A Multi-Agent Collaborative Reasoning Framework for Generating Physics Puzzles

Binze Li[1⋆], Soham Hans[2⋆], and Volkan Ustun[2]

[1] University of California, Los Angeles, USA
binzeli@ucla.edu
[2] USC Institute for Creative Technologies, USA
sohamhan@usc.edu
ustun@ict.usc.edu

Abstract. Achieving expert-level performance through simulation-based training relies heavily on complex and adaptable scenarios, but creating these scenarios manually is often laborious and resource-intensive. Large Language Models (LLMs) offer a promising avenue to automate and enhance scenario generation. However, their reliance on purely sequential text generation in standard prompting settings can hinder consistent understanding in complex systems. We present a multi-agent reasoning framework to leverage LLMs for puzzle generation within the 2D Physics Puzzle Environment CREATE (Chain REAction Tool Environment) to overcome these limitations. This testbed is used as a simplified analogy for scenario generation to allow the development of fundamental LLM capabilities needed to collaboratively design and solve intricate challenges, with a long-term goal of application in domains such as military training. Our framework employs a multi-agent ReAct architecture, integrating reasoning and action feedback loops to dynamically interact with CREATE. By assigning distinct roles, such as solver and designer, to individual agents, our framework preserves the complex reasoning pathways required for solving and generating puzzles — enabling complex reasoning that was too difficult to achieve with basic prompting or single-agent approaches. This work represents a step towards more robust LLM-driven scenario generation by demonstrating the ability of a multi-agent system built on our framework, while interacting with CREATE simulations, to collaboratively perform multi-step reasoning and adapt to environmental constraints. While not yet achieving real-world scenario generation, our findings demonstrate the potential of LLMs to generate solvable puzzles aligned with user prompts. However, we also highlight and address persistent challenges with their reasoning about precise spatial relationships and understanding complex, multi-step chain reactions, which are crucial for generating more advanced scenarios. We conclude by discussing the future role of multi-agent LLM frameworks in creating realistic and adaptable training scenarios for various applications, building upon the foundational capabilities developed in this work. Examples and our full code are available at: https://github.com/binzeli/Puzzle_generation

⋆The two authors contributed equally to this paper.

Keywords: multi-agent systems, procedural content generation, multi-modal LLM, multi-agent LLM, multi-step reasoning

1 Introduction

In highly specialized domains, such as military training, achieving expertise through deliberate practice presents unique challenges, mainly due to the complexity and effort required to create realistic training scenarios. Creating training units is costly and requires specialized expertise, significantly limiting the number and diversity of experiential training, both live and virtual. Consequently, the available scenarios fall far short of the practice hours needed for expert-level performance [5]. Adapting these scenarios to different conditions, such as varied terrain, weather, or new objectives for opposing forces, further exacerbates these challenges.

Mastering complex tasks, such as scenario generation, demands more than raw computational power—it requires structured reasoning, adaptability, and the ability to synthesize multiple perspectives into a coherent whole. Traditional AI-driven approaches, while powerful, often struggle to balance global planning with local decision-making, long-horizon reasoning with real-time adaptability. Single-agent frameworks, constrained by their monolithic structure, lack the fluidity needed for intricate problem-solving. In contrast, intelligence in the natural world emerges from collaboration, specialization, and iterative refinement—principles that inspire our multi-agent reasoning framework. By distributing cognitive effort across specialized agents, our approach mirrors the way human teams tackle complex challenges, dynamically orchestrating expertise to generate, evaluate, and refine solutions in high-dimensional problem spaces.

Although prior research explored scenario generation for military training, the AI and ML tools available even five years ago had significant limitations and could not generate sufficiently complex or adaptable scenarios. Some systems relied on a small set of parameter sliders optimized with reinforcement learning [12], while others employed cognitive task analysis models combined with novelty search to generate new scenarios [6] [7]. Although these earlier methods contributed to scenario generation, they were constrained in adapting to new contexts and producing the level of realism needed for advanced training. In contrast, today's Large Language Models (LLMs) can potentially demonstrate significant advancements in generating complex and adaptive scenarios, for instance, by leveraging vast amounts of data, including historical military operations, current geopolitical contexts, and doctrinal publications. These models may excel at creating contextually nuanced scenarios that can dynamically adapt to changes in constraints. The military is actively exploring the potential of LLMs for these purposes [3] and has recognized their value in generating more dynamic and realistic training scenarios [4].

To address the inherent complexity of scenario generation, we work towards a novel multi-agent reasoning framework that leverages the strengths of LLMs in collaborative problem-solving. Our framework integrates LLMs into a multi-

agent architecture to dynamically generate and evaluate training scenarios within simulation-based environments. Unlike single-agent or basic LLM frameworks, the multi-agent approach enables distributed reasoning and specialization, addressing challenges related to maintaining coherence and context in complex systems. To build up and test the fundamental capabilities of this multi-agent framework for scenario generation, we utilized the Chain REAction Tool Environment (CREATE) [1] as a testbed. CREATE allows us to explore parallels between physics puzzles and military scenarios. For example, much like training scenarios, the physics-based movements of the balls, influenced by tool placement, demand spatial understanding, multi-step reasoning, and comprehension of interactions between different elements.. Learning to effectively use a specific physics tool can also be set as a training objective. Additionally, CREATE provides an executable simulation environment to observe and analyze the outcomes of our designs. Lastly, the generation of physics puzzles is not straightforward, giving significant challenges for LLMs as they require an understanding of spatial relations, are objective-driven, and use multi-step reasoning similar to the challenges prevalent in training scenario designs.

By employing a central ReAct agent to coordinate specialized agents, such as the Solver for tool placement and the Designer for spatial planning, our approach dynamically integrates reasoning and action feedback loops. This architecture enables adaptive decision-making and more robust problem-solving capabilities than traditional methods. This paper presents our development of a novel multi-agent LLM framework that leverages the CREATE physics puzzle environment to generate and solve complex puzzles. We evaluated its performance on various navigation-based puzzles with varying difficulty, comparing it against no-agent and single-agent frameworks. Our findings aim to inform and enhance our scenario design pipeline's procedural content generation capabilities, with a long-term goal of advancing LLM-driven training scenario generation.

2 Background

As background, we briefly introduce the physics puzzle environment designed originally for reinforcement learning and discuss LLM-based approaches and inspirations that motivated our approach for a puzzle-generation pipeline.

2.1 CREATE Environment

The Chain REAction Tool Environment (CREATE) [1] is a benchmark for multi-step, physics-based puzzle reinforcement learning that includes various tools and tasks. The goal is to strategically choose and place tools, such as ramps, cannons, etc., to guide a red ball to a green goal with the help of another blue ball across different environmental setups.

In this work, we evaluate the performance of large language models (LLMs) in generating puzzles within the CREATE environment. This environment provides a standardized and flexible platform for simulating physics-based puzzles, allowing us to integrate our multi-agent LLM system into puzzle generation.

The CREATE environment allows us to run detailed simulations to observe how the placed tools interact with the balls at each step. The physics engine within the environment accurately simulates various physical interactions, including gravity, collisions, and momentum, providing a realistic assessment of the tool placements and their effects. The simulation continues iteratively, step by step, until one of the following conditions is met: (1) the red ball reaches the goal, (2) the balls move out of the frame for several consecutive steps, or (3) the balls cease to move for several steps.

2.2 LLMs for Reasoning

Generating puzzles in the CREATE environment involves a complex, multi-step reasoning process. When humans approach such challenges, they rarely conceive the entire puzzle at once. Instead, they analyze the requirements step by step, iteratively adding elements while considering their impact on the solver's experience. Similarly, supplying an LLM with a structured, multi-step reasoning process can enhance its ability to generate a complete puzzle.

Chain of Thought (CoT) [14] prompting has emerged as a valuable technique for enabling LLMs to tackle complex problems through incremental reasoning. This method encourages the model to logically build upon each step, improving its reasoning and leading to well-structured solutions, such as solvable puzzles.

However, this approach has its challenges. When given unlimited time for reasoning, LLMs may diverge from reality, creating unsolvable puzzles or introducing non-existent elements. The ReAct [13] framework, which stands for Reasoning and Action, offers a compelling solution to this issue. By prompting the LLM to take an action after each reasoning step, this approach enables the model to receive feedback from real-world observations. This feedback loop keeps the model grounded, allowing it to identify and correct errors based on action outcomes. The ReAct framework thus supports a more robust puzzle-generation process, reducing the risks of hallucinations in LLM outputs.

2.3 Related Work

Studies that have used LLMs as game-playing agents, while not a direct match, have used approaches similar to the strategies discussed in our paper. Voyager [18] describes how LLMs can play games like Minecraft by leveraging capabilities in planning and reasoning. Other notable works, such as those involving RDR2 [16], and Doom [17], utilize multimodal LLMs that unify the representation of text and images, allowing agents to perceive game states through visual inputs while seamlessly aligning this perception with text-based logical reasoning. In addition, studies like GATO [19] and SteveEye [20] rely on supervised learning with multimodal instruction sets. The Octopus [21] project showcases multimodal LLMs learning directly from environmental feedback via reinforcement learning. While our work also utilizes multimodal LLMs, we focus on enhancing

reasoning mechanisms rather than just perceiving game states. By integrating visual gameplay representations, we enable LLMs to self-correct by understanding their mistakes.

Furthermore, recent works highlight the potential of LLMs in multi-agent scenarios. For example, CICERO [24] demonstrates strategic reasoning and negotiation in Diplomacy, while Du et.al [25] introduces structured debates between agents to refine factuality and reasoning. These works highlight the utility of multi-agent systems where LLMs interact dynamically to achieve a shared objective. Inspired by these advancements, we explore how multi-agent setups enhance reasoning and performance in game-based contexts.

LLMs have also been utilized in game design to varying extents. For instance, MarioGPT [22] and Todd et al.'s work [23] on generating Sokoban levels exemplify how large language models can be applied in procedural content generation. These approaches typically involve fine-tuning GPT models to generate sequences directly representing game states. In contrast, our work uses LLMs not for direct game state generation but as a tool for reasoning and logical thinking behind the design process.

Since we are testing an application using LLMs to generate training scenarios, to our knowledge, it is a domain with few comparable methods. Previous methods employed procedural techniques or evolutionary models but lacked the needed flexibility and generality [26]. While some research explored LLMs for generating word puzzles [27], they primarily focus on text-based scenarios and do not address visual or 2D environments. This highlights the novelty of our approach in combining LLMs with visual and spatial reasoning tasks to design training scenarios dynamically.

3 Experiment Design

In this section, we outline the task structure for the 2D physics puzzle environment, which consists of two primary components: **puzzle solution** and **puzzle generation**. The LLM's puzzle-solving capability is essential for generating new, solvable puzzles, allowing the LLM to create challenges that are both logical and coherent. To evaluate this capability, we defined a separate task specifically for solving puzzles. Meanwhile, the puzzle generation task is our main task, where the user verbally provides the requirements for LLMs to generate puzzles.

The primary objective of the puzzle is for the blue ball to interact with various tools to push the red ball toward the goal. We simplified the overall design and management by limiting the toolset to three essential, diverse tools, ensuring each serves a unique function and together they can solve most puzzle levels effectively:

1. **Ramp**: Simulates inclined surfaces that enable balls to roll or slide. The ramp can be set at either a 30-degree angle (rolling to the right) or a 150-degree angle (rolling to the left).
2. **Cannon**: Shoots the ball in a specified direction. It can be angled at 75 degrees (shooting to the left) or 105 degrees (shooting to the right).

3. **Fixed Hexagon**: Deflects the ball to a different angle.

CREATE environment offers multiple puzzle types, but for our focus, we have chosen the **navigation challenge** as a key type for both the puzzle-solving and puzzle-generation tasks. In this challenge, the blue ball must traverse complex environments using available tools, which require multi-step planning, spatial reasoning, and tool interaction. Unlike simpler, one-step puzzles (such as moving an object from point A to point B), the navigation challenge demands that the LLMs account for constraints, anticipate tool effects, and strategize efficiently to reach the goal. Our broader objective is to develop a general framework capable of tackling challenges across an entire puzzle type rather than focusing on solving individual puzzle instances.

3.1 Puzzle Solution Task

The puzzle solution task challenges the LLMs to solve 2D physics-based puzzles by strategically placing tools to enable the blue ball to push the red ball to the goal. We have divided these puzzles into five categories of our own design—ordered by increasing difficulty—as shown in Figure 1.

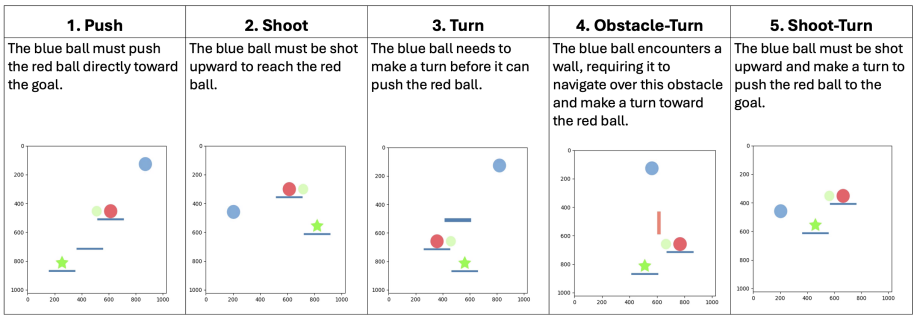

Fig. 1. Five Puzzle Categories

The last three puzzle categories —*Turn, Obstacle-Turn, and Shoot-Turn*— require more advanced reasoning, as they involve multi-step planning and the use of multiple tools. These complex scenarios are designed to assess the LLM's ability to solve dynamic problems and execute intricate strategies.

To assist the LLMs in making better decisions, we added a sub-goal, a green circle that was not part of the original CREATE environment design. This sub-goal can help the LLMs track their progress by determining whether the ball has reached the sub-goal, enabling more informed planning and strategic adjustments throughout the puzzle.

3.2 Puzzle Generation Task

The puzzle generation task involves automatically creating solvable puzzles by placing objects such as the red ball, blue ball, goal, sub-goal, and other essential elements based on the user prompt. We adopt two complementary tasks—Layout-Specified and Tool-Specified—to examine different aspects of the LLM's puzzle generation capabilities. User prompts are shown in Table 1.

Table 1. Prompts for two puzzle generation tasks: Layout-Specified, which focuses on positioning puzzle objects according to prompt requirement, and Tool-Specified, which involves using specific tools to solve the puzzle.

Layout-Specified	**Tool-Specified**
Push: The blue ball on the right must push the red ball down to the goal on the left.	(1) Use a *fixed hexagon* to help the blue ball reach the red ball and push it to the goal.
Shoot: The blue ball on the left goes up to reach a red ball, then pushes it to fall onto the goal on the right.	(2) Use a *cannon* to help the blue ball reach the red ball and push it to the goal.
Turn: The blue ball on the right lands on a separate platform and pushes the red ball on the left to the goal on the right.	(3) Use a *ramp* to help the blue ball reach the red ball and push it to the goal.
Obstacle: The blue ball on the right navigates over a wall to the red ball, then pushes it to the goal on the left.	

In the **Layout-Specified** task, the user specifies the type of layout from four available categories: *Push, Shoot, Turn, Obstacle*. This task allows us to test the LLM's ability to work within predefined puzzle structures, ensuring that it correctly places tools and elements based on the chosen layout. We have tested with these four general layout types, but the framework is designed to accommodate additional layouts in the future.

In the **Tool-Specified** task, which aligns more closely with our goal of training players in tool usage, the user specifies a particular tool to be used in the prompt, and the model needs to generate a puzzle that requires that tool to solve it. The available tools include the *Ramp, Cannon*, and *Fixed Hexagon*. The motivation for this task is to evaluate how well the LLMs can design puzzles that highlight the functionality of specific tools, helping us assess whether the LLM can reason about the tool's purpose and integrate it into generating puzzles.

4 Multi-agent Puzzle Framework

In this framework, we employ a multi-agent system where multiple LLM agents collaborate to handle both puzzle generation and solution tasks. At its core is the

ReAct agent, designed using the ReAct prompting technique, which generates reasoning thoughts and executes actions by invoking predefined functions. When needed, the ReAct agent calls on specialized LLM agents to plan the placement of tools or objects.

4.1 Prompting Technique

Providing steps is a critical aspect of our prompting technique, as it offers a structured framework to guide the model through the complex task of puzzle solution and creation. By breaking the process into clear, sequential steps (e.g., Step 1: Call the designer agent, Step 2: Place objects, etc.), we ensure the model stays focused and methodical. However, these steps are not meant to restrict the model's reasoning capabilities. Instead, they act as a foundation that the model can build upon, allowing it to dynamically adapt and decide the next action based on its observations and the evolving state of the puzzle. For example, if the predefined steps suggest placing a tool and observing the outcome, but the outcome isn't successful (e.g., the blue ball doesn't reach the red ball), the model can reason that it needs to take further action—like adjusting the position of the red ball or consulting the solver agent again—even if those actions aren't explicitly listed in the original steps. In other words, the model isn't just mechanically following a script; it's using the steps as a starting point and then applying its own reasoning to solve problems creatively and effectively.

We also provide an example output format, such as "Thought: [reasoning] Action: [tool or task] Observation: [result]," which structures the model's responses and ensures transparency in its decision-making. Overall, our prompting technique provides clear guidelines to keep the model on track while empowering it to think critically and adapt as needed.

4.2 Collaborative Puzzle Agents for Solving Puzzles

The goal of the puzzle solution phase is to develop an effective strategy for solving existing 2D physics puzzles. Such a strategy involves a ReAct agent and a new LLM helper agent, the Solver, working together to determine the best tool placements to guide the blue ball in pushing the red ball to the goal. The framework is shown in Figure 2.

The ReAct Agent serves as the primary executor in the puzzle solution process. It operates through a predefined set of actions, including "place ramp," "visualize simulation," "place fixed hexagon," "place cannon," and "call solver." These actions are functions explicitly provided to the ReAct agent at the beginning of its operation. It begins by running a simulation within the OpenAI Gym environment to observe the movement of the balls on the given puzzle layout. It then invokes the Solver agent through a function call, passing along relevant simulation data such as trajectory information and a sequence of images. The Solver suggests initial tool placements which are returned by this function. Based on the Solver's guidance, the ReAct agent reasons to place new tools, run simulations, and call the Solver to propose the next tool again. This iterative

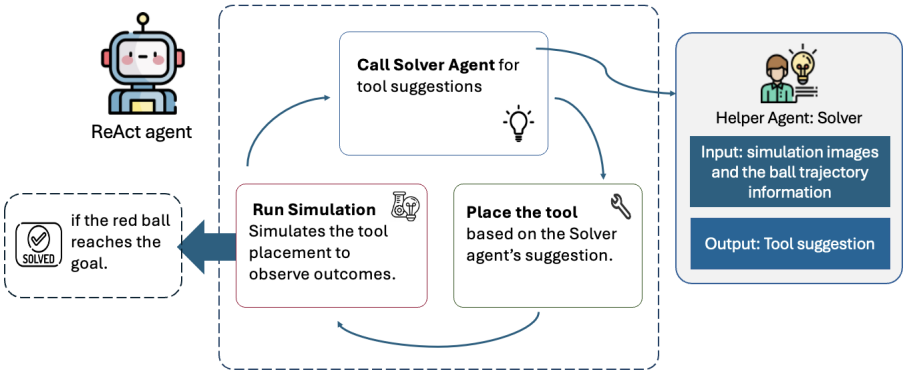

Fig. 2. Workflow for solving a 2D physics puzzle using the ReAct agent with a helper agent (Solver). The ReAct agent initially calls the Solver agent, which suggests tool placements given simulation images and ball trajectory data. The ReAct agent then places the suggested tool and runs the simulations. This loop continues until the red ball reaches the goal.

approach allows the ReAct agent to refine its strategy until the puzzle is solved or no further adjustments are viable.

The **Solver Agent** is embedded as a callable function within the ReAct agent's framework. It operates as a specialized LLM tasked with analyzing the current puzzle state and recommending optimal tool placements. The Solver agent processes the input it receives from the ReAct agent to evaluate whether objectives, such as the blue ball reaching the red ball or the red ball reaching the goal, have been achieved. It then proposes specific tool placements and their relative positions, such as "place a ramp to the left of the red ball."

4.3 Collaborative Puzzle Agents for Generating Puzzles

The puzzle generation task aims to create new 2D physics puzzles that are both solvable and aligned with user prompts. This process involves a more complex multi-agent system configuration, utilizing a ReAct agent with two independent, specialized LLM helper agents: the Designer agent and the Solver agent. The workflow is shown in Figure 3.

The ReAct Agent functions as the primary coordinator in this process, executing a sequence of defined actions through modular function calls. The generation process begins with the ReAct agent invoking the Designer agent via a "call designer" function to propose an initial object layout. The Designer agent provides precise coordinates for key objects such as the blue ball, red ball, goal, and structural elements (e.g., floors or walls) based on the user prompt. These are returned by the function to the ReAct agent who then places these objects within the environment.

After establishing the initial object layout, the ReAct agent initiates a simulation within the OpenAI Gym environment to observe interactions and trajectories. It then invokes the Solver agent to recommend tool placements that ensure the puzzle remains solvable. The ReAct agent places these tools and runs another simulation to test the effectiveness of the setup.

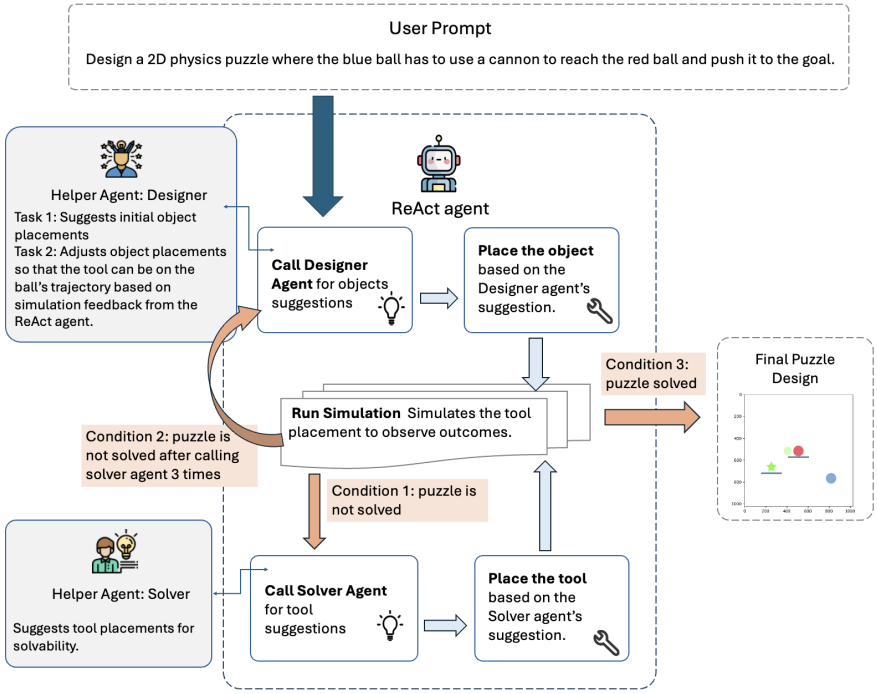

Fig. 3. Workflow for generating a 2D physics puzzle using the ReAct agent with helper agents (Designer and Solver).

If the desired outcome—such as the blue ball reaching the red ball or the red ball reaching the goal—is not achieved after three Solver calls, the ReAct agent opts to call the Designer agent to adjust the positions of existing objects. This approach is taken because adjusting the positions of objects is typically more straightforward and reliable, given the limited toolset and the challenge of precisely calibrating tool angles or lengths with the LLMs. The ReAct agent continues this cycle of adjustments and simulations until the puzzle meets the requirement for solvability.

The Designer Agent is responsible for the spatial planning and placement of objects within the puzzle environment. It performs two key tasks: proposing the initial design of objects or adjusting their positions based on simulation feedback. Initially, it sets the coordinates for the blue ball, red ball, goal, and

other objects like the floor or wall according to the user's prompt. After the initial step, it proposes the adjustment of the object's position to be on the ball's trajectory to ensure that the ball's path will result in effective collisions with tools. This is done based on a series of simulation images and detailed trajectory data sent by the ReAct agent as input during the function call.

The Solver Agent in the puzzle generation task is the same agent as the Solver agent used in the puzzle solution task. It continues to provide strategic guidance on tool selection and placement to ensure the puzzle is solvable.

5 Results

In both tasks, we evaluated three approaches on both puzzle solution and generation tasks, all utilizing GPT-4o as the core LLM, with implementations conducted with LlamaIndex package [2] in Python:

1. **Basic Agent**: A single agent proposes the relative positions of the tools or object, which will be placed by us manually.
2. **Single ReAct Agent**: A single agent reasons and acts to place tools and objects using the ReAct prompting technique.
3. **Collaborative Puzzle Agents**: A single agent reasons and acts using the ReAct prompting technique with assistance from specialized helper agents (Framework described in Section 4).

For all approaches, we adapted in-context learning by providing the LLM with an example puzzle and its solution to ensure it understood the problem before attempting the task. The prompts were identical in both *Single ReAct Agent* and *Collaborative Puzzle Agents* approaches, as we consolidated all prompts from the Designer and Solver agents into a single prompt for the *Single ReAct Agent*. Such an approach allowed the *Single ReAct Agent* without helpers to follow the same steps and use the same prompts as the *Collaborative Puzzle Agents*. With that being said, although *Single ReAct Agent* lacks the specialized helper agents, this agent would also handle the tasks typically performed by the helper agents that are available to *Collaborative Puzzle Agents*.

During testing across all puzzle categories, we kept the context information unchanged, modifying only the input puzzle layout for the puzzle solution task and the user prompt for the puzzle generation task for each case.

5.1 Puzzle Solution Task Results

For the puzzle solution task, Figure 4 shows the performance comparison of the three approaches across five puzzle categories: *Push, Shoot, Turn, Obstacle-Turn, and Shoot-Turn*. Each approach was tested five times in each category, and the number of successful attempts out of five was measured. This task is considered failed if the blue ball has not reached the red ball after placing three tools.

The *Basic Agent* struggled significantly, managing to solve only one puzzle in the *Turn* category while failing to solve any puzzles in the other four categories.

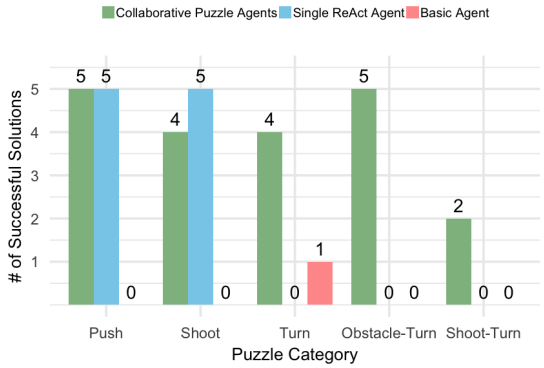

Fig. 4. Comparison of the number of successful puzzle solutions across five attempts for each of three approaches.

The *Single ReAct Agent* performed much better in comparison, achieving perfect scores in both the Push and Shoot categories. However, it was unable to solve any puzzles in the *Turn*, *Obstacle-Turn*, and *Shoot-Turn* categories, suggesting that more complex puzzles require additional support. The *Collaborative Puzzle Agents* showed the most robust performance.

5.2 Puzzle Generation Task Results

We tested how those three approaches work in the puzzle generation task. The results are presented with two metrics: 1) the number of solvable puzzles generated and 2) how many of those solvable puzzles matched the user prompt requirements (evaluated manually). Failure to design a solvable puzzle is detected under two conditions (1) the red ball's position is not adjusted when the red ball has not reached the goal after placing three tools, or (2) after adjusting the ball's location, the blue ball still fails to hit the red ball, or the red ball does not reach the goal. Each agent was also tested five times in each puzzle category.

The *Basic Agent* struggled to generate solvable puzzles across all categories on Table 2. The *Single ReAct Agent* showed moderate success, managing to generate solvable puzzles in select categories, but still encountered challenges in generating puzzles that aligned with the requirement in the user prompt. The *Collaborative Puzzle Agents* demonstrated stronger performance by producing more solvable puzzles and achieved higher alignment with the user's specifications. It generated solvable puzzles in most categories, but not in the *Turn* category which requires LLM to understand how to make a turn. Additionally, it occasionally faced challenges in fully aligning with user prompts.

Figure 5 shows example puzzles and solutions from the three approaches. In the *Obstacle* category of the *Layout-Specified* type, a wall is required to serve as an obstacle that the blue ball must navigate over. The *Basic Agent* placed the wall too close to the blue ball, leaving no space to go over it and making the

Table 2. Comparison of the number of solvable puzzles generated by LLM agents and their alignment with user prompts across five attempts. The first number indicates the number of puzzles successfully solved. The second number in parentheses represents how many of those solvable puzzles align with the specified user prompt.

Layout-Specified Tasks				
Agent Type	**Push**	**Shoot**	**Obstacle**	**Turn**
Basic Agent	0 (0)	0 (0)	0 (0)	0 (0)
Single ReAct Agent	2 (2)	0 (0)	0 (0)	0 (0)
Collaborative Puzzle Agents	**5 (5)**	**5 (2)**	**3 (1)**	**1 (1)**

Tool-Specified Tasks			
Agent Type	**Fixed Hexagon**	**Cannon**	**Ramp**
Basic Agent	0 (0)	0 (0)	0 (0)
Single ReAct Agent	1 (0)	2 (0)	0 (0)
Collaborative Puzzle Agents	**4 (0)**	**3 (2)**	**4 (1)**

puzzle unsolvable. The *Single ReAct Agent* left sufficient space between the blue ball and the wall to place a tool; however, it positioned the ramp in the wrong direction. Therefore it failed to solve the puzzle despite a plausible setup. In contrast, the *Collaborative Puzzle Agents* with a designer successfully generated a solvable puzzle. The Solver helped propose a solution where the blue ball slides down a ramp, navigates over a wall and pushes the red ball to the goal. The puzzle also aligns with the prompt, as all the required objects are included and correctly positioned within the puzzle. A complete sample response generated by GPT-4o on one puzzle generation task is provided in our GitHub page.

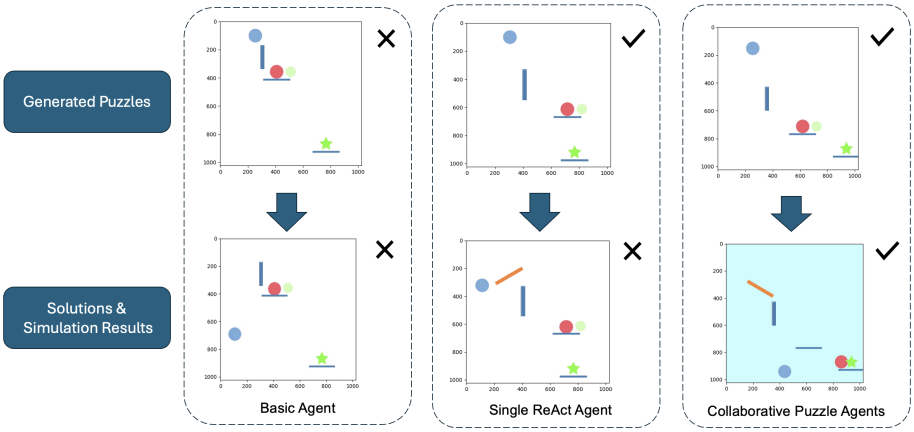

Fig. 5. Examples of generated puzzles and proposed solutions by three agents for the "Obstacle" case in the Layout-specific type. The blue bars are obstacles or platforms.

6 Discussion

In our initial experiments using the *Basic Agent*, we encountered two key challenges: spatial relationship confusion and incorrect tool selection. One frequent issue was misinterpreting the left-right relationship between objects. For example, when the red ball was positioned to the left of the blue ball, *Basic Agent* often incorrectly suggested moving the blue ball to the right instead of to the left. Additionally, the agent struggled to select the correct tools to solve puzzles. This struggle indicated a fundamental limitation in the *Basic Agent*'s ability to process directional and spatial information.

To address the issues, we developed the *Single ReAct Agent* to leverage the LLM's reasoning abilities. Initially, we provided only visual inputs, but this led to errors such as saying the ball slides down the ramp to the left, even though the ball did not even touch the ramp. We then decided to introduce textual descriptions to accompany the visual information provided to the model. After each simulation, detailed trajectory data and positional information—such as whether the blue ball was moving to the right or positioned to the right of the red ball—were included to enhance the model's understanding. This feedback mechanism enabled the *Basic Agent* to refine its decision-making in real-time, reducing spatial confusion and allowing it to handle complex scenarios more effectively and accurately.

While the *Single ReAct Agent* showed improvements in simpler puzzle categories, its struggles with only visual data highlighted that LLMs like GPT-4o still have difficulty accurately interpreting visual details. The agent performed well in puzzles requiring basic reasoning and a single tool but struggled with more complex puzzles involving multiple tools and advanced reasoning, even with guiding questions. It also occasionally failed to follow all instructions, such as not adjusting the red ball's position after repeated failed attempts. Although ReAct prompting and additional textual information improved spatial reasoning in some cases, they were insufficient for consistently generating solvable puzzles or fully aligning with user prompts in complex scenarios.

We then transitioned to a multi-agent approach, introducing two specialized agents: the Designer and the Solver. This setup improved performance by dividing tasks between agents, allowing for more focused reasoning. The Designer agent focused on object placement, while the Solver agent concentrated on ensuring tool placement aligned with solving the puzzle. This collaborative and distributed reasoning system significantly reduced the frequency of left-right confusion and improved the model's ability to interpret directional cues derived from both textual and visual information more accurately.

However, the *Collaborative Puzzle Agents* did not always generate puzzles that matched the user prompt. For instance, in Figure 6, the agent created a puzzle in the *Shoot* category where the blue ball will fall but it should be shot upward to reach the red ball. In another instance in the *Obstacle* category, the agent placed a wall below the red ball, which served no purpose as it did not obstruct the blue ball's path to the red ball.

These examples show that LLMs may not fully understand the user prompt and instead focus primarily on adjusting the objects' positions to ensure a solution. Such issues can lead to a puzzle that technically works but doesn't match the intent or details specified by the user. The fact that LLMs did not always generate puzzle solutions shows that they also lack an inherent understanding of physical principles. This makes it challenging for them to predict how a ball's trajectory will change after interacting with tools or obstacles in the puzzle. Another reason could be that LLMs are trained on textual data and lack exposure to physical systems or spatial reasoning tasks. Generating physics-based puzzles requires balancing multiple constraints (e.g., solvability, logical object placement, adherence to user intent), which is inherently challenging for models not specifically designed for such tasks.

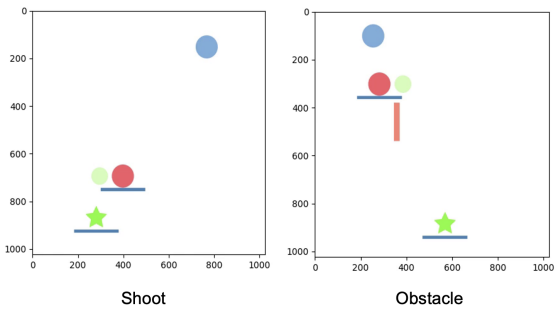

Fig. 6. Example puzzles that do not align with user prompts in the "Shoot" and "Obstacle" categories

Furthermore, during our experiments, we observed that GPT-4o struggled to provide precise coordinates for tool placement. We tested its ability to select an exact location within the ball's trajectory to ensure proper interaction. Although GPT-4o demonstrated strong logical reasoning and could describe plausible strategies, it lacked the precision required to translate high-level reasoning into actionable, spatially accurate outputs. This limitation could arise from the inherent architecture of LLMs, which primarily rely on sequence-based token prediction and lack an explicit internal representation of spatial geometry or continuous coordinate systems. For instance, in the Turn case, it correctly suggested placing a fixed hexagon for the blue ball to bounce off and make a turn toward the red ball but failed to calculate a precise position where the hexagon would function as intended. This imprecision highlighted the gap between text-based reasoning and the model's ability to process spatially grounded tasks. This limitation led us to adjust our approach, asking the LLM agents to provide only relative locations, such as placing the tool to the left or right of an object. We wrote code to help position the tool based on that information within the functions LLM agents would call as its actions.

A fundamental improvement in the future will be fine-tuning the LLMs to enhance their ability to recognize left and right orientations when analyzing visual inputs. This adjustment aims to resolve the persistent issue of directional confusion, allowing the model to rely more on its vision rather than requiring detailed trajectory information. Another important improvement is expanding the multi-agent framework by adding a new evaluator agent dedicated to checking whether the generated puzzles align with user prompts. This agent would serve as a quality control mechanism, ensuring that puzzles more consistently meet the user's specified requirements. Finally, we plan to test newer LLM models, such as GPT-o1, to assess whether they offer improved reasoning capabilities on these complex reasoning and spatial tasks.

7 Conclusion

This paper explored the potential of Large Language Models (LLMs) in generating complex and adaptable levels within a physics-based puzzle environment. Utilizing the CREATE platform, we developed a multi-agent ReAct framework that distributes responsibilities for complex tasks across specialized models. This engineering approach enhances dynamic interaction and reasoning, addressing key challenges such as spatial understanding and multi-step problem-solving critical for advanced training scenarios. However, despite this improvement, challenges remain in ensuring that generated puzzles consistently meet user specifications. The inability of LLMs to accurately interpret visual data and spatial cues underscores the need for further research and development in this domain.

Moving forward, fine-tuning LLMs to improve their spatial reasoning capabilities, integrating evaluative agents for quality control, and exploring newer model architectures will be essential steps in enhancing the effectiveness of LLM-driven scenario generation. Ultimately, our work contributes to the growing field of AI-driven engineering solutions, particularly in specialized domains like military training. The continued exploration of multi-agent frameworks holds significant potential for advancing procedural content generation and addressing the dynamic needs of modern training.

Acknowledgments. The authors acknowledge the use of Large Language Models for assistance with proofreading and grammar checking. All content was reviewed, edited, and approved by the human authors, who take full responsibility for the final manuscript. Research was sponsored by the Army Research Office and was accomplished under Cooperative Agreement Number W911NF-14-D-0005. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation.

References

1. Jain, A., Szot, A., Lim, J.: Generalization to New Actions in Reinforcement Learning. In: Daumé III, H., Singh, A. (eds.) Proceedings of the 37th International Conference on Machine Learning, PMLR, vol. 119, pp. 4661–4672 (2020). https://proceedings.mlr.press/v119/jain20b.html
2. Liu, J.: LlamaIndex. Zenodo (2022). https://doi.org/10.5281/zenodo.1234
3. Caballero, W.N., Jenkins, P.R.: On large language models in national security applications. arXiv preprint *arXiv:2407.03453* (2024)
4. Hill, J.: Hadean builds large language model for British Army virtual training space. https://www.army-technology.com/news/hadean-builds-large-language-model-for-british-army-virtual-training-space/, last accessed 2024/06/15
5. Goldberg, B., Owens, K., Gupton, K., Hellman, K., Robson, R., Blake-Plock, S., Hoffman, M.: Forging competency and proficiency through the synthetic training environment with an experiential learning for readiness strategy. In: Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC), Orlando, FL (2021)
6. Dargue, B., Folsom-Kovarik, J.T., Sanders, J.: Evolving Training Scenarios with Measurable Variance in Learning Effects. In: Adaptive Instructional Systems: First International Conference, AIS 2019, Held as Part of the 21st HCI International Conference, HCII 2019, Orlando, FL, USA, July 26–31, 2019, Proceedings 21, pp. 40–51. Springer, Heidelberg (2019)
7. Folsom-Kovarik, J.T., Rowe, J., Brawner, K., Lester, J.: Toward Automated Scenario Generation with GIFT. Design Recommendations for Intelligent Tutoring Systems **7**, 109–118 (2019)
8. Ericsson, K.A., Krampe, R.T., Tesch-Römer, C.: The role of deliberate practice in the acquisition of expert performance. Psychological Review **100**(3), 363 (1993), American Psychological Association
9. Ericsson, K.A.: The danger of delegating education to journalists: Why the APS Observer needs peer review when summarizing new scientific developments. Unpublished manuscript, http://www.psy.fsu.edu/faculty/ericsson/ericsson.hp.html (2012)
10. Santiago III, J.M., Parayno, R.L., Deja, J.A., Samson, B.P.V.: Rolling the dice: Imagining generative AI as a Dungeons & Dragons storytelling companion. arXiv preprint *arXiv:2304.01860* (2023). https://arxiv.org/abs/2304.01860
11. Zhu, A., Martin, L., Head, A., Callison-Burch, C.: CALYPSO: LLMs as Dungeon Master's Assistants. In: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 19(1), pp. 380–390 (2023)
12. Rowe, J., Smith, A., Pokorny, B., Mott, B., Lester, J.: Toward automated scenario generation with deep reinforcement learning in GIFT. In: Proceedings of the Sixth Annual GIFT User Symposium, pp. 65–74 (2018)
13. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: ReAct: Synergizing Reasoning and Acting in Language Models. arXiv preprint *arXiv:2210.03629* (2022). https://api.semanticscholar.org/CorpusID:252762395
14. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q.V., Zhou, D., et al.: Chain-of-thought prompting elicits reasoning in large language models. Advances in Neural Information Processing Systems **35**, 24824–24837 (2022)
15. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI Gym. CoRR **abs/1606.01540** (2016). http://arxiv.org/abs/1606.01540

16. Tan, W., Ding, Z., Zhang, W., Li, B., Zhou, B., Yue, J., Xia, H., Jiang, J., Zheng, L., Xu, X., et al.: Towards general computer control: A multimodal agent for Red Dead Redemption II as a case study. arXiv preprint *arXiv:2403.03186* (2024)
17. de Wynter, A.: Will GPT-4 Run DOOM? arXiv preprint *arXiv:2403.05468* (2024)
18. Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., Anandkumar, A.: Voyager: An open-ended embodied agent with large language models. arXiv preprint *arXiv:2305.16291* (2023)
19. Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S.G., Novikov, A., Barth-Maron, G., Gimenez, M., Sulsky, Y., Kay, J., Springenberg, J.T., et al.: A generalist agent. arXiv preprint *arXiv:2205.06175* (2022)
20. Zheng, S., Feng, Y., Lu, Z., et al.: Steve-eye: Equipping LLM-based embodied agents with visual perception in open worlds. In: The Twelfth International Conference on Learning Representations (2023)
21. Yang, J., Dong, Y., Liu, S., Li, B., Wang, Z., Jiang, C., Tan, H., Kang, J., Zhang, Y., Zhou, K., et al.: Octopus: Embodied vision-language programmer from environmental feedback. arXiv preprint *arXiv:2310.08588* (2023)
22. Sudhakaran, S., González-Duque, M., Freiberger, M., Glanois, C., Najarro, E., Risi, S.: Mariogpt: Open-ended text2level generation through large language models. Advances in Neural Information Processing Systems **36** (2024)
23. Todd, G., Earle, S., Nasir, M.U., Green, M.C., Togelius, J.: Level generation through large language models. In: Proceedings of the 18th International Conference on the Foundations of Digital Games, pp. 1–8 (2023)
24. Bakhtin, A., Hu, H., Khuong, T.S., Chen, E.D., Yuan, H., Szlam, A., Rocktäschel, T., Weston, J., Choudhury, P., Tromer, B.: Human-level play in the game of Diplomacy by combining language models with strategic reasoning. Science, 378(6623), 1067–1074 (2022)
25. Du, Y., Li, S., Torralba, A., Tenenbaum, J.B., Mordatch, I.: Improving factuality and reasoning in language models through multiagent debate. arXiv preprint arXiv:2305.14325 (2023)
26. Togelius, J., Yannakakis, G.N., Stanley, K.O., Browne, C.: Search-Based Procedural Content Generation: A Taxonomy and Survey. IEEE Transactions on Computational Intelligence and AI in Games, 3(3), 172–186 (2011)
27. Merino, T., Earle, S., Sudhakaran, R., Sudhakaran, S., Togelius, J.: Making New Connections: LLMs as Puzzle Generators for The New York Times' Connections Word Game. In: Proceedings of the 20th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, pp. 87–96 (2024)