. The solution switch rates across all scenarios are presented in Figure 7 and show that the solution switch rate ranges from 79% to 86% across the considered scenarios.

F. Accidental Solvability

In a continuous physical environment with unlimited actions, the possibility of unintended and unforeseen actions solving the tasks is a concern, as agents may exploit such actions during task-solving, compromising the reliability of conclusions drawn from agents' performance. Hence, it is crucial to assess the tasks' quality regarding their vulnerability to unintended solutions. To achieve this, we employ a brute-force strategy utilized by typical Angry Birds playing agents, systematically shooting at all blocks [11]. We exclude shooting at the block that initiates the solution interaction sequence of the task. Similarly to how it is done in [6], the accidental solvability rate (AS_i) is calculated by,

$$AS_i = \frac{1}{N_i} \sum_{n=1}^{N_i} \frac{1}{P_n} \sum_{p=1}^{P_n} (S_{np}) \tag{1}$$

where N_i is the total number of tasks tested in scenario i, P_n is the total number of plays used to test the n^{th} task in scenario i, and S_{np} is a binary variable that is 1 if the n^{th} level in scenario i is solved by the p^{th} strategy, and 0 otherwise. During this experiment, the value of P_n was observed to fall within the interval of 6 to 13, with a mean of 8. The value of AS_i falls within the range of 0 to 1, and a higher value signifies a greater susceptibility to accidental solutions. The results presented in Figure 7 demonstrate that AS_i varies within the range of 6% ($\pm$2 SD) to 12% ($\pm$4 SD) across the 12 scenarios. This range is comparable to the findings of tasks generated in [6], which exhibited a range of 3% ($\pm$1 SD) to 12% ($\pm$5 SD).

In summary, our evaluation satisfactorily demonstrates the effectiveness of our proposed approach for generating tasks that require novelty adaptation capabilities to solve. The presented results affirm that the generated tasks possess the desired attributes of solvability and unsolvability through their designated solutions, demonstrating their alignment with our original task design theory. Moreover, the observed low value in accidental solvability, akin to existing literature, underscores the tasks' resilience against arbitrary solutions. Lastly, our approach can guarantee the physical stability of the generated tasks, as well as generate them in a timely manner. Collectively, the results across these evaluation metrics provide strong evidence that the proposed methodology effectively attains our targeted performance objectives.

VIII. Conclusion and Future Work

In this study, we presented a procedural generation method for creating tasks with novelties in physics-based environments aimed at enabling a robust evaluation of AI agents' novelty adaptation capabilities. This approach involves systematically defining physics-based scenarios by specifying causal sequences of physical interactions between objects and introducing novelties that both disrupt existing causal interactions and construct new ones. We developed a theory for designing tasks that facilitated a robust evaluation of novelty adaptation capabilities and extended an existing grammar to incorporate novelties into physical scenario definitions. The generation process consists of a qualitative phase to restrict the generative space and a simulation phase to configure precise layouts and novelties. Our comprehensive metric-based evaluation verified the effectiveness of our proposed methodology for 12 example scenarios.

For future work, we envision exploring the applicability of this approach to other types of novelties beyond force-applying novelties, such as changes in the physical properties of objects or temporal-based novelties. Moreover, our approach can be readily applied to other physics-based domains similar to Angry Birds [22], [23], [36], where agents make single, one-time actions. Additionally, the applicability of our method to more complex physical environments, where agents interact continuously with the environment, can be explored [37]. Overall, this research paves the way for the systematic generation of physics-based novel tasks, offering a more efficient alternative to manual approaches. The generated tasks facilitate a comprehensive assessment of agents' novelty adaptation skills while providing insights into the nuanced physics mechanics that govern both novelty adaptation and task completion. We believe that these tasks will contribute to the advancement of AI agents with enhanced novelty adaptation capabilities in physical environments.

References

[1] T. Senator, "Science of artificial intelligence and learning for open-world novelty (SAIL-ON)," 2019. [Online]. Available: https://www.darpa.mil/program/science-of-artificial-intelligence-and-learning-for-open-world-novelty

[2] P. Langley, "Open-world learning for radically autonomous agents," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 13 539–13 543.

[3] C. Gamage, V. Pinto, C. Xue, P. Zhang, E. Nikonova, M. Stephenson, and J. Renz, "Novphy: A testbed for physical reasoning in open-world environments," *arXiv preprint arXiv:2303.01711*, 2023.

[4] M. Kejriwal and S. Thomas, "A multi-agent simulator for generating novelty in monopoly," *Simulation Modelling Practice and Theory*, vol. 112, p. 102364, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1569190X21000770

[5] V. Pinto, J. Renz, C. Xue, P. Zhang, K. Doctor, and D. W. Aha, "Measuring the performance of open-world AI systems," *AAAI, Designing Artificial Intelligence for Open Worlds*, 2022.

[6] C. Gamage, V. Pinto, M. Stephenson, and J. Renz, "Physics-based task generation through causal sequence of physical interactions," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 19, no. 1, 2023, pp. 53–63.

[7] C. Xue, V. Pinto, C. Gamage, E. Nikonova, P. Zhang, and J. Renz, "Phy-q as a measure for physical reasoning intelligence," *Nature Machine Intelligence*, vol. 5, no. 1, pp. 83–93, 2023.

[8] J. Renz, X. Ge, M. Stephenson, and P. Zhang, "AI meets angry birds," *Nature Machine Intelligence*, vol. 1, no. 7, pp. 328–328, 2019.

[9] C. Xue, V. Pinto, P. Zhang, C. Gamage, E. Nikonova, and J. Renz, "Science birds novelty: An open-world learning test-bed for physics domains," *Proceedings of the AAAI Conference on Artificial Intelligence, Designing Artificial Intelligence for Open Worlds*, 2022.

[10] C. Gamage, V. Pinto, C. Xue, M. Stephenson, P. Zhang, and J. Renz, "Novelty Generation Framework for AI Agents in Angry Birds Style Physics Games," in *2021 IEEE Conference on Games, COG 2021*, 2021.

[11] M. Stephenson, J. Renz, X. Ge, and P. Zhang, "The 2017 aibirds competition," *ArXiv*, vol. abs/1803.05156, 2018.

[12] N. Shaker, M. Shaker, and J. Togelius, "Evolving playable content for cut the rope through a simulation-based approach," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 9, no. 1, 2013, pp. 72–78.

[13] M. Stephenson and J. Renz, "Generating varied, stable and solvable levels for angry birds style physics games," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, 2017, pp. 288–295.

[14] M. Stephenson, J. Renz, X. Ge, and P. Zhang, "Generating stable building block structures from sketches," *IEEE Transactions on Games*, vol. 13, pp. 1–10, 2021.

[15] M. Stephenson and J. Renz, "Agent-based adaptive level generation for dynamic difficulty adjustment in angry birds," *arXiv preprint arXiv:1902.02518*, 2019.

[16] C. Gamage, V. Pinto, J. Renz, and M. Stephenson, "Deceptive level generation for angry birds," in *2021 IEEE Conference on Games (CoG)*. IEEE, Aug. 2021, pp. 572–579. [Online]. Available: https://ieee-cog.org/2021/

[17] P. Taveekitworachai, F. Abdullah, M. F. Dewantoro, R. Thawonmas, J. Togelius, and J. Renz, "Chatgpt4pcg competition: Character-like level generation for science birds," *arXiv preprint arXiv:2303.15662*, 2023.

[18] F. Baradel, N. Neverova, J. Mille, G. Mori, and C. Wolf, "Cophy: Counterfactual learning of physical dynamics," in *ICLR*, 2020. [Online]. Available: https://dx.doi.org/10.21227/ps5q-8m55

[19] D. Bear, E. Wang, D. Mrowca, F. J. Binder, H.-Y. Tung, R. Pramod, C. Holdaway, S. Tao, K. A. Smith, F.-Y. Sun *et al.*, "Physion: Evaluating physical prediction from vision in humans and machines," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[20] R. Riochet, M. Y. Castro, M. Bernard, A. Lerer, R. Fergus, V. Izard, and E. Dupoux, "Intphys 2019: A benchmark for visual intuitive physics understanding," *ArXiv*, vol. abs/1803.07616, 2020.

[21] K. Yi*, C. Gan*, Y. Li, P. Kohli, J. Wu, A. Torralba, and J. B. Tenenbaum, "Clevrer: Collision events for video representation and reasoning," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=HkxYzANYDB

[22] K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning," *Proceedings of the National Academy of Sciences*, vol. 117, no. 47, pp. 29 302–29 310, 2020. [Online]. Available: https://www.pnas.org/content/117/47/29302.full.pdf

[23] A. Bakhtin, L. van der Maaten, J. Johnson, L. Gustafson, and R. Girshick, "Phyre: A new benchmark for physical reasoning," in *NeurIPS*, 2019.

[24] V. Bapst, A. Sanchez-Gonzalez, C. Doersch, K. Stachenfeld, P. Kohli, P. Battaglia, and J. Hamrick, "Structured agents for physical construction," in *International conference on machine learning*. PMLR, 2019, pp. 464–474.

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

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

[27] S. Goel, G. Tatiya, M. Scheutz, and J. Sinapov, "NovelGridworlds: A benchmark environment for detecting and adapting to novelties in open worlds," *International Foundation for Autonomous Agents and Multiagent Systems, AAMAS*, 2021.

[28] M. Chevalier-Boisvert, L. Willems, and S. Pal, "Minimalistic gridworld environment for gymnasium," 2018. [Online]. Available: https://github.com/Farama-Foundation/Minigrid

[29] P. Feeney, S. Schneider, P. Lymperopoulos, L. Liu, M. Scheutz, and M. C. Hughes, "NovelCraft: A dataset for novelty detection and discovery in open worlds," *arXiv preprint arXiv:2206.11736*, 2022.

[30] CartPole, "CartPole 3D domain," 2022. [Online]. Available: https://github.com/holderlb/WSU-SAILON-NG/tree/master/domains/cartpole

[31] M. Bercasio, A. Wong, and D. Dannenhauer, "Human in the loop novelty generation," *arXiv preprint arXiv:2306.04813*, 2023.

[32] L. Ferreira and C. Toledo, "A search-based approach for generating angry birds levels," in *Proceedings of the 9th IEEE International Conference on Computational Intelligence in Games*, ser. CIG'14, 2014.

[33] A. M. Smith, E. Andersen, M. Mateas, and Z. Popović, "A case study of expressively constrainable level design automation tools for a puzzle game," in *Proceedings of the International Conference on the Foundations of Digital Games*, ser. FDG '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 156–163. [Online]. Available: https://doi.org/10.1145/2282338.2282370

[34] X. Liu, S. Shekhar, and S. Chawla, "Maintaining spatial constraints using a dimension graph approach," *International Journal on Artificial Intelligence Tools*, vol. 10, no. 04, pp. 639–662, 2001. [Online]. Available: https://doi.org/10.1142/S0218213001000696

[35] R. M. Haralick and G. L. Elliott, "Increasing tree search efficiency for constraint satisfaction problems," *Artificial Intelligence*, vol. 14, no. 3, pp. 263–313, 1980. [Online]. Available: https://www.sciencedirect.com/science/article/pii/000437028090051X

[36] K. R. Allen, A. Bakhtin, K. Smith, J. B. Tenenbaum, and L. van der Maaten, "Ogre: An object-based generalization for reasoning environment," in *NeurIPS Workshop on Object Representations for Learning and Reasoning*, 2020.

[37] M. Crosby, B. Beyret, M. Shanahan, J. Hernández-Orallo, L. Cheke, and M. Halina, "The animal-ai testbed and competition," in *Neurips 2019 competition and demonstration track*. PMLR, 2020, pp. 164–176.