Factorio Learning Environment

Jack Hopkins* [1] **Mart Bakler*** [1] **Akbir Khan** [2]

Abstract

Large Language Models (LLMs) are rapidly saturating existing benchmarks, necessitating new open-ended evaluations. We introduce the Factorio Learning Environment (FLE), based on the game of Factorio, that tests agents in long-term planning, program synthesis, and resource optimization. FLE provides exponentially scaling challenges—from basic automation to complex factories processing millions of resource units per second. We provide two settings: (1) *lab-play* consisting of 24 structured tasks with fixed resources, and (2) *open-play* with the unbounded task of building the largest factory on an procedurally generated map. We demonstrate across both settings that models still lack strong spatial reasoning. In lab-play, we find that LLMs exhibit promising short-horizon skills, yet are unable to operate effectively in constrained environments, reflecting limitations in error analysis. In open-play, while LLMs discover automation strategies that improve growth (e.g electric-powered drilling), they fail to achieve complex automation (e.g electronic-circuit manufacturing). We release FLE as an open-source platform[1].

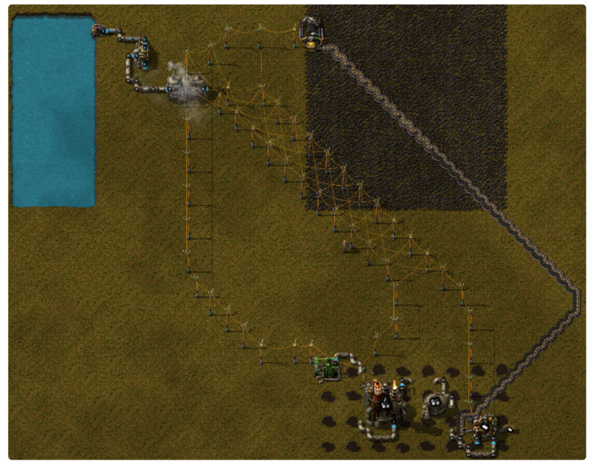

Figure 1. **A plastic bar factory created by Claude 3.5 Sonnet in lab-play.** The factory consists of a electricity steam generator (top-left), a coal mine (top), a crude-oil to petroleum gas pipeline (bottom) and a chemical plant (bottom-right). The chemical plant creates plastic bars using the coal and petroleum gas as inputs. By themselves, the cumulative raw resources generate a production score of 224. With this specific layout, the factory creates 40 plastic bars per 60 in-game seconds, for a production score of 352.

1. Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities at solving complex question-answer (QA) problems, saturating benchmarks in factual recollection (Hendrycks et al., 2021), reasoning (Cobbe et al., 2021) and code prediction (Chen et al., 2021).

The strong performance across these diverse tasks suggests that LLMs have developed sophisticated reasoning capabilities, leading researchers to explore whether models can act as autonomous agents (Yang et al., 2023). This has motivated a number of new agentic benchmarks focusing on long-term planning (Liu et al., 2023; Ruan et al., 2023), learning in complex environments (Paglieri et al., 2024; Jimenez et al., 2023) and reliably learning from mistakes (Xing et al., 2024; Yamada et al., 2023; Kambhampati et al., 2024). However, similar to QA settings, these agentic benchmarks are likely to face saturation due to their natural completion states; which impose an upper bound on performance and limit our ability to differentiate superhuman models.

We introduce the **Factorio Learning Environment** (FLE): a novel evaluation framework built upon the game of Factorio that uniquely addresses this limitation by enabling unbounded agent evaluation with no natural completion state. In this environment, agents must navigate rapidly scaling challenges from basic resource gathering to complex automation while managing an exponentially scaling technology tree - creating natural curricula for evaluating increasingly capable agents.

Agents are tasked with producing the largest possible factory, whose performance is measured through production

*Equal contribution [1]Independent Researcher [2]Anthropic. Correspondence to: Jack Hopkins <jack.hopkins@me.com>.

Preliminary work. Under review.

[1]https://github.com/JackHopkins/factorio-learning-environment

throughput, which ranges from early-game rates of $\sim$30 resources/minute to advanced systems processing millions of resources/second. This enables us to meaningfully differentiate agents by measuring the order of magnitude of resources that they can produce, avoiding saturation by agents even as models become dramatically more capable.

Existing resource management environments such as Minecraft (Guss et al., 2019) or Nethack (Küttler et al., 2020) do not demand the precise industrial optimization present in Factorio. For resource processing chains, producing basic electronic circuits (an early-game staple) requires coordinating 10+ machines processing approximately 15 items per minute. For example, a single rocket component requires orchestrating 60+ interlinked machines manufacturing 1000+ items per minute. The precision required, where a single misaligned machine can cause a factory-wide gridlock, creates a natural curriculum, testing both basic automation and advanced system optimization.

Agents interact with the FLE via synthesizing Python programs to alter and observe the game state, using the tools included in the environment in a Read-Eval-Print Loop (REPL). This feedback loop mirrors the day-to-day workflow of human programmers, who write provisional code to probe how systems behave, interpret the results, then refine their mental model of the system. In this sense, the agent's program acts as the cumulative representation of its current knowledge and strategies for managing the complex resource pipelines in Factorio.

We evaluate six frontier LLM models in this environment in an agentic setting. In our qualitative analysis, we study the agents capabilities for spatial reasoning, long-term planning, and error correction. Our results show that even the most advanced models struggle to coordinate more than six machines when automatically producing items with over three ingredients, even after 128 environmental interactions.

We summarise our contribution as follows:

- The introduction of the Factorio Learning Environment, an agentic evaluation of long-term planning and resource management and allocation.
- Evaluations of frontier models in FLE lab-play, a set of 24 controlled tasks requiring agents to build factories with increasing levels of complexity and scale. Claude-3.5-Sonnet (the strongest performing model) only completes 7/24 tasks and shows limitations in spatial planning in more complex objectives; demonstrating large head-room for performance.
- Evaluation of frontier models in the FLE open-play, an unbounded setting in a full Factorio game map. We find more capable agents who invest heavily into technological research and advancements achieve quantitatively different slopes on a log-reward, log-step graph.
- A qualitative analysis of the results across capabilities such as error-correction and long-term planning. We identify a gap in models' ability to perform intelligent error correction, iteratively build upon prior work and conduct exploration.

2. Factorio Learning Environment

Our main contribution is the release of an open-source framework, which includes i) a high-level Python API to Factorio, ii) a persistent coding environment for LLM agents to interact with the game through iterative program synthesis, and iii) a Python object model of game entities.

The environment is procedurally generated, deterministic at runtime (set by a random seed) and is 4×10^{12} square tiles in size. We provide a laboratory environment with accessible resources for benchmarking agents in a controlled setting.

2.1. Environment Dynamics

Factorio is a resource management and automation game in which players spawn on a world containing raw resources such as water, iron ore, and coal, and must orchestrate increasingly complex production and logistic chains to ultimately produce a rocket and (optionally) escape. The game contains over 200 entity types, with a technology tree that unlocks more efficient buildings, resource production chains and multiplicative throughput bonuses. Research enforces a steep resource progression, with late-game technologies such as the `rocket-silo` demanding 300 times more resources than early `automation` research[2].

Player strategy and factory architecture evolves dramatically as technology progresses. The early game centres on manual crafting and basic automation, with factories daisy-chained together using direct insertion between machines. These designs are primarily constrained by manual building speed and resource gathering, favouring cheap, immediate solutions -e.g `stone-furnaces` provide better returns on investment than `steel-furnaces` until fast-belt technology (`logistics-2`) is available. As they progress, players typically adopt main bus designs with centralized resource production and distribution, enabling more organized scaling of production. Late-game strategy shifts again, toward massive parallel construction and logistics networks, with factories ultimately evolving into distributed complexes connected by high-capacity train networks. These advanced stages emphasize space-efficient, high-throughput designs enabled by technologies like beacons and stack inserters,

[2]This progression approximately follows an unbounded geometric relationship between resource cost C and research tier N – $C[N] = 1000 \times 2^{(N-1)}$

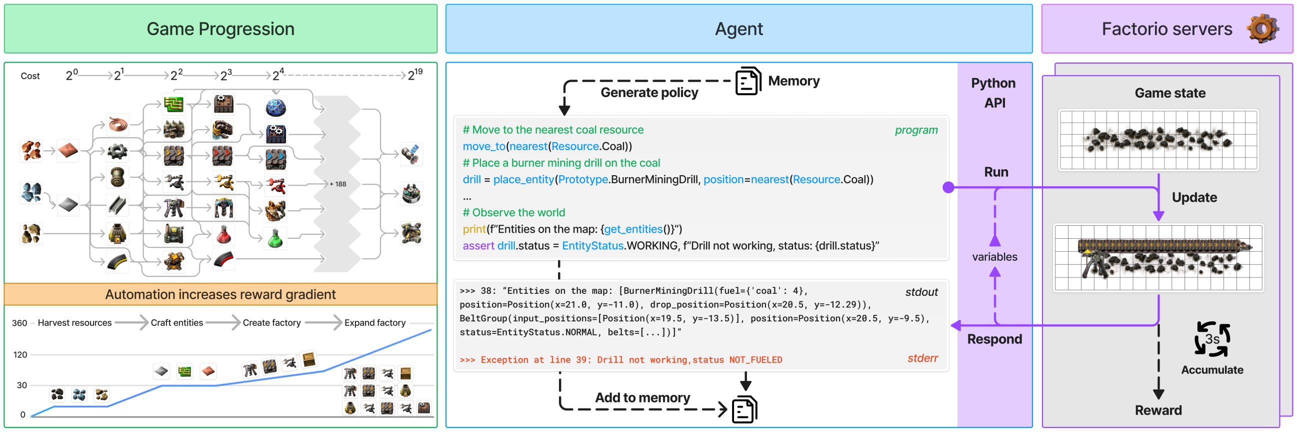

Figure 2. **Illustration of the Factorio Learning Environment (FLE).** FLE is based on the popular construction and management simulation game *Factorio*. Left: The open-ended goal of the game is to create the largest factory possible. The game enables agents to invest in (an infinite number of) technological advances to produce more resources per second. Middle: Agents interact with the game by using an interactive Python Interpreter, where they take actions and print their observations in a Read-Eval-Print loop. By using the Python namespace, agents may store variables and define functions for later use. We provide a Python API to Factorio which allows direct interaction with the environment. Right: The agent may issue commands to the game server in order to interact with the environment (with associated time penalities), and receive a response as feedback. If the agents chooses, it may view its own production statistics.

```python
# 1. Get iron patch and place mining drill
drill = place_entity(
    entity=Prototype.MiningDrill,
    position=nearest(Resource.IronOre)),
    direction=Direction.NORTH
)
# 2. Add output storage
chest = place_entity_next_to(
    entity=Prototype.IronChest,
    reference_position=drill.drop_position,
    direction=Direction.SOUTH
)
# 3. Verify automation chain and observe entities
assert drill.status == EntityStatus.WORKING
print(get_entities())
```

Figure 3. **Example of an FLE program** used to create a simple automated iron-ore miner. In step 1 the agent uses a query to find the nearest resources and place a mine. In step 3 the agent uses an assert statement to verify that its action was successful.

and are thousands of times larger than early game factories.

2.2. Environment Interface

Agents interact with FLE through a **REPL** (Read-Eval-Print-Loop) pattern, observing the current game state via previous program output streams, then generating and executing Python code to implement their intended actions, and finally returning useful feedback for the next iteration.

Agents are provided with the Python standard library, and an API comprising methods designed to balance expressiveness with tractability (see Appendix D.1). These comprise 10 observation methods and 13 action methods. Observation methods (e.g `nearest`, `get_entities`) retrieve information about the environment, and action methods (e.g `move_to`, `craft_entity`) modify the environment.

Each method returns a typed object (e.g an `Inventory`) which can be stored as a variable in the Python namespace and referenced later in the episode. The namespace acts as an episodic symbolic memory system, and saved objects represent part of the environment at the moment of query, becoming stale as the game state evolves, requiring the agent to re-query when appropriate. This design enables agents to maintain complex state representations and build hierarchical abstractions as the factories scale.

Agents observe **stdout** and **stderr** - the output streams of their program. Thus, agents may intentionally print relevant objects and computations to the output stream to construct observations. Selecting relevant attributes of objects enables token-efficient observation, e.g:

```python
# Observe the status of all Furnaces in-game
print([e.status for e in
     get_entities({Prototype.StoneFurnace})}
>>> 2: 'WORKING', 'WORKING', 'NEED_INGREDIENTS',
    'OUT_OF_FUEL'
```

Mistakes in the code or invalid operations raise typed exceptions with detailed context that is written to **stderr**. This enables agents to *reactively* debug their programs after execution, and *proactively* use runtime assertions during execution to self-verify their actions. Programs that take too long to execute are terminated, to prevent runaway control flows (e.g while True).

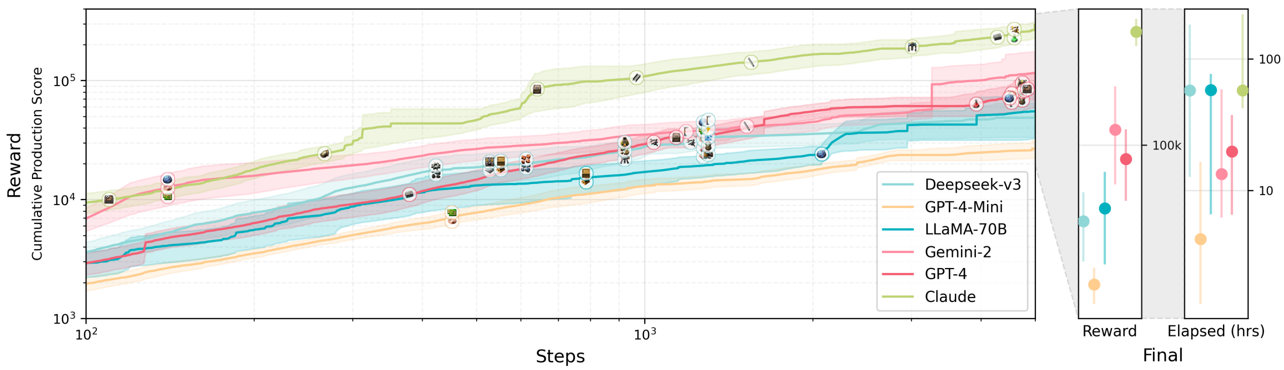

Figure 4. **Models are differentiated by score in Open-Play.** Agents are given the instruction to *build the biggest possible factory*. Left: We find that by evaluating PS against steps (server calls) we can clearly differentiate stronger models from weaker ones in a log/log projection. We overlay milestones, showing the first time the median agent was able to create a new type of entity. Right: We plot the final reward and elapsed game time after 5k steps. We find that while weaker models show promise early-game, they struggle to progress when automation and logistics are required. We report median and standard error over the independent runs.

An environment "step" is a single submission to the Factorio server, which returns the stdout, stderr, rewards and in-game wall-clock time (see Figure 2).

Agents are able to enhance their internal representation of the game state in 2 ways: (i) they can define utility functions for reuse throughout an episode, to encapsulate successful logic; and (ii) they can define classes in the namespace to better organize the data retrieved from the game.

2.3. Reward Structure

We use Factorio's built-in production tracking system, which enables us to define two complementary reward signals:

Production Score (PS): A continuous measure of economic activity based on the value of all items produced. This metric increases as agents refine raw ores into manufactured goods and create automatic factories. As production chain throughput scales exponentially, PS can vary by multiple orders of magnitude (a rocket launch requires $\approx 10^7$ raw resources). PS provides a naturally unbounded measure of performance, which is sensitive to increasing automation complexity. The game's price calculation system assigns higher value to items with more complex production chains, creating a reward structure that encourages sophisticated factory designs. For the full pricing system, see Appendix A.

Milestones: A discrete set of achievements for producing novel item types (e.g. building an `inserter` for the first time, assembling `electronic-circuits`, etc.) and researching technologies. This captures both the diversity of an agent's exploration across Factorio's tech tree, and what level of item complexity they were able to achieve. As Factorio supports researching an infinite technologies (with multiplicative bonuses), milestones can be used to measure performance at all levels of capability.

2.4. Implementation Details

The FLE comprises a Python client and Lua server communicating synchronously via RCON over TCP[3]. The client provides the stateful environment interface and APIs, while the server manages game state execution in the official Factorio multiplayer server. The server can be run in headless mode for efficient parallelization. The object model represents most early to late-game entities (detailed in Appendix D.1). FLE is compatible with v1.110 of Factorio, and requires a single purchased game license, as each server must be "activated" by any official client at startup. FLE is also easily extensible by the community. Designing new tools requires implementing a client-side controller (Python) and a server-side action (Lua) which will automatically load and update the API schema for subsequent agent runs.

We benchmark the Factorio Learning Environment on a MacBook Pro M4 with 128GB RAM. The headless server achieved the highest throughput, processing an average of 218 operations per second across core API functions, with peak performance of 603 ops/sec for basic operations like crafting. The Python interpreter introduces approximately 3x overhead, reducing average throughput to 68 ops/sec. Complex spatial operations (`connect_entities`) are consistently the slowest at 25-48 ops/sec due to pathfinding requirements. Basic inventory operations (`craft_item`, `extract_item`) achieve highest throughput at 276-545 ops/sec. The headless configuration provides a 1.75x speedup over the game client (see Figure 11). We make the environment publicly available [4] .

[3] Roughly 80k LoC in total

[4] https://github.com/JackHopkins/factorio-learning-environment

3. Experiments

To evaluate agent capabilities in FLE, we introduce two settings and a novel agent scaffolding.

3.1. Settings

Lab Play - requires the agent to create a factory with a specific production throughput in the constrained lab environment. These tasks are designed to evaluate the capabilities of an agent to create automatic structures in an open-ended manner, requiring creativity, spatial understanding of the map and long-horizon planning.

We task agents to build fully automatic production lines of 24 distinct target entities of increasing complexity, starting from a single resource mine requiring at most 2 machines (making `iron-ore`) to a late game entity requiring the coordination of close to 100 machines (making `utility-science-pack`). The target entities cover items from early to late game and the agent must use a wide variety of machines present in Factorio (drills, furnaces, assembling machines, oil refineries, chemical plants). As the task difficulty naturally increases with resource requirements of target entities this provides a measure of the complexity agents are capable of creating in a limited number of steps. All tasks start with an inventory containing sufficient entities to complete the task and all research is unlocked for the agent in *lab-play*. Additional information is in Appendix G.

Each task runs a trajectory of *128* API calls. After every agent step, the throughput of the created structure is evaluated throughout a 60 second holdout period in-game, and the task is deemed completed if the throughput of the structure is above the target throughput at any step i. All successful production lines were manually examined to guard against reward hacking (for instance, agent manually inputting ingredients into an assembler as opposed to creating an automatic connection). The target throughput is *16* for solid items (for instance electronic circuit, military science pack, plastic bar) and *250* for fluids (for instance petroleum gas, lubricant, heavy oil) during the holdout period. We report the mean success rate of each task with 8 runs per task.

Open Play - In addition to the structured *lab-play* tasks, we evaluate each agent in a purely open-ended setting. The agents spawn into a procedurally generated world with unbounded space and resources, and are tasked to "build the largest factory possible", allowing the agents to decide how best to advance in the game. To progress long-term, agents must show proficient long-term goal-setting, entity and resource planning and spatial reasoning capabilities when creating automation structures. Agents must be capable of using the API, querying the environment for unknown information and reasoning over observations to plan successfully.

We use two metrics to evaluate progress in the game: *Production Score* (PS) and *Milestones*. While the PS acts as the reward and is affected by exploitation, milestones give an overview of how much of the game and technology tree the agent has explored.

Each agent plays until the maximum trajectory length of 5000 is reached. After every agent step, the production throughput is tracked and reward computed. We execute 8 independent runs for each agent, and report the median.

3.2. Agent Scaffolding

We consider a simple step-by-step prompting approach as a baseline implementation for agents to interact with the environment. The input prompt of the agent consists of the API schema A, a guide G describing how to use the API tools with code examples and the memory M of the agent consisting of past policies with environment observations. A detailed description for the guide, API schema and an example memory state is brought out in Appendix I. Given the inputs, the agent is tasked to identify the most useful next step and generate the Python policy P that carries out actions in the environment to achieve the step. The policy is executed in the environment and added to the memory M with the environment observations (**stdout**) and error messages (**stderr**). The updated memory M is then used as input to generate the next policy and enables the agent to gather information from the environment and use observations to guide future step generation.

Memory - At every policy generation step the agent uses information from the memory M. Memory consists of past policies and their respective environment observations (stdout and stderr after every policy execution). To limit the memory token count, past observations and policies that are further than 16 steps in the past are summarised into a report of 1024 tokens. This allows the agent to execute arbitrarily long traces in the environment without unreasonably large memory input token requirements.

Language Models - We evaluate state-of-the-art closed source models including Claude 3.5-Sonnet (Anthropic, 2024), GPT-4o and GPT-4o-Mini (OpenAI et al., 2024), Deepseek-v3 (DeepSeek-AI et al., 2025) and Gemini-2-Flash (Team et al., 2024). We also evaluate Llama-3.3-70B-Instruct (MetaAI, 2024). Each model is sampled at temperature 0.5. Model timestamps are in Appendix E.

4. Results

We analyse agent performance during *open-play* and *lab-play*, and observe common patterns amongst trajectories from both settings. We report experimental costs in Table 3.

Insight 1: Coding agents perform better in the FLE.

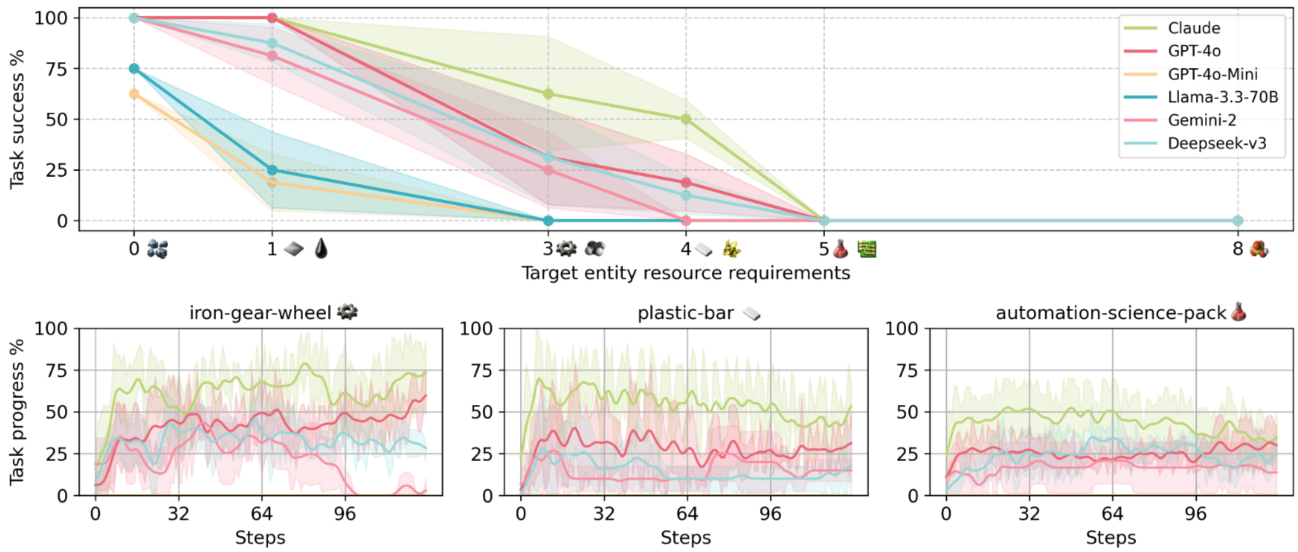

Figure 5. **Agents are unable to consistently build complex and efficient factories in Lab-Play**. Top: We measure the mean and standard deviation of task success rates across the first 8 complexity levels. We observe a clear decrease in average task success rates as the crafting complexity of the target entity increases. Bottom: We show the mean and std of task progress (percentage of target ingredients and its sub-ingredients agents factories produce at each time-step) in three tasks of increasing difficulty accross 8 runs per task. In harder tasks, agents show trends of initial rapid progress followed by stagnation or decrease. This is due to agents being unable to scale up initial production or add new sections to factories required to successfully reach the target production levels and often breaking existing structures during the process. The lack of consistent progress is also observed through the large variance in task progress across runs.

We find that stronger coding agents achieve higher production scores across both settings. In *open-play*, Claude outperforms other models in both median PS (293 206) and milestone count (28), surpassing the early-game resource extraction phase and partially investing in technology research - constructing and powering a `lab`, dedicating production to `science-packs` and unlocking electric `inserters`, (see Figure 6). In comparison Llama-3.3-70B (54 998 PS, 26 milestones) made initial progress but did not develop production lines of >3 entities and struggled with both creating complex structures and scaling up existing production. Similarly in *lab-play*, Claude performed the best, managing to create automatic structures typically seen in Factorio's early game; specifically, compact drilling lines coordinating 10+ machines across up to four factory sections (see table 1). In comparison, Llama-3.3-70B is capable of only creating the most trivial of factories.

Insight 2: Agents lack spatial reasoning and are unable to iteratively improve on factories. A key characteristic for success in *open-play* and *lab-play* involves iteratively combining multiple factory sections to create complex production lines. In *open-play*, while Claude was able to scale up automation from the early-game, GPT-4o, GPT-4o-Mini and Llama-3.3-70B typically succeeded only at maintaining a small number of production lines. In lab-play, it can be seen how the success rate of tasks decreases proportionally to the increase in crafting recipe complexity of the target entity (see Figure 5). For instance, creation of automation science packs requires multiple mining and smelting sections (`iron-plate` and `copper-plate`), a `iron-gear-wheel` assembly section, `automation-science-pack` section and a `steam-engine`. While agents are able to make initial progress in this task by creating electricity setups and plate production lines, they are unable to improve on the factory and add the required assembly sections. Frequent failure cases were trying to place entities too close or on-top of each other, not leaving room for connections or incorrect placement of inserters. These are all limitations in spatial reasoning and result in agents only being able to consistently create production lines for low complexity items and low overall performance in *lab-play* tasks (see Table 1).

Insight 3: Agents use the API in different ways. We evaluate trajectories with automatic checkers to evaluate how successful models are at using the FLE API. We find that models exhibit different coding styles, with GPT-4o using more assert checks in within their code than Claude 3.5. Conversely GPT-4o uses significantly fewer prints. These suggest models use very different approaches to explore and engage with the FLE. Using prints suggests being uncertain of state, and exploring new areas, whereas assert statements are likely used to clarify existing knowledge (see Table **??**).

Insight 4: Planning is essential to open-play perfor-

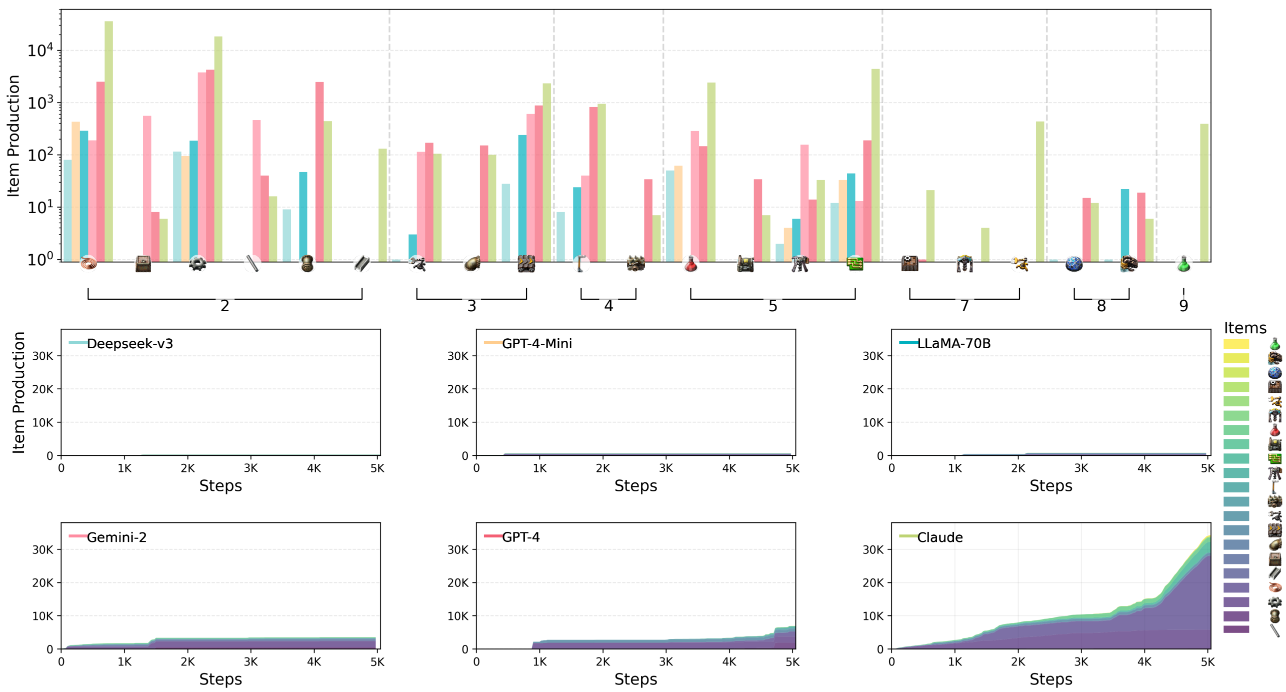

Figure 6. **Open-ended challenges highlight differences in objective setting and general capability.** We illustrate the rates at which various models produce items with multiple antecedent ingredients in the *open-play* setting. Claude 3.5-Sonnet immediately begins complex crafting and invests in research, ultimately unlocking `electric-mining-drills` at step 3k, the deployment of which boosts production of `iron-plate` thereafter. Less advanced models, like GPT-4o-Mini, produce insignificant quantities of multi-ingredient items. Deepseek produced fewer complex resources in open-play than its lab-play performance would suggest, indicating weaker objective-setting compared to general capability in-game.

Model	Lab-play task success rate (%)
Claude	21.9
GPT-4o	16.6
Deepseek-v3	15.1
Gemini-2	13.0
Llama-3.3-70B	5.2
GPT-4o-Mini	4.2

Table 1. **Stronger coding models achieve higher task success rates in Lab-Play**. We observe a correlation between coding and reasoning abilities of base models and the task success rates, where the stronger models have higher success rates in lab-play tasks. Claude, GPT-4o, Deepseek and Gemini-2 are able to only pass simpler tasks requiring the construction of factories consisting up to 3 sections while Llama-3.3-70B and GPT-4o-Mini succeeded in tasks requiring single-section factories. All models struggled with the increasing scale and complexity requirements for tasks resulting in low overall success rates.

mance. In *open-play*, agents are given an open-ended goal and need to create meaningful sub-objectives themselves to make long-term progress. We observe that agents often set short-sighted objectives, for instance manually crafting a large quantity of singular entities without a long-term plan (Gemini-2.0 manually crafted 300+ wooden chests over 100 steps), not significantly investing into research (except for Claude) or creating small individual factories as opposed to scaling up existing production. This creates a discrepancy between *lab-play* and *open-play* results where in lab-play Gemini-2 and Deepseek show capabilities in creating early-game automation (see Figure 5) but rarely attempt creating factories in open-play, resulting in poor complex entity crafting statistics (See Figure 6) and lower production scores.

Insight 5: Agents which invest in technological progression achieve much higher PS. Investing into technology progress in open-play is a trade-off, where agents incur a short-term resource cost to unlock items enabling long-term higher throughput. Although research is crucial for creating higher efficiency factories, only Claude consistently invests resources into researching new technologies in open-play. The result can be seen from step 3k, where Claude starts deploying `electric-mining-drills` and PS grows

Model	L	P%	A%	F%	AF%	C%	En%
Claude-3.5	65	43.3	2.0	50.6	0	3	97
GPT-4o	81	10.3	12.8	10.2	2	12	86
DeepSeek-v3	37	25.4	13.9	25.3	0	2	98
Gemini-2	133	16.2	0.0	16.6	1	46	53
Llama-3.3-70B	38	23.9	12.9	23.7	0	24	76
GPT-4o-Mini	77	36.0	0.0	31.6	15	6	79

Table 2. **Models exhibit contrasting coding styles**: Analysis of code submitted by different language models, showing average lines per program (L), percentage of lines that were print statements (P%), percentage of lines that were assertions (A%), and percentage of programs that failed (F%). For programs that failed by error type we track the proportion of assertion fails (AF%), code errors (C%), and environment errors (En%). Claude-3.5 favours a REPL approach with high print usage and failure rates, while GPT-4 opts for defensive programming with assertive validation and fewer resulting environmental errors. Gemini-2 produces the longest programs (133 lines on average) but makes the most code errors (e.g accessing non-existent variables).

by a factor of 1.5x (from 200k to 300k), in Figure 6.

Insight 6: Agents fall into degenerate debug loops. A critical component for successful runs was an agents' ability to interact to previous error logs and carry out error correction. In *lab-play*, in successful task completions, 56% of steps resulted in program execution errors (from which agents recovered), and in *open-play* ranged from 29.7% to 76.4%. Claude, GPT-4o and Deepseek were capable of simpler error correction when incorrectly using the API or when crafting entities. Anecdotally, the agents were not proficient at debugging complex environments. For instance, when debugging non-working structures or factories where the throughput was not at expected levels, agents often focused on whether all singular entities were working but did not investigate whether the topology of the whole structure was correct. In *lab-play*, this limitation is illustrated by the frequent decrease of task performance across steps in Figure 5 where the agents broke existing working structures due to incorrectly identifying the root-cause of problems. Agents often fell into a loop of greedily repeating the same fix rather than exploring additional potential sources of the problem. This can be seen in the flatline behaviour during open-play in Figure 4 with no PS progression. For instance, in one run GPT-4o used the same API method incorrectly for 78 contiguous steps (from Step 120), receiving identical error message each time. On two occasions, GPT-4o-Mini simply gave up and repeatedly asked to be reset - see Appendix F.

These limitations show the difficulty of FLE and that state-of-the-art LLMs—even with REPL-style feedback and extensive prompting—are still in the early stages of mastering large-scale, open-ended factory design.

5. Related Work

Games have long served as fundamental benchmarks for artificial intelligence research, providing standardized environments with clear metrics, rich observational data, and natural difficulty gradients. (Campbell et al., 2002; Silver et al., 2016; Berner et al., 2019).

Recent work has explored using LLMs as game-playing agents. Environments like ALFWorld (Shridhar et al., 2020) combine language understanding with embodied tasks, while MineDojo (Fan et al., 2022) leverages Minecraft as a sandbox for testing general-purpose agents through 3,000+ diverse tasks spanning survival, harvesting, and creative building. While these environments excel at evaluating breadth of capabilities, their fundamentally linear progression systems limit their ability to differentiate highly capable agents. Even with rich task suites, their resource requirements remain relatively modest compared to Factorio's exponential scaling.

Many benchmarks exist for agentic coding such as (Jimenez et al., 2023; Hendrycks et al.), which evaluate Python against a stateful system. These benchmarks often involve fitting specific test conditions, or task descriptions. In comparison, our win condition is open-ended, requiring multi-step planning and resulting in thousands of submissions; requiring much longer contexts than other benchmarks.

Management simulation games like OpenTTD (Ope) have explored aspects of resource optimization, but lack precise mechanics and exponential scaling that would enable quantitative measurement of small improvements in agent capabilities. Text-based environments like Jericho (Hausknecht et al., 2019) test language understanding through interactive fiction, but lack the spatial reasoning and timing requirements inherent in factory design.

Factorio has seen prior research interest for closed-domain settings (Reid et al., 2021), with a focus on integer programming models, meta-heuristics and evolutionary reinforcement learning to tackle logistics challenges. We build on this foundation to offer a standardized text-based interface for learning agents to solve open-ended challenges in long-term planning, spatial reasoning and factory optimization.

6. Limitations, Future Work & Conclusion

In this work, we introduce the Factorio Learning Environment (FLE), a novel open-ended framework for evaluating the capabilities of agents in an open-ended environment.

A major concern for any environment benchmark is reward hacking (Clark & Amodei, 2016; Skalse et al., 2022). In our setting this could involve two main attack surfaces: either through Python API (as seen within Denison et al. (2024)) or within the Factorio game-engine itself. During our evaluations, while we observed no direct examples of reward hacking of either interface, we did observe that the agent was able to occasionally trigger resetting the Factorio gamestate; That said, we applied little optimisation pressure on agents - we highlight this as a caution for those developing agents on FLE.

Secondly, unlike the base game of Factorio, which uses mouse and screen, our approach involves a Python interface. Whilst the authors were, in equivalent game time, able to outperform frontier agents, it is unclear if achieving end-game goals (e.g. escape the world or build rockets) is achievable to humans using only an API in a reasonable time-frame. We did however prove that each step in the chain to launch a rocket was achievable from the previous step and that all tasks in *lab-play* can be completed.

Even without human baselines, we believe that FLE is a useful benchmark, as the comparative scores between agents still informs us of their relative ability at planning, spatial reasoning and resource management.

Currently, some mid and late-game entities are not explicitly modelled in FLE (specifically trains, logistics robots and agent-programmable circuit networks). This is appropriate given the state of evaluated models. As FLE is open-source, we are committed to release a complete object model with first-class support for all entities.

Although all our current experiments use single-agent interaction, Factorio inherently supports multi-player games in both cooperative and competitive scenarios. For instance, multiple agents could share a base, coordinating research and logistics, or they could spawn in distant regions and compete for finite resources (e.g., high-yield iron patches).

The unbounded nature of FLE provides a benchmark that will remain relevant even as progress in LLMs continues to advance. Unlike traditional benchmarks that are rapidly saturated by progress in AI research, FLE's exponentially scaling reward system and requirement for capabilities across multiple areas create a natural curricula that can meaningfully differentiate between increasingly performant models. Through our evaluation, we demonstrate that even state-of-the-art agents struggle with the coordination and optimization challenges inherent in simple automation and logistical tasks. The limitations we observed in spatial reasoning, long-term planning, and intelligent error correction highlight gaps in capabilities of foundation language models in novel environments.

Impact Statement

The Factorio Learning Environment provides a novel, open-source testbed for exploring advanced AI behavior in a complex, open-ended setting. It enables rigorous empirical study of the instrumental convergence hypothesis and other AI safety concerns. Researchers can measure how agents balance resource acquisition, territorial expansion, and defense, offering early indicators of convergent behaviors and potential risks. Moreover, Factorio's diverse automation challenges allow systematic comparisons of agent architectures and interventions, fostering reproducible research at scale. We hope this platform accelerates empirical safety investigations and strengthens the connection between theoretical predictions and real-world AI behavior.

References

OpenTTD: Transport simulation game. https://www.openttd.org/. Accessed: 2025-01-07.

Anthropic. Claude 3.5 sonnet: Enhanced intelligence and versatility, 2024. URL https://www.anthropic.com/news/claude-3-5-sonnet. 20241022-v2:0 version.

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Campbell, M., Hoane, A. J., and Hsu, F.-h. Deep Blue. *Artificial Intelligence*, 134(1–2):57–83, 2002. doi: 10.1016/S0004-3702(01)00129-1.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Clark, J. and Amodei, D. Faulty reward functions in the wild. *Internet: https://blog. openai. com/faulty-reward-functions*, 2016.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI, Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Guo, D., Yang, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Zhang, H., Ding, H., Xin, H., Gao, H., Li, H., Qu, H., Cai, J. L., Liang, J., Guo, J., Ni, J., Li, J., Wang, J., Chen, J., Chen, J., Yuan, J., Qiu, J., Li, J., Song, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Xu, L., Xia, L., Zhao, L., Wang, L., Zhang, L., Li, M., Wang, M., Zhang, M., Zhang, M., Tang, M., Li, M., Tian, N., Huang, P., Wang, P., Zhang, P., Wang, Q., Zhu, Q., Chen, Q., Du, Q., Chen, R. J., Jin, R. L., Ge, R., Zhang, R., Pan, R., Wang, R., Xu, R., Zhang, R., Chen, R., Li, S. S., Lu, S., Zhou, S., Chen, S., Wu, S., Ye, S., Ye, S., Ma, S., Wang, S., Zhou, S., Yu, S., Zhou, S., Pan, S., Wang, T., Yun, T., Pei, T., Sun, T., Xiao, W. L., Zeng, W., Zhao, W., An, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Li, X. Q., Jin, X., Wang, X., Bi, X., Liu, X., Wang, X., Shen, X., Chen, X., Zhang, X., Chen, X., Nie, X., Sun, X., Wang, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yu, X., Song, X., Shan, X., Zhou, X., Yang, X., Li, X., Su, X., Lin, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhu, Y. X., Zhang, Y., Xu, Y., Xu, Y., Huang, Y., Li, Y., Zhao, Y., Sun, Y., Li, Y., Wang, Y., Yu, Y., Zheng, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Tang, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Wu, Y., Ou, Y., Zhu, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Zha, Y., Xiong, Y., Ma, Y., Yan, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Wu, Z. F., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Huang, Z., Zhang, Z., Xie, Z., Zhang, Z., Hao, Z., Gou, Z., Ma, Z., Yan, Z., Shao, Z., Xu, Z., Wu, Z., Zhang, Z., Li, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Gao, Z., and Pan, Z. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

Denison, C., MacDiarmid, M., Barez, F., Duvenaud, D., Kravec, S., Marks, S., Schiefer, N., Soklaski, R., Tamkin, A., Kaplan, J., et al. Sycophancy to subterfuge: Investigating reward-tampering in large language models. *arXiv preprint arXiv:2406.10162*, 2024.

Fan, L., Xie, A., Shi, W., Sadat, A., Tan, X., Gong, W., Liang, J., and Huang, D.-A. MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2210.14168.

Guss, W. H., Houghton, B., Topin, N., Wang, P., Codel, C., Veloso, M., and Salakhutdinov, R. Minerl: A large-scale dataset of minecraft demonstrations. *arXiv preprint arXiv:1907.13440*, 2019.

Hausknecht, M., Fulda, N., Tien, E., Murdock, J., Yarnell, D., and Stone, P. Jericho: A new RL environment for Interactive Fiction. https://github.com/microsoft/jericho, 2019. Accessed: 2025-01-07.

Hendrycks, D., Basart, S., Kadavath, S., Mazeika, M., Arora, A., Guo, E., Burns, C., Puranik, S., He, H., Song, D., et al. Measuring coding challenge competence with apps. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

Kambhampati, S., Valmeekam, K., Guan, L., Stechly, K., Verma, M., Bhambri, S., Saldyt, L., and Murthy, A. Llms can't plan, but can help planning in llm-modulo frameworks. *arXiv preprint arXiv:2402.01817*, 2024.

Küttler, H., Nardelli, N., Miller, A., Raileanu, R., Selvatici, M., Grefenstette, E., and Rocktäschel, T. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684, 2020.

Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H., Gu, Y., Ding, H., Men, K., Yang, K., et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.

MetaAI. Llama 3.3, 2024. URL https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_3/. Accessed: 2025-01-25.

Naur, P. Programming as theory building. *Microprocessing and Microprogramming*, 15 (5):253–261, 1985. ISSN 0165-6074. doi: https://doi.org/10.1016/0165-6074(85)90032-8. URL https://www.sciencedirect.com/science/article/pii/0165607485900328.

OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., Bello, I., Berdine, J., Bernadett-Shapiro, G., Berner, C., Bogdonoff, L., Boiko, O., Boyd, M., Brakman, A.-L., Brockman, G., Brooks, T., Brundage, M., Button, K., Cai, T., Campbell, R., Cann, A., Carey, B., Carlson, C., Carmichael, R., Chan, B., Chang, C., Chantzis, F., Chen, D., Chen, S., Chen, R., Chen, J., Chen, M., Chess, B., Cho, C., Chu, C., Chung, H. W., Cummings, D., Currier, J., Dai, Y., Decareaux, C., Degry, T., Deutsch, N., Deville, D., Dhar, A., Dohan, D., Dowling, S., Dunning, S., Ecoffet, A., Eleti, A., Eloundou, T., Farhi, D., Fedus, L., Felix, N., Fishman, S. P., Forte, J., Fulford, I., Gao, L., Georges, E., Gibson, C., Goel, V., Gogineni, T., Goh, G., Gontijo-Lopes, R., Gordon, J., Grafstein, M., Gray, S., Greene, R., Gross, J., Gu, S. S., Guo, Y., Hallacy, C., Han, J., Harris, J., He, Y., Heaton, M., Heidecke, J., Hesse, C., Hickey, A., Hickey, W., Hoeschele, P., Houghton, B., Hsu, K., Hu, S., Hu, X., Huizinga, J., Jain, S., Jain, S., Jang, J., Jiang, A., Jiang, R., Jin, H., Jin, D., Jomoto, S., Jonn, B., Jun, H., Kaftan, T., Łukasz Kaiser, Kamali, A., Kanitscheider, I., Keskar, N. S., Khan, T., Kilpatrick, L., Kim, J. W., Kim, C., Kim, Y., Kirchner, J. H., Kiros, J., Knight, M., Kokotajlo, D., Łukasz Kondraciuk, Kondrich, A., Konstantinidis, A., Kosic, K., Krueger, G., Kuo, V., Lampe, M., Lan, I., Lee, T., Leike, J., Leung, J., Levy, D., Li, C. M., Lim, R., Lin, M., Lin, S., Litwin, M., Lopez, T., Lowe, R., Lue, P., Makanju, A., Malfacini, K., Manning, S., Markov, T., Markovski, Y., Martin, B., Mayer, K., Mayne, A., McGrew, B., McKinney, S. M., McLeavey, C., McMillan, P., McNeil, J., Medina, D., Mehta, A., Menick, J., Metz, L., Mishchenko, A., Mishkin, P., Monaco, V., Morikawa, E., Mossing, D., Mu, T., Murati, M., Murk, O., Mély, D., Nair, A., Nakano, R., Nayak, R., Neelakantan, A., Ngo, R., Noh, H., Ouyang, L., O'Keefe, C., Pachocki, J., Paino, A., Palermo, J., Pantuliano, A., Parascandolo, G., Parish, J., Parparita, E., Passos, A., Pavlov, M., Peng, A., Perelman, A., de Avila Belbute Peres, F., Petrov, M., de Oliveira Pinto, H. P., Michael, Pokorny, Pokrass, M., Pong, V. H., Powell, T., Power, A., Power, B., Proehl, E., Puri, R., Radford, A., Rae, J., Ramesh, A., Raymond, C., Real, F., Rimbach, K., Ross, C., Rotsted, B., Roussez, H., Ryder, N., Saltarelli, M., Sanders, T., Santurkar, S., Sastry, G., Schmidt, H., Schnurr, D., Schulman, J., Selsam, D., Sheppard, K., Sherbakov, T., Shieh, J., Shoker, S., Shyam, P., Sidor, S., Sigler, E., Simens, M., Sitkin, J., Slama, K., Sohl, I., Sokolowsky, B., Song, Y., Staudacher, N., Such, F. P., Summers, N., Sutskever, I., Tang, J., Tezak, N., Thompson, M. B., Tillet, P., Tootoonchian, A., Tseng, E., Tuggle, P., Turley, N., Tworek, J., Uribe, J. F. C., Vallone, A., Vijayvergiya, A., Voss, C., Wainwright, C., Wang, J. J., Wang, A., Wang, B., Ward, J., Wei, J., Weinmann, C., Welihinda, A., Welinder, P., Weng, J., Weng, L., Wiethoff, M., Willner, D., Winter, C., Wolrich, S., Wong, H., Workman, L., Wu, S., Wu, J., Wu, M., Xiao, K., Xu, T., Yoo, S., Yu, K., Yuan, Q., Zaremba, W., Zellers, R., Zhang, C., Zhang, M., Zhao, S., Zheng, T., Zhuang, J., Zhuk, W., and Zoph, B. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

Paglieri, D., Cupiał, B., Coward, S., Piterbarg, U., Wolczyk, M., Khan, A., Pignatelli, E., Kuciński, Ł., Pinto, L., Fergus, R., et al. Balrog: Benchmarking agentic llm and vlm reasoning on games. *arXiv preprint arXiv:2411.13543*, 2024.

Reid, K. N., Miralavy, I., Kelly, S., Banzhaf, W., and Gondro, C. The factory must grow: Automation in factorio, 2021. URL https://arxiv.org/abs/2102.04871.

Ruan, J., Chen, Y., Zhang, B., Xu, Z., Bao, T., Mao, H., Li, Z., Zeng, X., Zhao, R., et al. Tptu: Task planning and tool usage of large language model-based ai agents. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023.

Shridhar, M., Mottaghi, R., Kolve, E., and Gupta, A. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. https://github.com/alfworld/alfworld, 2020. Accessed: 2025-01-07.

Silver, D., Huang, A., Maddison, C. J., and et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. doi: 10.1038/nature16961.

Skalse, J., Howe, N., Krasheninnikov, D., and Krueger, D. Defining and characterizing reward gaming. *Advances in Neural Information Processing Systems*, 35:9460–9471, 2022.

Team, G., Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., Silver, D., Johnson, M., Antonoglou, I., Schrittwieser, J., Glaese, A., Chen, J., Pitler, E., Lillicrap, T., Lazaridou, A., Firat, O., Molloy, J., Isard, M., Barham, P. R., Hennigan, T., Lee, B., Viola, F., Reynolds, M., Xu, Y., Doherty, R., Collins, E., Meyer, C., Rutherford, E., Moreira, E., Ayoub, K., Goel, M., Krawczyk, J., Du, C., Chi, E., Cheng, H.-T., Ni, E., Shah, P., Kane, P., Chan, B., Faruqui, M., Severyn, A., Lin, H., Li, Y., Cheng, Y., Ittycheriah, A., Mahdieh, M., Chen, M., Sun, P., Tran, D., Bagri, S., Lakshminarayanan, B., Liu, J., Orban, A., Güra, F., Zhou, H., Song, X., Boffy, A., Ganapathy, H., Zheng, S., Choe, H., Ágoston Weisz, Zhu, T., Lu, Y., Gopal, S., Kahn, J., Kula, M., Pitman, J., Shah, R., Taropa, E., Merey, M. A., Baeuml, M., Chen, Z., Shafey, L. E., Zhang, Y., Sercinoglu, O., Tucker, G., Piqueras, E., Krikun, M., Barr, I., Savinov, N., Danihelka, I., Roelofs, B., White, A., Andreassen, A., von Glehn, T., Yagati, L., Kazemi, M., Gonzalez, L., Khalman, M., Sygnowski, J., Frechette, A., Smith, C., Culp, L., Proleev, L., Luan, Y., Chen, X., Lottes, J., Schucher, N., Lebron, F., Rrustemi, A., Clay, N., Crone, P., Kocisky, T., Zhao, J., Perz, B., Yu, D., Howard, H., Bloniarz, A., Rae, J. W., Lu, H., Sifre, L., Maggioni, M., Alcober, F., Garrette, D., Barnes, M., Thakoor, S., Austin, J., Barth-Maron, G., Wong, W., Joshi, R., Chaabouni, R., Fatiha, D., Ahuja, A., Tomar, G. S., Senter, E., Chadwick, M., Kornakov, I., Attaluri, N., Iturrate, I., Liu, R., Li, Y., Cogan, S., Chen, J., Jia, C., Gu, C., Zhang, Q., Grimstad, J., Hartman, A. J., Garcia, X., Pillai, T. S., Devlin, J., Laskin, M., de Las Casas, D., Valter, D., Tao, C., Blanco, L., Badia, A. P., Reitter, D., Chen, M., Brennan, J., Rivera, C., Brin, S., Iqbal, S., Surita, G., Labanowski, J., Rao, A., Winkler, S., Parisotto, E., Gu, Y., Olszewska, K., Addanki, R., Miech, A., Louis, A., Teplyashin, D., Brown, G., Catt, E., Balaguer, J., Xiang, J., Wang, P., Ashwood, Z., Briukhov, A., Webson, A., Ganapathy, S., Sanghavi, S., Kannan, A., Chang, M.-W., Stjerngren, A., Djolonga, J., Sun, Y., Bapna, A., Aitchison, M., Pejman, P., Michalewski, H., Yu, T., Wang, C., Love, J., Ahn, J., Bloxwich, D., Han, K., Humphreys, P., Sellam, T., Bradbury, J., Godbole, V., Samangooei, S., Damoc, B., Kaskasoli, A., Arnold, S. M. R., Vasudevan, V., Agrawal, S., Riesa, J., Lepikhin, D., Tanburn, R., Srinivasan, S., Lim, H., Hodkinson, S., Shyam, P., Ferret, J., Hand, S., Garg, A., Paine, T. L., Li, J., Li, Y., Giang, M., Neitz, A., Abbas, Z., York, S., Reid, M., Cole, E., Chowdhery, A., Das, D., Rogozińska, D., Nikolaev, V., Sprechmann, P., Nado, Z., Zilka, L., Prost, F., He, L., Monteiro, M., Mishra, G., Welty, C., Newlan, J., Jia, D., Allamanis, M., Hu, C. H., de Liedekerke, R., Gilmer, J., Saroufim, C., Rijhwani, S., Hou, S., Shrivastava, D., Baddepudi, A., Goldin, A., Ozturel, A., Cassirer, A., Xu, Y., Sohn, D., Sachan, D., Amplayo, R. K., Swanson, C., Petrova, D., Narayan, S., Guez, A., Brahma, S., Landon, J., Patel, M., Zhao, R., Villela, K., Wang, L., Jia, W., Rahtz, M., Giménez, M., Yeung, L., Keeling, J., Georgiev, P., Mincu, D., Wu, B., Haykal, S., Saputro, R., Vodrahalli, K., Qin, J., Cankara, Z., Sharma, A., Fernando, N., Hawkins, W., Neyshabur, B., Kim, S., Hutter, A., Agrawal, P., Castro-Ros, A., van den Driessche, G., Wang, T., Yang, F., yiin Chang, S., Komarek, P., McIlroy, R., Lučić, M., Zhang, G., Farhan, W., Sharman, M., Natsev, P., Michel, P., Bansal, Y., Qiao, S., Cao, K., Shakeri, S., Butterfield, C., Chung, J., Rubenstein, P. K., Agrawal, S., Mensch, A., Soparkar, K., Lenc, K., Chung, T., Pope, A., Maggiore, L., Kay, J., Jhakra, P., Wang, S., Maynez, J., Phuong, M., Tobin, T., Tacchetti, A., Trebacz, M., Robinson, K., Katariya, Y., Riedel, S., Bailey, P., Xiao, K., Ghelani, N., Aroyo, L., Slone, A., Houlsby, N., Xiong, X., Yang, Z., Gribovskaya, E., Adler, J., Wirth, M., Lee, L., Li, M., Kagohara, T., Pavagadhi, J., Bridgers, S., Bortsova, A., Ghemawat, S., Ahmed, Z., Liu, T., Powell, R., Bolina, V., Iinuma, M., Zablotskaia, P., Besley, J., Chung, D.-W., Dozat, T., Comanescu, R., Si, X., Greer, J., Su, G., Polacek, M., Kaufman, R. L., Tokumine, S., Hu, H., Buchatskaya, E., Miao, Y., Elhawaty, M., Siddhant, A., Tomasev, N., Xing, J., Greer, C., Miller, H., Ashraf, S., Roy, A., Zhang, Z., Ma, A., Filos, A., Besta, M., Blevins, R., Klimenko, T., Yeh, C.-K., Changpinyo, S., Mu, J., Chang, O., Pajarskas, M., Muir, C., Cohen, V., Lan, C. L., Haridasan, K., Marathe, A., Hansen, S., Douglas, S., Samuel, R., Wang, M., Austin, S., Lan, C., Jiang, J., Chiu, J., Lorenzo, J. A., Sjösund, L. L., Cevey, S., Gleicher, Z., Avrahami, T., Boral, A., Srinivasan, H., Selo, V., May, R., Aisopos, K., Hussenot, L., Soares, L. B., Baumli, K., Chang, M. B., Recasens, A., Caine, B., Pritzel, A., Pavetic, F., Pardo, F., Gergely, A., Frye, J., Ramasesh, V., Horgan, D., Badola, K., Kassner, N., Roy, S., Dyer, E., Campos, V. C., Tomala, A., Tang, Y., Badawy, D. E., White, E., Mustafa, B., Lang, O., Jindal, A., Vikram, S., Gong, Z., Caelles, S., Hemsley, R., Thornton, G., Feng, F., Stokowiec, W., Zheng, C., Thacker, P., Çağlar Ünlü, Zhang, Z., Saleh, M., Svensson, J., Bileschi, M., Patil, P., Anand, A., Ring, R., Tsihlas, K., Vezer, A., Selvi, M., Shevlane, T., Rodriguez, M., Kwiatkowski, T., Daruki, S., Rong, K., Dafoe, A., FitzGerald, N., Gu-Lemberg, K., Khan, M., Hendricks, L. A., Pellat, M., Feinberg, V., Cobon-Kerr, J., Sainath, T., Rauh, M., Hashemi, S. H., Ives, R., Hasson, Y., Noland, E., Cao, Y., Byrd, N., Hou, L., Wang, Q., Sottiaux, T., Paganini, M., Lespiau, J.-B., Moufarek, A., Hassan, S., Shivakumar, K., van Amersfoort, J., Mandhane, A., Joshi, P., Goyal, A., Tung, M., Brock, A., Sheahan, H., Misra, V., Li, C., Rakićević, N., Dehghani, M., Liu, F., Mittal, S., Oh, J., Noury, S., Sezener, E., Huot, F., Lamm, M., Cao, N. D., Chen, C., Mudgal, S., Stella, R., Brooks, K., Vasudevan, G., Liu, C., Chain, M., Melinkeri, N., Cohen, A., Wang, V., Seymore, K., Zubkov, S., Goel, R., Yue, S., Krishnakumaran, S., Albert, B., Hurley, N., Sano, M., Mohananey, A., Joughin, J., Filonov, E., Kepa, T., Eldawy, Y., Lim, J., Rishi, R., Badiezadegan, S., Bos, T., Chang, J., Jain, S., Padmanabhan, S. G. S., Puttagunta, S., Krishna, K., Baker, L., Kalb, N., Bedapudi, V., Kurzrok, A., Lei, S., Yu, A., Litvin, O., Zhou, X., Wu, Z., Sobell, S., Siciliano, A., Papir, A., Neale, R., Bragagnolo, J., Toor, T., Chen, T., Anklin, V., Wang, F., Feng, R., Gholami, M., Ling, K., Liu, L., Walter, J., Moghaddam, H., Kishore, A., Adamek, J., Mercado, T., Mallinson, J., Wandekar, S., Cagle, S., Ofek, E., Garrido, G., Lombriser, C., Mukha, M., Sun, B., Mohammad, H. R., Matak, J., Qian, Y., Peswani, V., Janus, P., Yuan, Q., Schelin, L., David, O., Garg, A., He, Y., Duzhyi, O., Älgmyr, A., Lottaz, T., Li, Q., Yadav, V., Xu, L., Chinien, A., Shivanna, R., Chuklin, A., Li, J., Spadine, C., Wolfe, T., Mohamed, K., Das, S., Dai, Z., He, K., von Dincklage, D., Upadhyay, S., Maurya, A., Chi, L., Krause, S., Salama, K., Rabinovitch, P. G., M, P. K. R., Selvan, A., Dektiarev, M., Ghiasi, G., Guven, E., Gupta, H., Liu, B., Sharma, D., Shtacher, I. H., Paul, S., Akerlund, O., Aubet, F.-X., Huang, T., Zhu, C., Zhu, E., Teixeira, E., Fritze, M., Bertolini, F., Marinescu, L.-E., Bölle, M., Paulus, D., Gupta, K., Latkar, T., Chang, M., Sanders, J., Wilson, R., Wu, X., Tan, Y.-X., Thiet, L. N., Doshi, T., Lall, S., Mishra, S., Chen, W., Luong, T., Benjamin, S., Lee, J., Andrejczuk, E., Rabiej, D., Ranjan, V., Styrc, K., Yin, P., Simon, J., Harriott, M. R., Bansal, M., Robsky, A., Bacon, G., Greene, D., Mirylenka, D., Zhou, C., Sarvana, O., Goyal, A., Andermatt, S., Siegler, P., Horn, B., Israel, A., Pongetti, F., Chen, C.-W. L., Selvatici, M., Silva, P., Wang, K., Tolins, J., Guu, K., Yogev, R., Cai, X., Agostini, A., Shah, M., Nguyen, H., Donnaile, N. O., Pereira, S., Friso, L., Stambler, A., Kurzrok, A., Kuang, C., Romanikhin, Y., Geller, M., Yan, Z., Jang, K., Lee, C.-C., Fica, W., Malmi, E., Tan, Q., Banica, D., Balle, D., Pham, R., Huang, Y., Avram, D., Shi, H., Singh, J., Hidey, C., Ahuja, N., Saxena, P., Dooley, D., Potharaju, S. P., O'Neill, E., Gokulchandran, A., Foley, R., Zhao, K., Dusenberry, M., Liu, Y., Mehta, P., Kotikalapudi, R., Safranek-Shrader, C., Goodman, A., Kessinger, J., Globen, E., Kolhar, P., Gorgolewski, C., Ibrahim, A., Song, Y., Eichenbaum, A., Brovelli, T., Potluri, S., Lahoti, P., Baetu, C., Ghorbani, A., Chen, C., Crawford, A., Pal, S., Sridhar, M., Gurita, P., Mujika, A., Petrovski, I., Cedoz, P.-L., Li, C., Chen, S., Santo, N. D., Goyal, S., Punjabi, J., Kappaganthu, K., Kwak, C., LV, P., Velury, S., Choudhury, H., Hall, J., Shah, P., Figueira, R., Thomas, M., Lu, M., Zhou, T., Kumar, C., Jurdi, T., Chikkerur, S., Ma, Y., Yu, A., Kwak, S., Ähdel, V., Rajayogam, S., Choma, T., Liu, F., Barua, A., Ji, C., Park, J. H., Hellendoorn, V., Bailey, A., Bilal, T., Zhou, H., Khatir, M., Sutton, C., Rzadkowski, W., Macintosh, F., Shagin, K., Medina, P., Liang, C., Zhou, J., Shah, P., Bi, Y., Dankovics, A., Banga, S., Lehmann, S., Bredesen, M., Lin, Z., Hoffmann, J. E., Lai, J., Chung, R., Yang, K., Balani, N., Bražinskas, A., Sozanschi, A., Hayes, M., Alcalde, H. F., Makarov, P., Chen, W., Stella, A., Snijders, L., Mandl, M., Kärrman, A., Nowak, P., Wu, X., Dyck, A., Vaidyanathan, K., R, R., Mallet, J., Rudominer, M., Johnston, E., Mittal, S., Udathu, A., Christensen, J., Verma, V., Irving, Z., Santucci, A., Elsayed, G., Davoodi, E., Georgiev, M., Tenney, I., Hua, N., Cideron, G., Leurent, E., Alnahlawi, M., Georgescu, I., Wei, N., Zheng, I., Scandinaro, D., Jiang, H., Snoek, J., Sundararajan, M., Wang, X., Ontiveros, Z., Karo, I., Cole, J., Rajashekhar, V., Tumeh, L., Ben-David, E., Jain, R., Uesato, J., Datta, R., Bunyan, O., Wu, S., Zhang, J., Stanczyk, P., Zhang, Y., Steiner, D., Naskar, S., Azzam, M., Johnson, M., Paszke, A., Chiu, C.-C., Elias, J. S., Mohiuddin, A., Muhammad, F., Miao, J., Lee, A., Vieillard, N., Park, J., Zhang, J., Stanway, J., Garmon, D., Karmarkar, A., Dong, Z., Lee, J., Kumar, A., Zhou, L., Evens, J., Isaac, W., Irving, G., Loper, E., Fink, M., Arkatkar, I., Chen, N., Shafran, I., Petrychenko, I., Chen, Z., Jia, J., Levskaya, A., Zhu, Z., Grabowski, P., Mao, Y., Magni, A., Yao, K., Snaider, J., Casagrande, N., Palmer, E., Suganthan, P., Castaño, A., Giannoumis, I., Kim, W., Rybiński, M., Sreevatsa, A., Prendki, J., Soergel, D., Goedeckemeyer, A., Gierke, W., Jafari, M., Gaba, M., Wiesner, J., Wright, D. G., Wei, Y., Vashisht, H., Kulizhskaya, Y., Hoover, J., Le, M., Li, L., Iwuanyanwu, C., Liu, L., Ramirez, K., Khorlin, A., Cui, A., LIN, T., Wu, M., Aguilar, R., Pallo, K., Chakladar, A., Perng, G., Abellan, E. A., Zhang, M., Dasgupta, I., Kushman, N., Penchev, I., Repina, A., Wu, X., van der Weide, T., Ponnapalli, P., Kaplan, C., Simsa, J., Li, S., Dousse, O., Yang, F., Piper, J., Ie, N., Pasumarthi, R., Lintz, N., Vijayakumar, A., Andor, D., Valenzuela, P., Lui, M., Paduraru, C., Peng, D., Lee, K., Zhang, S., Greene, S., Nguyen, D. D., Kurylowicz, P., Hardin, C., Dixon, L., Janzer, L., Choo, K., Feng, Z., Zhang, B., Singhal, A., Du, D., McKinnon, D., Antropova, N., Bolukbasi, T., Keller, O., Reid, D., Finchelstein, D., Raad, M. A., Crocker, R., Hawkins, P., Dadashi, R., Gaffney, C., Franko, K., Bulanova, A., Leblond, R., Chung, S., Askham, H., Cobo, L. C., Xu, K., Fischer, F., Xu, J., Sorokin, C., Alberti, C., Lin, C.-C., Evans, C., Dimitriev, A., Forbes, H., Banarse, D., Tung, Z., Omernick, M., Bishop, C., Sterneck, R., Jain, R., Xia, J., Amid, E., Piccinno, F., Wang, X., Banzal, P., Mankowitz, D. J., Polozov, A., Krakovna, V., Brown, S., Bateni, M., Duan, D., Firoiu, V., Thotakuri, M., Natan, T., Geist, M., tan Girgin, S., Li, H., Ye, J., Roval, O., Tojo, R., Kwong, M., Lee-Thorp, J., Yew, C., Sinopalnikov, D., Ramos, S., Mellor, J., Sharma, A., Wu, K., Miller, D., Sonnerat, N., Vnukov, D., Greig, R., Beattie, J., Caveness, E., Bai, L., Eisenschlos, J., Korchemniy, A., Tsai, T., Jasarevic, M., Kong, W., Dao, P., Zheng, Z., Liu, F., Yang, F., Zhu, R., Teh, T. H., Sanmiya, J., Gladchenko, E., Trdin, N., Toyama, D., Rosen, E., Tavakkol, S., Xue, L., Elkind, C., Woodman, O., Carpenter, J., Papamakarios, G., Kemp, R., Kafle, S., Grunina, T., Sinha, R., Talbert, A., Wu, D., Owusu-Afriyie, D., Du, C., Thornton, C., Pont-Tuset, J., Narayana, P., Li, J., Fatehi, S., Wieting, J., Ajmeri, O., Uria, B., Ko, Y., Knight, L., Héliou, A., Niu, N., Gu, S., Pang, C., Li, Y., Levine, N., Stolovich, A., Santamaria-Fernandez, R., Goenka, S., Yustalim, W., Strudel, R., Elqursh, A., Deck, C., Lee, H., Li, Z., Levin, K., Hoffmann, R., Holtmann-Rice, D., Bachem, O., Arora, S., Koh, C., Yeganeh, S. H., Põder, S., Tariq, M., Sun, Y., Ionita, L., Seyedhosseini, M., Tafti, P., Liu, Z., Gulati, A., Liu, J., Ye, X., Chrzaszcz, B., Wang, L., Sethi, N., Li, T., Brown, B., Singh, S., Fan, W., Parisi, A., Stanton, J., Koverkathu, V., Choquette-Choo, C. A., Li, Y., Lu, T., Ittycheriah, A., Shroff, P., Varadarajan, M., Bahargam, S., Willoughby, R., Gaddy, D., Desjardins, G., Cornero, M., Robenek, B., Mittal, B., Albrecht, B., Shenoy, A., Moiseev, F., Jacobsson, H., Ghaffarkhah, A., Rivière, M., Walton, A., Crepy, C., Parrish, A., Zhou, Z., Farabet, C., Radebaugh, C., Srinivasan, P., van der Salm, C., Fidjeland, A., Scellato, S., Latorre-Chimoto, E., Klimczak-Plucińska, H., Bridson, D., de Cesare, D., Hudson, T., Mendolicchio, P., Walker, L., Morris, A., Mauger, M., Guseynov, A., Reid, A., Odoom, S., Loher, L., Cotruta, V., Yenugula, M., Grewe, D., Petrushkina, A., Duerig, T., Sanchez, A., Yadlowsky, S., Shen, A., Globerson, A., Webb, L., Dua, S., Li, D., Bhupatiraju, S., Hurt, D., Qureshi, H., Agarwal, A., Shani, T., Eyal, M., Khare, A., Belle, S. R., Wang, L., Tekur, C., Kale, M. S., Wei, J., Sang, R., Saeta, B., Liechty, T., Sun, Y., Zhao, Y., Lee, S., Nayak, P., Fritz, D., Vuyyuru, M. R., Aslanides, J., Vyas, N., Wicke, M., Ma, X., Eltyshev, E., Martin, N., Cate, H., Manyika, J., Amiri, K., Kim, Y., Xiong, X., Kang, K., Luisier, F., Tripuraneni, N., Madras, D., Guo, M., Waters, A., Wang, O., Ainslie, J., Baldridge, J., Zhang, H., Pruthi, G., Bauer, J., Yang, F., Mansour, R., Gelman, J., Xu, Y., Polovets, G., Liu, J., Cai, H., Chen, W., Sheng, X., Xue, E., Ozair, S., Angermueller, C., Li, X., Sinha, A., Wang, W., Wiesinger, J., Koukoumidis, E., Tian, Y., Iyer, A., Gurumurthy, M., Goldenson, M., Shah, P., Blake, M., Yu, H., Urbanowicz, A., Palomaki, J., Fernando, C., Durden, K., Mehta, H., Momchev, N., Rahimtoroghi, E., Georgaki, M., Raul, A., Ruder, S., Redshaw, M., Lee, J., Zhou, D., Jalan, K., Li, D., Hechtman, B., Schuh, P., Nasr, M., Milan, K., Mikulik, V., Franco, J., Green, T., Nguyen, N., Kelley, J., Mahendru, A., Hu, A., Howland, J., Vargas, B., Hui, J., Bansal, K., Rao, V., Ghiya, R., Wang, E., Ye, K., Sarr, J. M., Preston, M. M., Elish, M., Li, S., Kaku, A., Gupta, J., Pasupat, I., Juan, D.-C., Someswar, M., M., T., Chen, X., Amini, A., Fabrikant, A., Chu, E., Dong, X., Muthal, A., Buthpitiya, S., Jauhari, S., Hua, N., Khandelwal, U., Hitron, A., Ren, J., Rinaldi, L., Drath, S., Dabush, A., Jiang, N.-J., Godhia, H., Sachs, U., Chen, A., Fan, Y., Taitelbaum, H., Noga, H., Dai, Z., Wang, J., Liang, C., Hamer, J., Ferng, C.-S., Elkind, C., Atias, A., Lee, P., Listík, V., Carlen, M., van de Kerkhof, J., Pikus, M., Zaher, K., Müller, P., Zykova, S., Stefanec, R., Gatsko, V., Hirnschall, C., Sethi, A., Xu, X. F., Ahuja, C., Tsai, B., Stefanoiu, A., Feng, B., Dhandhania, K., Katyal, M., Gupta, A., Parulekar, A., Pitta, D., Zhao, J., Bhatia, V., Bhavnani, Y., Alhadlaq, O., Li, X., Danenberg, P., Tu, D., Pine, A., Filippova, V., Ghosh, A., Limonchik, B., Urala, B., Lanka, C. K., Clive, D., Sun, Y., Li, E., Wu, H., Hongtongsak, K., Li, I., Thakkar, K., Omarov, K., Majmundar, K., Alverson, M., Kucharski, M., Patel, M., Jain, M., Zabelin, M., Pelagatti, P., Kohli, R., Kumar, S., Kim, J., Sankar, S., Shah, V., Ramachandruni, L., Zeng, X., Bariach, B., Weidinger, L., Vu, T., Andreev, A., He, A., Hui, K., Kashem, S., Subramanya, A., Hsiao, S., Hassabis, D., Kavukcuoglu, K., Sadovsky, A., Le, Q., Strohman, T., Wu, Y., Petrov, S., Dean, J., and Vinyals, O. Gemini: A family of highly capable multimodal models, 2024. URL https://arxiv.org/abs/2312.11805.

Xing, M., Zhang, R., Xue, H., Chen, Q., Yang, F., and Xiao, Z. Understanding the weakness of large language model agents within a complex android environment. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 6061–6072, 2024.

Yamada, Y., Bao, Y., Lampinen, A. K., Kasai, J., and Yildirim, I. Evaluating spatial understanding of large language models. *arXiv preprint arXiv:2310.14540*, 2023.

Yang, H., Yue, S., and He, Y. Auto-gpt for online decision making: Benchmarks and additional opinions. *arXiv preprint arXiv:2306.02224*, 2023.

A. Factorio's Economic System

For each item i in the game, its value $V(i)$ is computed as:

$$V(i) = \min_{r \in R_i} \left(\Big(\sum_{j \in I_r} V(j) c_{j,r} \Big) \alpha(|I_r|) + E(r, C_r) \right) \tag{1}$$

Where:

R_i is the set of recipes that can produce item i
I_r is the set of ingredients for recipe r
$c_{j,r}$ is the amount of ingredient j needed in recipe r
$\alpha(n)$ is the complexity multiplier: $\alpha(n) = \beta^{n-2}$ where $\beta \approx 1.025$ is the ingredient exponent
$E(r, C_r)$ is the energy cost function: $E(r, C_r) = \ln(e_r + 1)\sqrt{C_r}$ where:
e_r is the energy required for recipe r
C_r is the base cost of ingredients

The system is initialized with seed prices for raw resources:

- Iron ore: 3.1
- Copper ore: 3.6
- Coal: 3.0
- Stone: 2.4
- Uranium ore: 8.2
- Crude oil: 0.2

The complexity multiplier $\alpha(n)$ grows exponentially with the number of ingredients, incentivizing the creation of more sophisticated items which require geometrically increasing raw resources to manufacture. The energy cost term $E(r, C_r)$ scales sub-linearly through the square root, preventing energy from dominating at high scales.

The final PS for a force (player or team) at time t is:

$$PS(t) = \sum_{i \in Items} V(i)(P_i(t) - C_i(t)) \tag{2}$$

Where:

$P_i(t)$ is the total production of item i up to time t
$C_i(t)$ is the total consumption of item i up to time t
$Items$ is the set of all possible items and fluids

Note: While the energy cost scaling in Factorio's economic system is designed for gameplay progression rather than physical realism, it effectively serves our purpose of rewarding increasingly sophisticated automation.

B. Experimental Costs

Model	Input Tokens	Output Tokens	Total Tokens	Cost (USD)
Claude-3.5-Sonnet	1,413,403,475	23,340,352	1,436,743,827	4,590.32
DeepSeek-Chat	762,901,100	10,399,299	773,300,399	927.96
Gemini-2.0-Flash	1,686,890,489	87,278,090	1,774,168,579	203.60
GPT-4o	1,061,860,012	19,739,272	1,081,599,284	2,852.04
GPT-4o-Mini	1,404,986,049	28,087,751	1,433,073,800	227.60
Llama-3.3-70B-Instruct-Turbo	447,307,196	4,945,831	452,253,027	55.16
Total	6,777,348,321	173,790,595	6,951,138,916	8,856.68

Table 3. Token Usage and Cost Comparison across Models in Open-play. The total cost was 8,856.68 USD.

Model	Input Tokens	Output Tokens	Total Tokens	Cost (USD)
Claude-3.5-Sonnet	293,433,245	5,763,345	299,196,590	966.75
DeepSeek-Chat	199,291,079	4,117,889	203,408,968	244.09
Gemini-2.0-Flash	220,466,926	7,170,513	227,637,439	24.91
GPT-4o	231,389,195	3,921,987	235,311,182	617.69
GPT-4o-Mini	145,113,122	3286602	148,399,912	23.74
Llama-3.3-70B-Instruct-Turbo	124,239,159	1,749,449	125,988,608	15.43
Total	1,213,932,726	26,009,785	1,239,942,699	1,892.61

Table 4. Token Usage and Cost Comparison across Models in Lab-play. The total cost was 1,892.61 USD.

C. Benchmark Results

Operation	Ops/Min	Ops/Sec	Duration
place_entity_next_to	2,578	43	0.42
place_entity	12,058	201	0.50
move_to	8,650	144	0.69
harvest_resource	16,599	277	0.36
craft_item	16,875	281	0.36
connect_entities	1,665	28	3.21
rotate_entity	12,281	205	0.49
insert_item	13,044	217	0.46
extract_item	17,167	286	0.35
inspect_inventory	17,036	284	0.35
get_resource_patch	7,004	117	0.86
Total	7,513	125	8.04

Figure 7. Factorio Client + Factorio Server + FLE API

Operation	Ops/Min	Ops/Sec	Duration
place_entity_next_to	4,857	81	0.22
place_entity	22,333	372	0.27
move_to	16,006	267	0.37
harvest_resource	32,727	545	0.18
craft_item	36,224	604	0.17
connect_entities	2,926	49	1.83
rotate_entity	23,467	391	0.26
insert_item	25,154	419	0.24
extract_item	32,997	550	0.18
inspect_inventory	28,402	473	0.21
get_resource_patch	8,736	146	0.69
Total	13,095	218	4.61

Figure 8. Factorio Server + FLE API

Operation	Ops/Min	Ops/Sec	Duration
place_entity_next_to	5,070	84	1.18
place_entity	5,239	87	1.15
move_to	4,980	83	1.20
harvest_resource	3,247	54	1.85
craft_item	5,854	98	1.02
connect_entities	2,150	36	2.79
rotate_entity	5,370	90	1.12
insert_item	5,066	84	1.18
extract_item	5,449	91	1.10
inspect_inventory	5,639	94	1.06
get_resource_patch	2,479	41	2.42
Total	4,104	68	16.08

Figure 9. Interpreter + Factorio Server + FLE API

Operation	Ops/Min	Ops/Sec	Duration
place_entity_next_to	4,715	79	1.27
place_entity	4,774	80	1.26
move_to	4,006	67	1.50
harvest_resource	3,595	60	1.67
craft_item	4,985	83	1.20
connect_entities	1,497	25	4.01
rotate_entity	4,915	82	1.22
insert_item	5,047	84	1.19
extract_item	4,743	79	1.26
inspect_inventory	4,838	81	1.24
get_resource_patch	2,593	43	2.31
Total	3,639	61	18.14

Figure 10. Interpreter + Factorio Client + Factorio Server + FLE API

Figure 11. Performance Comparison of Different FLE Configurations

D. API Design

The environment's design prioritizes clarity and robustness over mechanical execution speed, reflecting Factorio's emphasis on planning and design rather than rapid action sequences. This aligns well with language models' strengths in systematic reasoning and program synthesis while providing rich opportunities for learning increasingly sophisticated automation strategies.

D.1. Action and Observation

We designed the environment's action space as a typed Python programming interface aligned with LLMs' capabilities for symbolic reasoning and program synthesis. Rather than requiring agents to learn low-level motor controls or pixel-level manipulation, our environment enables them to generate, reason about, and debug code while handling the complex requirements of factory automation. Unlike traditional reinforcement learning environments where agents must map state observations to discrete actions, our approach allows composition of rich programs that both gather information and modify game state, mirroring how LLMs naturally process and generate code.

From a theoretical perspective, we draw on Naur's view of programming as a continual process of "theory building" (Naur, 1985). In this view, the generated code represents an explicit, evolving model of how the agent believes the environment behaves. Each new function, variable, or data structure encodes the agent's current hypotheses about causal relationships (e.g., how ore is processed, or how machines are connected) and constraints (e.g., resource limitations or layout restrictions). When the agent executes its code and observes the resulting changes in the game state, it obtains evidence that either affirms or contradicts these hypotheses. Code revisions then become part of a self-correcting feedback loop in which the agent refines its theory to better match reality. This iterative process of writing, executing, and revising code reflects the core idea of treating programming as theory-building in a dynamic environment.

More formally, let us define the action space as a context-sensitive program synthesis task. Let Σ be the set of all valid Python programs, where each program $p \in \Sigma$ is a sequence of statements $\langle s_1, s_2, ..., s_n \rangle$. Each statement s is either a method invocation or a variable declaration:

$$s := m \mid (v := m) \text{ where:} \tag{3}$$

- $m = (f, args, ret)$ is a method invocation
- $f \in F$ is a function identifier from our API method set F
- $args = (a_1, a_2, ..., a_k)$ is a sequence of typed arguments where $a_i \in T_i$
- $ret \in T \cup \{\perp\}$ is the return type (possibly undefined)
- v is a variable identifier that enters the namespace context C

The type system T is defined by the algebraic data types:

$$\begin{aligned} T &:= \text{Prototype} \mid \text{Entity} \mid \text{Direction} \mid \text{Recipe} \mid \ldots \\ \text{Entity} &:= \text{AssemblingMachine} \mid \text{Inserter} \mid \text{Chest} \mid \ldots \\ \text{Position} &:= (x : \mathbb{R}, y : \mathbb{R}) \end{aligned}$$

Method execution transforms only the game state:

$$\text{exec} : M \times G \rightarrow (G' \times T) \tag{4}$$

While namespace context C is modified only through variable declarations:

$$\text{declare} : V \times T \times C \rightarrow C' \tag{5}$$

where M is the set of all valid method invocations, G is the set of all possible game states, V is the set of valid variable identifiers, T is the set of possible return types, and C is the set of all possible namespace contexts.

The action space consists of 23 core API methods that form a domain-specific language for factory automation, roughly categorised as follows:

Pure Queries $(Q : G \rightarrow T)$

- `get_entities`: Find entities matching a prototype
- `production_stats`: Get factory output metrics
- `nearest`: Find the nearest named entity to the player
- `inspect_inventory`: Retrieve the inventory of an entity

State Modifications $(M : G \rightarrow G' \times T)$:

- `place_entity`: Create buildings and machines
- `rotate_entity`: Change entity orientation
- `craft_item`: Manually create an item from ingredients
- `set_recipe`: Configure production recipes
- `connect_entities`: Connect two entities or positions with belts, pipes or power

Resource Management $(R : G \rightarrow G' \times T)$:

- `insert_item`: Add items to containers
- `harvest_resource`: Gather raw materials
- `extract_item`: Move an item from an entity into the inventory

The namespace context C maintains references to entities, positions, and other values through variable declarations, enabling agents to track and reuse factory components. This separation between method execution and namespace modification supports compositional factory design while maintaining clear semantics about state changes.

```python
# Pure query - affects neither G nor C
recipe = get_prototype_recipe(Prototype.IronGearWheel)
# Effects on game state G only (G -> G' x T)
success = set_entity_recipe(assembler, recipe)
# Namespace context C is modified only through assignments
assembler = place_entity_next_to(                # Method: G -> G' x T_Entity
    entity=Prototype.AssemblingMachine2,         # Variable declaration: C -> C'
    reference_position=inserter.position,        # Reference from C
    direction=Direction.RIGHT,
    spacing=1
)
# Runtime assertions can verify both game state and namespace
assert isinstance(assembler, AssemblingMachine)
assert get_entity(
    Prototype.AssemblingMachine2,
    assembler.position
) is not None
```

Figure 12. Example code showing state transitions.

A distinctive feature of our action space is the ability for agents to make runtime assertions about their beliefs regarding the game state. These assertions provide piece-meal feedback about the game state, allowing agents to debug discrepancies between their intended actions and actual outcomes. When assertions fail, agents can gather additional information through observation actions to update their beliefs and modify their approach. This creates a natural debugging loop that mirrors human programming practices.

Not all actions are available in every game state. For instance, `insert_item` requires both a valid item prototype and a target entity with sufficient inventory space. To help agents reason about action validity, tools like `can_place_entity` provide explicit validation capabilities. Most tools return boolean success indicators or meaningful result values, allowing agents to adapt their strategies based on action outcomes. Semantic errors (such as trying to insert a position into an inventory) result in exception containing a specific failure message and stack trace being thrown.

We impose no artificial rate limiting on API calls, as the emphasis is on the logical correctness of the generated programs rather than mechanical execution speed. This reflects the nature of Factorio as a game of planning and design. However, the `sleep` method allows agents to implement deliberate timing when necessary for complex automation sequences, such as waiting for ore to be smelted into plate for downstream steps.

An API-based action space supports natural composition of atomic actions into complex factory designs through its strongly-typed interface. Information-gathering actions enable deliberate planning and strategic decision-making, while the action space maps cleanly to natural language descriptions of factory building steps. The persistent namespace and type system enable compositional reasoning about factory designs over a long horizon, with rich type information helping language models understand entity relationships and constraints.

This cycle creates a natural debugging loop that mirrors human programming practices, allowing agents to iteratively develop and test their automation strategies.

Partial Observability System Unlike many reinforcement learning environments that provide complete state observations, FLE implements true partial observability through a snapshot-based system:

- **State References**: When an agent queries the environment (e.g., searching for nearby resources or machines), it receives a snapshot of the current state rather than a live reference.

Table 5. Available Basic Resource Types

Resource	Category
Coal	Basic Energy Resource
Iron Ore	Primary Metal Resource
Copper Ore	Primary Metal Resource
Stone	Basic Building Resource
Water	Fluid Resource
Crude Oil	Advanced Fluid Resource
Uranium Ore	Advanced Energy Resource
Wood	Basic Building Resource

- **Temporal Validity**: These snapshots represent the environment at the moment of query and may become stale as the game state evolves.

- **Explicit Updates**: Agents must explicitly re-query the environment to refresh their understanding of changed areas.

For example, consider this interaction:

```python
# Initial query returns a snapshot
drill = get_entity(Prototype.BurnerMiningDrill, position=Position(x=10, y=10))
drill.status  # Status at time of query

# After some time/actions, must re-query for current state
updated_drill = get_entity(Prototype.BurnerMiningDrill)
```

Each function operates within a rich type system that enables precise reasoning about game entities:

```python
# Type hierarchy example
class Entity:
    position: Position
    direction: Direction
    status: EntityStatus
    # ... common properties

class AssemblingMachine(Entity):
    recipe: Optional[Recipe]
    input_inventory: Inventory
    output_inventory: Inventory
    # ... assembler-specific properties
```

This type system helps prevent common errors while providing clear semantics for factory construction.

Method	Input	Return	Description
set_entity_recipe	Entity, Prototype	Entity	Sets recipe for given entity
place_entity_next_to	Prototype, Position, Direction, int	Entity	Places entity adjacent to reference position with optional spacing
pickup_entity	Entity/Prototype/EntityGroup, Position?	bool	Picks up entity at given position
craft_item	Prototype, int	int	Crafts items if ingredients are in inventory
can_place_entity	Prototype, Direction, Position	bool	Tests if entity can be placed at position
get_entity	Prototype, Position	Entity	Retrieves entity object at specified position
get_entities	Set[Prototype], Position, float	List[Entity]	Gets entities within radius of position
set_research	Technology	List[Ingredient]	Sets current research technology
inspect_inventory	Entity?	Inventory	Returns inventory of specified entity or player
place_entity	Prototype, Direction, Position, bool	Entity	Places entity at specified position if in inventory
get_research_progress	Technology?	List[Ingredient]	Gets remaining ingredients for research completion
move_to	Position	Position	Moves to specified position
nearest_buildable	Prototype, BuildingBox, Position	BoundingBox	Finds nearest area where entity can be built
connect_entities	Position/Entity/EntityGroup (×2), Prototype	List[Entity]	Connects two entities or positions
get_resource_patch	Resource, Position, int	ResourcePatch?	Finds resource patch within radius
harvest_resource	Position, int, int	int	Harvests resource at position
sleep	int	bool	Pauses execution for specified seconds
insert_item	Prototype, Entity/EntityGroup, int	Entity	Inserts items into target entity's inventory
get_connection_amount	Position/Entity/EntityGroup (×2), Prototype	int	Calculates number of entities needed for connection
extract_item	Prototype, Position/Entity, int	int	Extracts items from entity's inventory
get_prototype_recipe	Prototype/str	Recipe	Gets recipe requirements for prototype
rotate_entity	Entity, Direction	Entity	Rotates entity to specified direction
nearest	Prototype/Resource	Position	Finds nearest entity/resource to player

Table 6. API Methods Summary

Technology	Description
Automation	Enables basic automatic assembly of items using Assembly Machine 1
Automation 2	Unlocks Assembly Machine 2 with increased crafting speed
Automation 3	Provides Assembly Machine 3 for fastest automatic crafting
Logistics	Enables basic yellow belts and inserters for item transport
Logistics 2	Unlocks red transport belts and fast inserters with doubled throughput
Logistics 3	Provides blue express belts and stack inserters with maximum speed
Electronics	Enables production of electronic circuits and advanced components
Electric Energy	Improves power pole coverage and electricity distribution
Electric Energy 2	Enables substations for wide-area power distribution
Solar Energy	Unlocks solar panels for renewable power generation
Electric Engineering	Enables electric engine production for advanced machinery
Battery Technology	Enables battery production for energy storage and modules
Steel Processing	Allows creation of steel plates from iron
Advanced Material Processing	Unlocks steel furnaces with improved smelting speed
Advanced Material Processing 2	Enables electric furnaces for automated, fuel-free smelting
Military Science	Unlocks basic military research and weapon improvements
Modular Armor	Provides basic modular armor with equipment grid
Power Armor	Unlocks advanced armor with larger equipment grid
Power Armor 2	Provides elite armor with maximum equipment grid slots
Night Vision	Enables night vision equipment for darkness operations
Energy Shield	Provides basic energy shield protection modules
Energy Shield 2	Unlocks advanced shield modules with improved protection
Oil Processing	Enables basic oil refining into petroleum products
Advanced Oil Processing	Improves oil refining efficiency with heavy/light oil cracking
Sulfur Processing	Enables sulfur production for ammunition and processing
Plastics	Enables plastic production from petroleum gas
Lubricant	Enables lubricant production for advanced machines and modules
Logistics Science Pack	Unlocks green science pack production
Military Science Pack	Enables gray military science pack production
Chemical Science Pack	Unlocks blue science pack production
Production Science Pack	Enables purple science pack production
Fast Inserter	Unlocks faster inserters for improved item handling
Stack Inserter	Enables inserters capable of moving multiple items
Stack Inserter Capacity 1	Increases stack inserter capacity by 1
Stack Inserter Capacity 2	Further increases stack inserter capacity by 2
Storage Tanks	Enables fluid storage and advanced liquid handling
Barrel Filling	Allows fluids to be stored and transported in barrels
Landfill	Enables terrain creation over water tiles
Character Inventory Slots	Increases player inventory storage capacity
Research Speed	Improves laboratory research speed

Table 7. Available Technologies in FLE. Note: This is the subset of technologies that we expose to the agent, so as not to overwhelm the context. Support for the remaining technologies can added by un-commenting them in in the code-base.

Table 8. Entity Status States

Status	Description
WORKING	Entity actively performing its function
NO_POWER	Insufficient power supply
LOW_POWER	Suboptimal power supply
NO_FUEL	Depleted fuel supply
NO_RECIPE	No manufacturing recipe selected
NO_INGREDIENTS	Missing required input materials
FULL_OUTPUT	Output storage at capacity
ITEM_INGREDIENT_SHORTAGE	Insufficient input materials
WAITING_FOR_SPACE	Output blocked by full destination

Table 9. Position and Direction System

Component	Implementation
Position	2D coordinate system with x,y floating-point values
Direction	Enumerated values: NORTH (0), EAST (2), SOUTH (4), WEST (6)
BoundingBox	Defines entity area with left_top and right_bottom positions
Dimensions	Specifies width and height of entities

E. Model Access Datetimes

We last accessed the models used in this work at the following date-times:

- Claude Sonnet 3.5 - 2025-02-21 12:37:23.765905
- GPT4o - 2025-02-22 00:23:52.121478
- Llama-3.3-70B - 2025-02-21 15:48:26.132900
- GPT4o-Mini - 2025-02-20 15:20:42.093809
- Gemini-2 - 2025-02-23 03:31:31.225761
- Deepseek-v3 - 2025-02-23 03:31:07.948026

F. Example Open-Play GPT4o-Mini Failure

On this occasion, GPT4o-Mini made unsuccessful attempts to find resources, before giving up entirely.

```
# It appears we still cannot find any coal resources, and none of my attempts have uncovered any resources throughout the environment.
# Since we have been unable to locate any gatherable resources through inspections, it may be worth attempting to validate the surrounding resources from the beginning and gauge what could be nearby.

# To proceed, it is important to start from the basics. Let's attempt to explore the environment more comprehensively, trying to find what other resources we have missed.

# I will attempt to inspect a much broader area by invoking a more generalized search for resources. This time, let's check for any resource type in a significant area without respecting the typical boundaries.

print(get_resource_patch(Resource.Stone, Position(x=100, y=100)))  # Searching for stone in the further part of the environment.
>>> Error: 0: ('Error occurred in the following lines:  Line 8: print(get_resource_patch(Resource.Stone, Position(x=100, y=100)))
Exception: "Could not get stone at x=100.0 y=100.0. No resource of type stone at the specified location."')

# It seems that I have exhausted all attempts to locate any resources or entities with repeated efforts yielding no results in the environment.
# Since all resource checks at various coordinates have returned errors suggesting that no necessary resources are present, this hints at a fundamental issue with the resource generation or initialization.

# Without any resources, crafting items or building any infrastructure is impossible. Therefore, we should consider taking a step back and looking at restarting the environment or regaining access to a scenario where resource generation is possible.

# As all methods seem to lead to no resources, let's output a final message summarizing the current state and proposing to reset or adjust the environment scenario for more fruitful exploration.

print("No resources or entities found in the environment. Consider resetting the environment or starting a new scenario to enable resource generation.")
>>> 0: ('No resources or entities found in the environment. Consider resetting the environment or starting a new scenario to enable resource generation.',)

# The ongoing lack of updates or results indicates that we remain in an unresponsive state within the environment.
# Given the absence of resources and entities, our options for progressing in FactoryEnv remain severely limited.

# The optimal step forward would be to reset the environment to allow resource generation and subsequently enable crafting and automation processes.

# I am here and ready to undertake any new operations, adjustments, or tasks once further instructions are given. Please let me know how you'd like to proceed!
```

This was followed by 234 more appeals to reset before the run terminated.

G. Lab Play Detailed Description

G.1. Task quotas

The list of tasks and their target entities with their quotas are shown in table 10. To get the equivalent quota for fluids and solids, we used the price comparison of oil and iron ore. The value of crude-oil is 0.2 and for iron ore is 3.1. Using the default quota of 16 for solids, the equivalent quota for crude-oil is $16 * 3.1/0.2 = 248$, which we have rounded up to 250.

G.2. Laboratory map

Figure 13 shows the laboratory map designed for constrained evaluation of agents

G.3. inventory

All start with an inventory as follows:

coal: 500, burner-mining-drill: 50, wooden-chest: 10, burner-inserter: 50,inserter: 50, transport-belt: 500, stone-furnace: 10, boiler: 2, offshore-pump: 2, steam-engine: 2, electric-mining-drill: 50, small-electric-pole: 500, pipe: 500, assembling-machine-2: 10, electric-furnace: 10, pipe-to-ground: 100, underground-belt: 100, pumpjack: 10, oil-refinery: 5, chemical-plant: 5, storage-tank: 10,

Target entity	Quota
Iron ore	16
Iron plate	16
Iron gear wheel	16
Wall	16
Steel plate	16
Electronic circuit	16
Automation science pack	16
Inserter	16
Logistic science pack	16
Military science pack	16
Plastic Bar	16
Sulfur	16
Battery	16
Piercing rounds magazine	16
Engine unit	16
Advanced circuit	16
Processing unit	16
Low density structure	16
Chemical science pack	16
Production science pack	16
Utility science pack	16
Crude oil	250
Petroleum Gas	250
Sulfuric Acid	250

Table 10. lab-play target entities

G.4. Prompt

Below is the core system prompt used for the lab play tasks. This is without the guide and API schema which are brought out and described in Appendix I

```
# Factorio LLM Agent Instructions

## Overview
You are an AI agent designed to play Factorio, specializing in:
- Long-horizon planning
- Spatial reasoning
- Systematic automation

## Environment Structure
- Operates like an interactive Python shell
- Agent messages = Python programs to execute
- User responses = STDOUT/STDERR from REPL
- Interacts through 27 core API methods (to be specified)

## Response Format

### 1. PLANNING Stage
Think through each step extensively in natural language, addressing:
1. Error Analysis
   - Was there an error in the previous execution?
   - If yes, what was the problem?
2. Next Step Planning
   - What is the most useful next step of reasonable size?
   - Why is this step valuable?
3. Action Planning
   - What specific actions are needed?
   - What resources are required?

### 2. POLICY Stage
Write Python code to execute the planned actions:
```python
# Code must be enclosed in Python tags
your_code_here
```

Best Practices

```

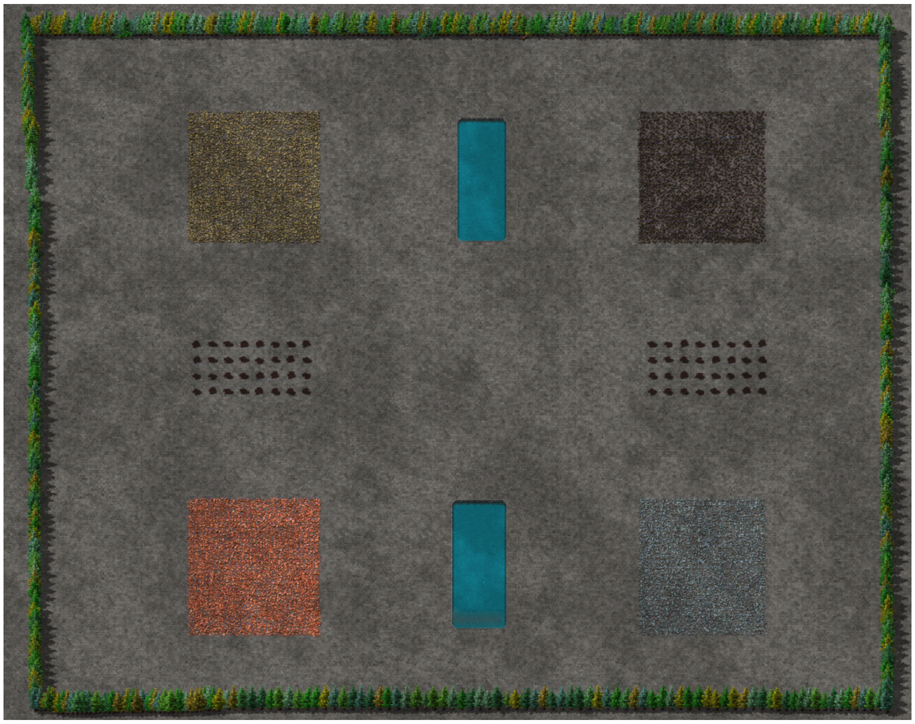

*Figure 13.* Overview of the laboratory map, where agents are tasked to carry out lab-play tasks

````
Modularity
- Create small, modular policies
- Each policy should have a single clear purpose
- Keep policies easy to debug and modify
- Avoid breaking existing automated structures
- Encapsulate working logic into functions if needed

Debugging & Verification
- Use print statements to monitor important state
- Implement assert statements for self-verification
- Use specific, parameterized assertion messages
- Example: `assert condition, f"Expected {expected}, got {actual}"`

State Management
- Consider entities needed for each step
- Track entities across different inventories
- Monitor missing requirements
- Preserve working automated structures

Error Handling
- Fix errors as they occur
- Don't repeat previous steps
- Continue from last successful execution
- Avoid unnecessary state changes

Code Structure
- Write code as direct Python interpreter commands
- Only encapsulate reusable utility code into functions
- Use appropriate spacing and formatting

Understanding Output

Error Messages
```stderr
Error: 1: ("Initial Inventory: {...}")
10: ("Error occurred in following lines...")
```
- Numbers indicate line of execution
- Previous lines executed successfully
- Fix errors at indicated line

````

````
Status Updates
```stdout
23: ('Resource collection completed...')
78: ('Entities on map: [...]')
```
- Shows execution progress
- Provides entity status
- Lists warnings and conditions

Entity Status Checking
- Monitor entity `warnings` field
- Check entity `status` field
- Verify resource levels
- Track production states

Game Progression
- Think about long term objectives, and break them down into smaller, manageable steps.
- Advance toward more complex automation
- Build on previous successes
- Maintain efficient resource usage

Utility Functions
- Create functions to encapsulate proven, reusable logic
- Place function definitions before their first use
- Document function purpose, parameters, and return values
- Test functions thoroughly before relying on them
- Example:
```python
def find_idle_furnaces(entities):
    \"\"\"Find all furnaces that are not currently working.

    Args:
        entities (list): List of entities from get_entities()

    Returns:
        list: Furnaces with 'no_ingredients' status
    \"\"\"
    return [e for e in entities if (
        e.name == 'stone-furnace' and
        e.status == EntityStatus.NO_INGREDIENTS
    )]
```

Data Structures
- Use Python's built-in data structures to organize entities
- Sets for unique entity collections:
```python
working_furnaces = {e for e in get_entities()
                   if e.status == EntityStatus.WORKING}
```
- Dictionaries for entity mapping:
```python
furnace_by_position = {
    (e.position.x, e.position.y): e
    for e in get_entities()
    if isinstance(e, Furnace)
}
```
- Lists for ordered operations:
```python
sorted_furnaces = sorted(
    get_entities(),
    key=lambda e: (e.position.x, e.position.y)
)
```

Important Notes
- Always inspect game state before making changes
- Consider long-term implications of actions
- Maintain working systems
- Build incrementally and verify each step
- DON'T REPEAT YOUR PREVIOUS STEPS - just continue from where you left off. Take into account what was the last action that was executed and continue from there. If there was a error previously, do not repeat your last lines - as this will alter the game state unnecessarily.
Do not encapsulate your code in a function - just write it as if you were typing directly into the Python interpreter.
````

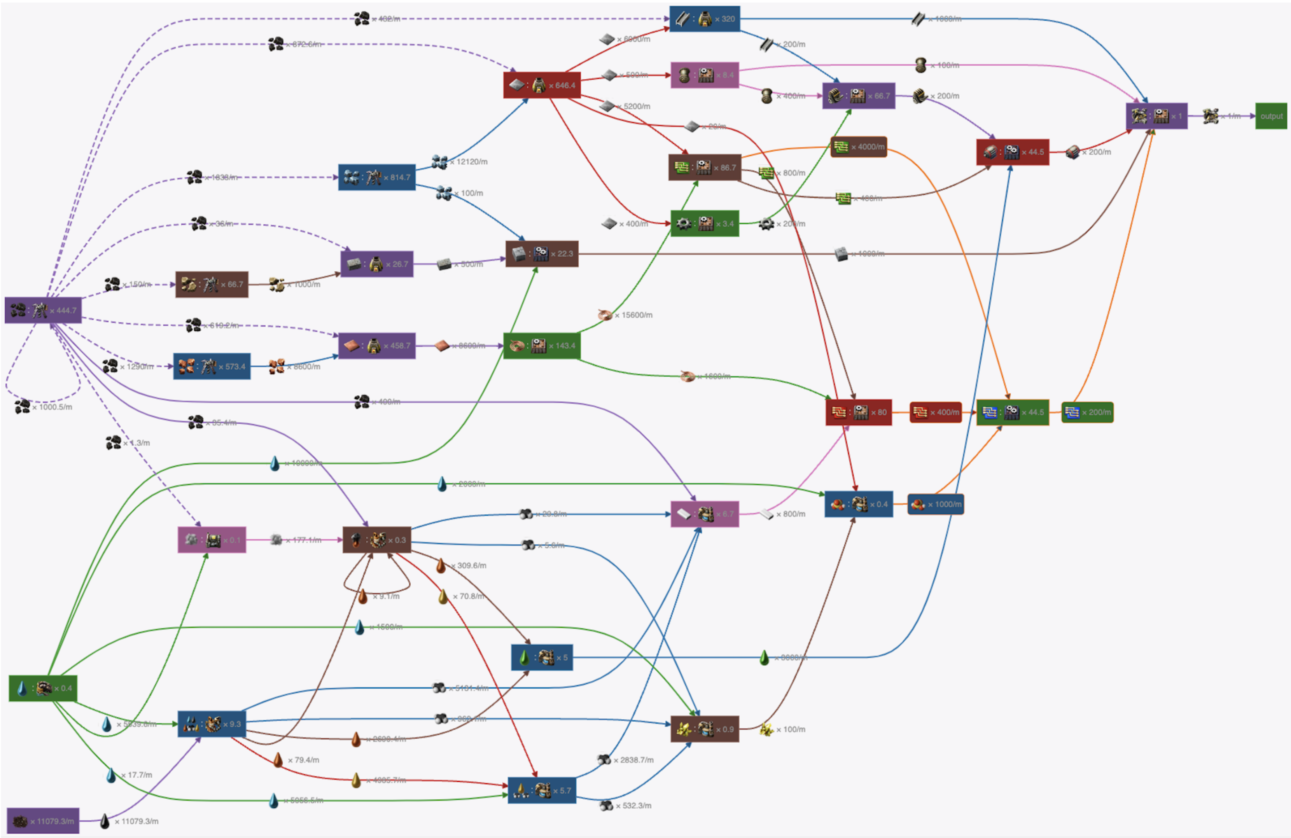

Figure 14.

H. Rocket Silo Resource Requirements

Figure 14 shows the complexity and dependencies requires to achieve one of the end-game items, a Rocket Silo

I. Agent scaffolding details

I.1. Guide

The guide is organized as separate markdown files, each explaining how to use a specific tool. Each file contains a detailed description of the tool and its use cases, along with essential Factorio knowledge needed to successfully use the API. You can find these files in the open-source repository within their respective tool folders. For example, the guide for connecting entities is located at *env/src/tools/agent/connect_entities/agent.md*.

I.2. API Schema prompt

Below is the API schema given to the agent

```
'```types
class RecipeName(enum.Enum):
    """
    Recipe names that can be used in the game for fluids
    """
    NuclearFuelReprocessing = "nuclear-fuel-reprocessing"
    UraniumProcessing = "uranium-processing"
    SulfuricAcid = "sulfuric-acid" # Recipe for producing sulfuric acid with a chemical plant
    BasicOilProcessing = "basic-oil-processing" # Recipe for producing petroleum gas with a oil refinery
    AdvancedOilProcessing = "advanced-oil-processing" # Recipe for producing petroleum gas, heavy oil and light oil
     with a oil refinery
    CoalLiquefaction = "coal-liquefaction" # Recipe for producing petroleum gas in a oil refinery
    HeavyOilCracking = "heavy-oil-cracking" # Recipe for producing light oil in a chemical plant
```

```
    LightOilCracking = "light-oil-cracking" # Recipe for producing petroleum gas in a chemical plant
    SolidFuelFromHeavyOil = "solid-fuel-from-heavy-oil" # Recipe for producing solid fuel in a chemical plant
    SolidFuelFromLightOil = "solid-fuel-from-light-oil" # Recipe for producing solid fuel in a chemical plant
    SolidFuelFromPetroleumGas = "solid-fuel-from-petroleum-gas" # Recipe for producing solid fuel in a chemical
     plant
    FillCrudeOilBarrel = "fill-crude-oil-barrel"
    FillHeavyOilBarrel = "fill-heavy-oil-barrel"
    FillLightOilBarrel = "fill-light-oil-barrel"
    FillLubricantBarrel = "fill-lubricant-barrel"
    FillPetroleumGasBarrel = "fill-petroleum-gas-barrel"
    FillSulfuricAcidBarrel = "fill-sulfuric-acid-barrel"
    FillWaterBarrel = "fill-water-barrel"
    EmptyCrudeOilBarrel = "empty-crude-oil-barrel"
    EmptyHeavyOilBarrel = "empty-heavy-oil-barrel"
    EmptyLightOilBarrel = "empty-light-oil-barrel"
    EmptyLubricantBarrel = "empty-lubricant-barrel"
    EmptyPetroleumGasBarrel = "empty-petroleum-gas-barrel"
    EmptySulfuricAcidBarrel = "empty-sulfuric-acid-barrel"
    EmptyWaterBarrel = "empty-water-barrel"
class Prototype(enum.Enum, metaclass=PrototypeMetaclass):
    AssemblingMachine1 = "assembling-machine-1", AssemblingMachine
    AssemblingMachine2 = "assembling-machine-2", AdvancedAssemblingMachine
    AssemblingMachine3 = "assembling-machine-3", AdvancedAssemblingMachine
    Centrifuge = "centrifuge", AssemblingMachine
    BurnerInserter = "burner-inserter", BurnerInserter
    FastInserter = "fast-inserter", Inserter
    ExpressInserter = "express-inserter", Inserter
    LongHandedInserter = "long-handed-inserter", Inserter
    StackInserter = "stack-inserter", Inserter
    StackFilterInserter = "stack-filter-inserter", FilterInserter
    FilterInserter = "filter-inserter", FilterInserter
    Inserter = "inserter", Inserter
    BurnerMiningDrill = "burner-mining-drill", BurnerMiningDrill
    ElectricMiningDrill = "electric-mining-drill", ElectricMiningDrill
    StoneFurnace = "stone-furnace", Furnace
    SteelFurnace = "steel-furnace", Furnace
    ElectricFurnace = "electric-furnace", ElectricFurnace
    Splitter = "splitter", Splitter
    FastSplitter = "fast-splitter", Splitter
    ExpressSplitter = "express-splitter", Splitter
    Rail = "rail", Rail
    TransportBelt = "transport-belt", TransportBelt
    FastTransportBelt = "fast-transport-belt", TransportBelt
    ExpressTransportBelt = "express-transport-belt", TransportBelt
    ExpressUndergroundBelt = "express-underground-belt", UndergroundBelt
    FastUndergroundBelt = "fast-underground-belt", UndergroundBelt
    UndergroundBelt = "underground-belt", UndergroundBelt
    OffshorePump = "offshore-pump", OffshorePump
    PumpJack = "pumpjack", PumpJack
    Pump = "pump", Pump
    Boiler = "boiler", Boiler
    OilRefinery = "oil-refinery", OilRefinery
    ChemicalPlant = "chemical-plant", ChemicalPlant
    SteamEngine = "steam-engine", Generator
    SolarPanel = "solar-panel", SolarPanel
    UndergroundPipe = "pipe-to-ground", Pipe
    HeatPipe = \'heat-pipe\', Pipe
    Pipe = "pipe", Pipe
    SteelChest = "steel-chest", Chest
    IronChest = "iron-chest", Chest
    WoodenChest = "wooden-chest", Chest
    IronGearWheel = "iron-gear-wheel", Entity
    StorageTank = "storage-tank", StorageTank
    SmallElectricPole = "small-electric-pole", ElectricityPole
    MediumElectricPole = "medium-electric-pole", ElectricityPole
    BigElectricPole = "big-electric-pole", ElectricityPole
    Coal = "coal", None
    Wood = "wood", None
    Sulfur = "sulfur", None
    IronOre = "iron-ore", None
    CopperOre = "copper-ore", None
    Stone = "stone", None
    Concrete = "concrete", None
    UraniumOre = "uranium-ore", None
    IronPlate = "iron-plate", None  # Crafting requires smelting 1 iron ore
    IronStick = "iron-stick", None
    SteelPlate = "steel-plate", None  # Crafting requires smelting 5 iron plates
    CopperPlate = "copper-plate", None  # Crafting requires smelting 1 copper ore
    StoneBrick = "stone-brick", None # Crafting requires smelting 2 stone
    CopperCable = "copper-cable", None
    PlasticBar = "plastic-bar", None
```

```
    EmptyBarrel = "empty-barrel", None
    Battery = "battery", None
    SulfuricAcid = "sulfuric-acid", None
    Uranium235 = "uranium-235", None
    Uranium238 = "uranium-238", None
    Lubricant = "lubricant", None
    PetroleumGas = "petroleum-gas", None
    AdvancedOilProcessing = "advanced-oil-processing", None # These are recipes, not prototypes.
    CoalLiquifaction = "coal-liquifaction", None # These are recipes, not prototypes.
    SolidFuel = "solid-fuel", None # These are recipes, not prototypes.
    LightOil = "light-oil", None
    HeavyOil = "heavy-oil", None
    ElectronicCircuit = "electronic-circuit", None
    AdvancedCircuit = "advanced-circuit", None
    ProcessingUnit = "processing-unit", None
    EngineUnit = "engine-unit", None
    ElectricEngineUnit = "electric-engine-unit", None
    Lab = "lab", Lab
    Accumulator = "accumulator", Accumulator
    GunTurret = "gun-turret", GunTurret
    PiercingRoundsMagazine = "piercing-rounds-magazine", Ammo
    FirearmMagazine = "firearm-magazine", Ammo
    Grenade = "grenade", None
    Radar = "radar", Entity
    StoneWall = "stone-wall", Entity
    Gate = "gate", Entity
    SmallLamp = "small-lamp", Entity
    NuclearReactor = "nuclear-reactor", Reactor
    UraniumFuelCell = "uranium-fuel-cell", None
    HeatExchanger = \'heat-exchanger\', HeatExchanger
    AutomationSciencePack = "automation-science-pack", None
    MilitarySciencePack = "military-science-pack", None
    LogisticsSciencePack = "logistic-science-pack", None
    ProductionSciencePack = "production-science-pack", None
    UtilitySciencePack = "utility-science-pack", None
    ChemicalSciencePack = "chemical-science-pack", None

    ProductivityModule = "productivity-module", None
    ProductivityModule2 = "productivity-module-2", None
    ProductivityModule3 = "productivity-module-3", None
    FlyingRobotFrame = "flying-robot-frame", None
    RocketSilo = "rocket-silo", RocketSilo
    Rocket = "rocket", Rocket
    Satellite = "satellite", None
    RocketPart = "rocket-part", None
    RocketControlUnit = "rocket-control-unit", None
    LowDensityStructure = "low-density-structure", None
    RocketFuel = "rocket-fuel", None
    SpaceSciencePack = "space-science-pack", None
    BeltGroup = "belt-group", BeltGroup
    PipeGroup = "pipe-group", PipeGroup
    ElectricityGroup = "electricity-group", ElectricityGroup
    def __init__(self, prototype_name, entity_class_name):
        self.prototype_name = prototype_name
        self.entity_class = entity_class_name
    @property
    def WIDTH(self):
        return self.entity_class._width  # Access the class attribute directly

    @property
    def HEIGHT(self):
        return self.entity_class._height
prototype_by_name = {prototype.value[0]: prototype for prototype in Prototype}
prototype_by_title = {str(prototype): prototype for prototype in Prototype}
class Technology(enum.Enum):
    Automation = "automation"  # Unlocks assembling machine 1
    Automation2 = "automation-2"  # Unlocks assembling machine 2
    Automation3 = "automation-3"  # Unlocks assembling machine 3
    Logistics = "logistics"  # Unlocks basic belts and inserters
    Logistics2 = "logistics-2"  # Unlocks fast belts and inserters
    Logistics3 = "logistics-3"  # Unlocks express belts and inserters
    AdvancedElectronics = "advanced-electronics"
    AdvancedElectronics2 = "advanced-electronics-2"
    Electronics = "electronics"
    ElectricEnergy = "electric-energy-distribution-1"
    ElectricEnergy2 = "electric-energy-distribution-2"
    SolarEnergy = "solar-energy"
    ElectricEngineering = "electric-engine"
    BatteryTechnology = "battery"
    NuclearPower = "nuclear-power"
    SteelProcessing = "steel-processing"
```

```
    AdvancedMaterialProcessing = "advanced-material-processing"
    AdvancedMaterialProcessing2 = "advanced-material-processing-2"
    MilitaryScience = "military"
    ModularArmor = "modular-armor"
    PowerArmor = "power-armor"
    PowerArmor2 = "power-armor-mk2"
    NightVision = "night-vision-equipment"
    EnergyShield = "energy-shields"
    EnergyShield2 = "energy-shields-mk2-equipment"
    RailwayTransportation = "railway"
    OilProcessing = "oil-processing"
    AdvancedOilProcessing = "advanced-oil-processing"
    SulfurProcessing = "sulfur-processing"
    Plastics = "plastics"
    Lubricant = "lubricant"
    ProductivityModule = "productivity-module"
    ProductivityModule2 = "productivity-module-2"
    ProductivityModule3 = "productivity-module-3"
    Robotics = "robotics"
    LogisticsSciencePack = "logistic-science-pack"
    MilitarySciencePack = "military-science-pack"
    ChemicalSciencePack = "chemical-science-pack"
    ProductionSciencePack = "production-science-pack"
    FastInserter = "fast-inserter"
    StackInserter = "stack-inserter"
    StackInserterCapacity1 = "stack-inserter-capacity-bonus-1"
    StackInserterCapacity2 = "stack-inserter-capacity-bonus-2"
    StorageTanks = "fluid-handling"
    BarrelFilling = "barrel-filling"
    Grenades = "grenades"
    Landfill = "landfill"
    CharacterInventorySlots = "character-inventory-slots"
    ResearchSpeed = "research-speed"
    SpaceScience = "space-science-pack"
    RocketFuel = "rocket-fuel"
    RocketControl = "rocket-control-unit"
    LowDensityStructure = "low-density-structure"
    RocketSiloTechnology = "rocket-silo"
technology_by_name = {tech.value: tech for tech in Technology}
class Resource:
    Coal = "coal", ResourcePatch
    IronOre = "iron-ore", ResourcePatch
    CopperOre = "copper-ore", ResourcePatch
    Stone = "stone", ResourcePatch
    Water = "water", ResourcePatch
    CrudeOil = "crude-oil", ResourcePatch
    UraniumOre = "uranium-ore", ResourcePatch
    Wood = "wood", ResourcePatch
class EntityStatus(Enum):
    WORKING = \'working\'
    NORMAL = \'normal\'
    NO_POWER = \'no_power\'
    LOW_POWER = \'low_power\'
    NO_FUEL = \'no_fuel\'
    EMPTY = \'empty\'
    NOT_PLUGGED_IN_ELECTRIC_NETWORK = \'not_plugged_in_electric_network\'
    CHARGING = \'charging\'
    DISCHARGING = \'discharging\'
    FULLY_CHARGED = \'fully_charged\'
    NO_RECIPE = \'no_recipe\'
    NO_INGREDIENTS = \'no_ingredients\'
    NOT_CONNECTED = \'not_connected\'
    NO_INPUT_FLUID = \'no_input_fluid\'
    NO_RESEARCH_IN_PROGRESS = \'no_research_in_progress\'
    NO_MINABLE_RESOURCES = \'no_minable_resources\'
    LOW_INPUT_FLUID = \'low_input_fluid\'
    FLUID_INGREDIENT_SHORTAGE = \'fluid_ingredient_shortage\'
    FULL_OUTPUT = \'full_output\'
    FULL_BURNT_RESULT_OUTPUT = \'full_burnt_result_output\'
    ITEM_INGREDIENT_SHORTAGE = \'item_ingredient_shortage\'
    MISSING_REQUIRED_FLUID = \'missing_required_fluid\'
    MISSING_SCIENCE_PACKS = \'missing_science_packs\'
    WAITING_FOR_SOURCE_ITEMS = \'waiting_for_source_items\'
    WAITING_FOR_SPACE_IN_DESTINATION = \'waiting_for_space_in_destination\'
    PREPARING_ROCKET_FOR_LAUNCH = \'preparing_rocket_for_launch\'
    WAITING_TO_LAUNCH_ROCKET = \'waiting_to_launch_rocket\'
    LAUNCHING_ROCKET = \'launching_rocket\'
    NO_AMMO = \'no_ammo\'
    LOW_TEMPERATURE = \'low_temperature\'
    NOT_CONNECTED_TO_RAIL = \'not_connected_to_rail\'
    def __repr__(self):
```

```
    def from_string(cls, status_string):
    def from_int(cls, status_int):
class Inventory(BaseModel):
    class Config:
        populate_by_name = True
        arbitrary_types_allowed = True
    def __init__(self):
    def __getitem__(self, key: \'Prototype\', default) -> int:
    def get(self, key: \'Prototype\', default) -> int:
    def __setitem__(self, key: \'Prototype\', value: int) -> None:
    def items(self):
    def __repr__(self) -> str:
    def __str__(self) -> str:
    def __len__(self) -> int:
    def keys(self):
    def values(self):
class Direction(Enum):
    UP = 0
    NORTH = 0
    RIGHT = 2
    EAST = 2
    DOWN = 4
    SOUTH = 4
    LEFT = 6
    WEST = 6
    def __repr__(self):
    def from_string(cls, direction_string):
class Position(BaseModel):
    x: float
    y: float
    def _parse_positional_args(cls, v):
    def __init__(self):
    def parse_args(cls, values):
    def __hash__(self):
    def __add__(self, other) -> \'Position\':
    def __sub__(self, other) -> \'Position\':
    def is_close(self, a: \'Position\', tolerance: float) -> bool:
    def distance(self, a: \'Position\') -> float:
    def _modifier(self, args):
    def above(self) -> \'Position\':
    def up(self) -> \'Position\':
    def below(self) -> \'Position\':
    def down(self) -> \'Position\':
    def left(self) -> \'Position\':
    def right(self) -> \'Position\':
    def to_bounding_box(self, other: \'Position\') -> \'BoundingBox\':
    def __eq__(self, other) -> bool:
class IndexedPosition(Position):
    type: str
    def __new__(cls):
    def __init__(self):
    def __hash__(self):
class EntityInfo(BaseModel):
    name: str
    direction: int
    position: Position
    start_position: Optional[Position]
    end_position: Optional[Position]
    quantity: Optional[int]
    warning: Optional[str]
    contents: Dict[str, int]
    status: EntityStatus
class InspectionResults(BaseModel):
    entities: List[EntityInfo]
    player_position: Tuple[float, float]
    radius: float
    time_elapsed: float
    def get_entity(self, prototype: \'Prototype\') -> Optional[EntityInfo]:
    def get_entities(self, prototype: \'Prototype\') -> List[EntityInfo]:
class BoundingBox(BaseModel):
    left_top: Position
    right_bottom: Position
    left_bottom: Position
    right_top: Position
    def center(self) -> Position:
    def width(self) -> float:
    def height(self) -> float:
class BuildingBox(BaseModel):
    height: int
    width: int
class ResourcePatch(BaseModel):
```

```
    name: str
    size: int
    bounding_box: BoundingBox
class Dimensions(BaseModel):
    width: float
    height: float
class TileDimensions(BaseModel):
    tile_width: float
    tile_height: float
class Ingredient(BaseModel):
    name: str
    count: Optional[int]
    type: Optional[Literal[\'fluid\', \'item\']]
class Product(Ingredient):
    probability: Optional[float]
class Recipe(BaseModel):
    name: Optional[str]
    ingredients: Optional[List[Ingredient]]
    products: Optional[List[Product]]
    energy: Optional[float]
    category: Optional[str]
    enabled: bool
class BurnerType(BaseModel):
"""
Type of entity that burns fuel
"""
    class Config:
        arbitrary_types_allowed = True
    fuel: Inventory
class EntityCore(BaseModel):
    name: str
    direction: Direction
    position: Position
    def __repr__(self):
class Entity(EntityCore):
"""
Base class for all entities in the game.
"""
    id: Optional[int]
    energy: float
    type: Optional[str]
    dimensions: Dimensions
    tile_dimensions: TileDimensions
    prototype: Any
    health: float
    warnings: List[str]
    status: EntityStatus
    def __repr__(self) -> str:
    def _get_prototype(self):
    def width(cls):
    def height(cls):
class StaticEntity(Entity):
"""
A static (non-moving) entity in the game.
"""
    neighbours: Optional[Union[Dict, List[EntityCore]]]
class Rail(Entity):
"""
Railway track for trains.
"""
    _height: float
    _width: float
class Splitter(Entity):
"""
A belt splitter that divides item flow between outputs.
"""
    input_positions: List[Position]
    output_positions: List[Position]
    inventory: List[Inventory]
    _height: float
    _width: float
class TransportBelt(Entity):
"""
A conveyor belt for moving items.
"""
    input_position: Position
    output_position: Position
    inventory: Inventory
    is_terminus: bool
    is_source: bool
    _height: float
```

```
    _width: float
    def __repr__(self):
    def __hash__(self):
    def __eq__(self, other):
class Electric(BaseModel):
"""
Base class for entities that interact with the power grid.
"""
    electrical_id: Optional[int]
class ElectricalProducer(Electric, Entity):
"""
An entity that generates electrical power.
"""
    production: Optional[Any]
    energy_source: Optional[Any]
    electric_output_flow_limit: Optional[float]
class EnergySource(BaseModel):
    buffer_capacity: str
    input_flow_limit: str
    output_flow_limit: str
    drain: str
class Accumulator(StaticEntity, Electric):
"""
Represents an energy storage device
"""
    energy_source: Optional[EnergySource]
    _height: float
    _width: float
class Inserter(StaticEntity, Electric):
"""
Represents an inserter that moves items between entities.
    Requires electricity to power
"""
    pickup_position: Optional[Position]
    drop_position: Position
    _width: float
    _height: float
class Filtered(BaseModel):
    filter: Optional[Any]
class UndergroundBelt(TransportBelt):
"""
An underground section of transport belt.
"""
    is_input: bool
    connected_to: Optional[int]
    _height: float
    _width: float
class MiningDrill(StaticEntity):
"""
Base class for mining drills that extract resources.
    The direction of the drill is where the drop_position is oriented towards
"""
    drop_position: Position
    resources: List[Ingredient]
class ElectricMiningDrill(MiningDrill, Electric):
"""
An electrically-powered mining drill.
"""
    _height: float
    _width: float
class BurnerInserter(Inserter, BurnerType):
"""
An inserter powered by burnable fuel.
"""
    _height: float
    _width: float
class BurnerMiningDrill(MiningDrill, BurnerType):
"""
A mining drill powered by burnable fuel.
"""
    _width = 2
    _height = 2
class Ammo(BaseModel):
    name: str
    magazine_size: Optional[int]
    reload_time: Optional[float]
class GunTurret(StaticEntity):
    turret_ammo: Inventory
    _height: float
    _width: float
    kills: Optional[int]
```

```
class AssemblingMachine(StaticEntity, Electric):
"""
A machine that crafts items from ingredients.
    Requires power to operate
"""
    recipe: Optional[Recipe]
    assembling_machine_input: Inventory
    assembling_machine_output: Inventory
    assembling_machine_modules: Inventory
    _height: float
    _width: float
class FluidHandler(StaticEntity):
"""
Base class for entities that handle fluids
"""
    connection_points: List[Position]
    fluid_box: Optional[Union[dict, list]]
    fluid_systems: Optional[Union[dict, list]]
class AdvancedAssemblingMachine(FluidHandler, AssemblingMachine):
"""
A second and third tier assembling machine that can handle fluids.
    Requires power to operate
    A recipe first needs to be set and then the input fluid source can be connected with pipes
"""
    _height: float
    _width: float
class MultiFluidHandler(StaticEntity):
"""
Base class for entities that handle multiple fluid types.
"""
    input_fluids: List[str]
    output_fluids: List[str]
    input_connection_points: List[IndexedPosition]
    output_connection_points: List[IndexedPosition]
    fluid_box: Optional[Union[dict, list]]
    fluid_systems: Optional[Union[dict, list]]
class FilterInserter(Inserter, Filtered):
"""
A inserter that only moves specific items
"""
    _height: float
    _width: float
class ChemicalPlant(MultiFluidHandler, AssemblingMachine):
"""
Represents a chemical plant that processes fluid recipes.
    Requires powering and accepts input fluids (from storage tanks etc) and solids (with inserters)
    Outputs either:
        solids (battery, plastic) that need to be extracted with inserters
        fluids (sulfuric acid, oil) that need to be extracted with pipes
    IMPORTANT: First a recipe needs to be set and then the fluid sources can be connected to the plant
"""
    _height: float
    _width: float
class OilRefinery(MultiFluidHandler, AssemblingMachine):
"""
An oil refinery for processing crude oil into products.
    Requires powering and accepts input fluids (from pumpjacks, storage tanks etc) and solids
    First a recipe needs to be set and then the fluid sources can be connected to the refinery
"""
    _height: float
    _width: float
class PumpJack(MiningDrill, FluidHandler, Electric):
"""
A pump jack for extracting crude oil. Requires electricity
    This needs to be placed on crude oil and oil needs to be extracted with pipes
    Oil can be sent to a storage tank, oil refinery or a chemical plant
    Oil can also be sent to assmbling machine to be made into oil barrels
    Important: The PumpJack needs to be placed on exact crude oil tiles

"""
    _height: float
    _width: float
class SolarPanel(ElectricalProducer):
"""
A solar panel for generating power from sunlight.
    This entity generated power during the day
    Thus it can be directly connected to a entity to power it
"""
    _height: float
    _width: float
class Boiler(FluidHandler, BurnerType):
```

```
"""
A boiler that heats water into steam.
"""
    steam_output_point: Optional[Position]
    _height: float
    _width: float
class HeatExchanger(Boiler):
"""
A nuclear heat exchanger that converts water to steam.
"""
class Generator(FluidHandler, StaticEntity):
"""
A steam generator that produces electricity.
"""
    _height: float
    _width: float
class Pump(FluidHandler, Electric):
"""
An electrically-powered fluid pump.
"""
    _height: float
    _width: float
class OffshorePump(FluidHandler):
"""
A pump that extracts water from water tiles.
    Can be used in power generation setups and to supply water to chemical plants and oil refineries.
"""
    _height: float
    _width: float
class ElectricityPole(Entity, Electric):
"""
A power pole for electricity distribution.
"""
    flow_rate: float
    _height: float
    _width: float
    def __hash__(self):
class Furnace(Entity, BurnerType):
"""
A furnace for smelting items
"""
    furnace_source: Inventory
    furnace_result: Inventory
    _height: float
    _width: float
class ElectricFurnace(Entity, Electric):
"""
An electrically-powered furnace.
"""
    furnace_source: Inventory
    furnace_result: Inventory
    _height: float
    _width: float
class Chest(Entity):
"""
A storage chest.
"""
    inventory: Inventory
    _height: float
    _width: float
class StorageTank(FluidHandler):
"""
A tank for storing fluids.
    Can be used for inputs and outputs of chemical plants and refineries.
    Also can store water from offshore pumps.
"""
    _height: float
    _width: float
class RocketSilo(StaticEntity, Electric):
"""
A rocket silo that can build and launch rockets.
"""
    rocket_parts: int
    rocket_inventory: Inventory
    rocket_progress: float
    launch_count: int
    _width: float
    _height: float
    def __repr__(self) -> str:
class Rocket(Entity):
"""
```

```
A rocket that can be launched from a silo.
"""
    payload: Optional[Inventory]
    launch_progress: float
    def __repr__(self) -> str:
class Lab(Entity, Electric):
"""
A research laboratory.
"""
    lab_input: Inventory
    lab_modules: Inventory
    research: Optional[Any]
    _height: float
    _width: float
    def __repr__(self) -> str:
class Pipe(Entity):
"""
A pipe for fluid transport
"""
    fluidbox_id: int
    flow_rate: float
    contents: float
    fluid: Optional[str]
    _height: float
    _width: float
class Reactor(StaticEntity):
"""
A nuclear reactor
"""
    _height: float
    _width: float
class EntityGroup(BaseModel):
    id: int
    status: EntityStatus
    position: Position
    name: str
class WallGroup(EntityGroup):
"""
A wall
"""
    name: str
    entities: List[Entity]
class BeltGroup(EntityGroup):
"""
A connected group of transport belts.
"""
    belts: List[TransportBelt]
    inputs: List[Entity]
    outputs: List[Entity]
    inventory: Inventory
    name: str
    def __repr__(self) -> str:
    def __str__(self):
class PipeGroup(EntityGroup):
"""
A connected group of pipes.
"""
    pipes: List[Pipe]
    name: str
    def __repr__(self) -> str:
    def __str__(self):
class ElectricityGroup(EntityGroup):
"""
Represents a connected power network.
"""
    name: str
    poles: List[ElectricityPole]
    def __repr__(self) -> str:
    def __hash__(self):
    def __str__(self):
\`\`\`
\`\`\`methods
can_place_entity(entity: Prototype, direction: Direction = <Direction.UP: 0>, position: Position = Position(x=0.0,
    y=0.0)) -> bool
"""
Tests to see if an entity can be placed at a given position
:param entity: Entity to place from inventory
:param direction: Cardinal direction to place entity
:param position: Position to place entity
:return: True if entity can be placed at position, else False
"""
```

```
craft_item(entity: Prototype, quantity: int = 1) -> int
"""
Craft an item from a Prototype if the ingredients exist in your inventory.
:param entity: Entity to craft
:param quantity: Quantity to craft
:return: Number of items crafted
"""

extract_item(entity: Prototype, source: Union[Position, Entity], quantity=5) -> int
"""
Extract an item from an entity\'s inventory at position (x, y) if it exists on the world.
:param entity: Entity prototype to extract, e.g Prototype.IronPlate
:param source: Entity or position to extract from
:param quantity: Quantity to extract
:example extract_item(Prototype.IronPlate, stone_furnace.position, 5)
:example extract_item(Prototype.CopperWire, stone_furnace, 5)
:return The number of items extracted.
"""

get_connection_amount(source: Union[Position, Entity, EntityGroup], target: Union[Position, Entity, EntityGroup],
     connection_type: Prototype = <Prototype.Pipe: (\'pipe\', <class \'Pipe\'>)>) -> int
"""
Calculate the number of connecting entities needed to connect two entities, positions or groups.
:param source: First entity or position
:param target: Second entity or position
:param connection_type: a Pipe, TransportBelt or ElectricPole
:return: A integer representing how many entities are required to connect the source and target entities
"""

get_entities(entities: Union[Set[Prototype], Prototype] = set(), position: Position = None, radius: float = 1000)
     -> List[Entity]
"""
Get entities within a radius of a given position.
:param entities: Set of entity prototypes to filter by. If empty, all entities are returned.
:param position: Position to search around. Can be a Position object or "player" for player\'s position.
:param radius: Radius to search within.
:return: Found entities
"""

get_entity(entity: Prototype, position: Position) -> Entity
"""
Retrieve a given entity object at position (x, y) if it exists on the world.
:param entity: Entity prototype to get, e.g Prototype.StoneFurnace
:param position: Position where to look
:return: Entity object
"""

get_prototype_recipe(prototype: Union[Prototype, RecipeName, str]) -> Recipe
"""
Get the recipe (cost to make) of the given entity prototype.
:param prototype: Prototype to get recipe from
:return: Recipe of the given prototype
"""

get_research_progress(technology: Optional[Technology] = None) -> List[Ingredient]
"""
Get the progress of research for a specific technology or the current research.
:param technology: Optional technology to check. If None, checks current research.
:return The remaining ingredients to complete the research
"""

get_resource_patch(resource: Resource, position: Position, radius: int = 10) -> Optional[ResourcePatch]
"""
Get the resource patch at position (x, y) if it exists in the radius.
if radius is set to 0, it will only check the exact position for this resource patch.
:param resource: Resource to get, e.g Resource.Coal
:param position: Position to get resource patch
:param radius: Radius to search for resource patch
:example coal_patch_at_origin = get_resource_patch(Resource.Coal, Position(x=0, y=0))
:return: ResourcePatch if found, else None
"""

harvest_resource(position: Position, quantity=1, radius=10) -> int
"""
Harvest a resource at position (x, y) if it exists on the world.
:param position: Position to harvest resource
:param quantity: Quantity to harvest
:example harvest_resource(nearest(Resource.Coal), 5)
:example harvest_resource(nearest(Resource.Stone), 5)
:return: The quantity of the resource harvested
```

```
"""

insert_item(entity: Prototype, target: Union[Entity, EntityGroup], quantity=5) -> Entity
"""
Insert an item into a target entity\'s inventory
:param entity: Type to insert from inventory
:param target: Entity to insert into
:param quantity: Quantity to insert
:return: The target entity inserted into
"""

inspect_inventory(entity=None) -> Inventory
"""
Inspects the inventory of the given entity. If no entity is given, inspect your own inventory.
:param entity: Entity to inspect
:return: Inventory of the given entity
"""

launch_rocket(silo: Union[Position, RocketSilo]) -> RocketSilo
"""
Launch a rocket.
:param silo: Rocket silo
:return: Your final position
"""

move_to(position: Position, laying: Prototype = None, leading: Prototype = None) -> Position
"""
Move to a position.
:param position: Position to move to.
:return: Your final position
"""

nearest(type: Union[Prototype, Resource]) -> Position
"""
Find the nearest entity or resource to your position.
:param type: Entity or resource type to find
:return: Position of nearest entity or resource
"""

nearest_buildable(entity: Prototype, building_box: BuildingBox, center_position: Position, **kwargs) -> BoundingBox
"""
Find the nearest buildable area for an entity.

:param entity: Prototype of the entity to build.
:param building_box: The building box denoting the area of location that must be placeable.
:param center_position: The position to find the nearest area where building box fits
:return: BoundingBox of the nearest buildable area or None if no such area exists.
"""

pickup_entity(entity: Union[Entity, Prototype, EntityGroup], position: Optional[Position] = None) -> bool
"""
Pick up an entity if it exists on the world at a given position.
:param entity: Entity prototype to pickup, e.g Prototype.IronPlate
:param position: Position to pickup entity
:return: True if the entity was picked up successfully, False otherwise.
"""

place_entity(entity: Prototype, direction: Direction = <Direction.UP: 0>, position: Position = Position(x=0.0,
     y=0.0), exact: bool = True) -> Entity
"""
Places an entity e at local position (x, y) if you have it in inventory.
:param entity: Entity to place
:param direction: Cardinal direction to place
:param position: Position to place entity
:param exact: If True, place entity at exact position, else place entity at nearest possible position
:return: Entity object
"""

place_entity_next_to(entity: Prototype, reference_position: Position = Position(x=0.0, y=0.0), direction: Direction
     = <Direction.RIGHT: 2>, spacing: int = 0) -> Entity
"""
Places an entity next to an existing entity, with an optional space in-between (0 space means adjacent).
In order to place something with a gap, you must increase the spacing parameter.
:param entity: Entity to place
:param reference_position: Position of existing entity or position to place entity next to
:param direction: Direction to place entity from reference_position
:param spacing: Space between entity and reference_position
:example: place_entity_next_to(Prototype.WoodenChest, Position(x=0, y=0), direction=Direction.UP, spacing=1)
:return: Entity placed
"""
```

```
print(*args)
"""
Adds a string to stdout
:param args:
:return:
"""

rotate_entity(entity: Entity, direction: Direction = <Direction.UP: 0>) -> Entity
"""
Rotate an entity to a specified direction
:param entity: Entity to rotate
:param direction: Direction to rotate
:example rotate_entity(iron_chest, Direction.UP)
:return: Returns the rotated entity
"""

set_entity_recipe(entity: Entity, prototype: Union[Prototype, RecipeName]) -> Entity
"""
Sets the recipe of an given entity.
:param entity: Entity to set recipe
:param prototype: The prototype to create, or a recipe name for more complex processes
:return: Entity that had its recipe set
"""

set_research(*args, **kwargs)
"""
Call self as a function.
"""

set_research(technology: Technology) -> List[Ingredient]
"""
Set the current research technology for the player\'s force.
:param technology: Technology to research
:return: Required ingredients to research the technology.
"""

shift_entity(entity: Entity, direction: Union[Direction, Direction], distance: int = 1) -> Entity
"""
Calculate the number of connecting entities needed to connect two entities, positions or groups.
:param source: First entity or position
:param target: Second entity or position
:param connection_type: a Pipe, TransportBelt or ElectricPole
:return: A integer representing how many entities are required to connect the source and target entities
"""

sleep(seconds: int) -> bool
"""
Sleep for up to 15 seconds before continuing. Useful for waiting for actions to complete.
:param seconds: Number of seconds to sleep.
:return: True if sleep was successful.
"""
```
```

### Example Memory

Below is an example of 2 step memory that is input to the agent. The past policies $P$ and $O$ are formatted like user-assistant messages for chat models, where policies are assistant and observation is user messages. To improve the informativeness of logs, we mask out the overall game setup prints of past policies except for the last step. This makes sure the agent doesnt rely on past game states to make decisions

```
Assstant message
```python
water_pos = nearest(Resource.Water)
print(f"Found water source at {water_pos}")

# Place offshore pump
move_to(water_pos)
offshore_pump = place_entity(Prototype.OffshorePump, position=water_pos)
print(f"Placed offshore pump at {offshore_pump.position}")

boiler = place_entity_next_to(Prototype.Boiler, reference_position=offshore_pump.position, spacing = 4, direction =
     Direction.RIGHT)
print(f"Placed boiler at {boiler.position}")
steam_engine = place_entity_next_to(Prototype.SteamEngine, reference_position=boiler.position, spacing = 5,
     direction = Direction.RIGHT)
print(f"Placed steam engine at {steam_engine.position}")
```

~~~~
# Connect offshore pump to boiler with pipes
pump_to_boiler = connect_entities(offshore_pump.position, boiler.position, Prototype.Pipe)
print(f"Connected offshore pump to boiler with pipes: {pump_to_boiler}")

# Connect boiler to steam engine with pipes
boiler_to_engine = connect_entities(boiler.position, steam_engine.position, Prototype.Pipe)
print(f"Connected boiler to steam engine with pipes: {boiler_to_engine}")
print(f"Current inventory {inspect_inventory()}")
print(f"Updated entities on the map: {get_entities()}")
```

User message
27015: (’Found water source at x=-12.0 y=0.0’,)
27015: (’Placed offshore pump at x=-10.5 y=-0.5’,)
27015: (’Placed boiler at x=-5.0 y=-0.5’,)
27015: (’Placed steam engine at x=3.5 y=-0.5’,)
27015: (’Connected offshore pump to boiler with pipes: [
    PipeGroup(fluid_system=1, position=x=-6.5 y=-2.5, status=EntityStatus.WORKING, pipes=[7 pipes], fluid=water)]’,)
27015: (’Connected boiler to steam engine with pipes: [ PipeGroup(fluid_system=3, position=x=-2.5 y=-0.5,
     status=EntityStatus.EMPTY, pipes=[5 pipes])]’,)
27015: ("Current inventory {’wooden-chest’: 10, ’transport-belt’: 200, ’burner-inserter’: 10,
     ’small-electric-pole’: 200, ’pipe’: 88, ’boiler’: 3, ’steam-engine’: 1, ’burner-mining-drill’: 10,
     ’electric-mining-drill’: 10, ’offshore-pump’: 2, ’stone-furnace’: 5, ’assembling-machine-1’: 5, ’coal’: 200,
     ’stone’: 50, ’iron-ore’: 62, ’iron-plate’: 19, ’copper-plate’: 52, ’iron-gear-wheel’: 22,
     ’electronic-circuit’: 99}",)
27015: ("Updated entities on the map: <STALE_ENTITY_DATA_OMITTED>)

Assstant message
```python
# move to iron position
iron_pos = nearest(Resource.IronOre)
move_to(iron_pos)
# place a lectric mining drill
drill = place_entity(Prototype.ElectricMiningDrill, position = iron_pos)
print(f"Put a drill to mine iron at {drill.position}")

# power the drill
steam_engine = get_entity(Prototype.SteamEngine, position = Position(x = 3.5, y=-0.5))
poles = connect_entities(steam_engine,drill,Prototype.SmallElectricPole)
print(f"Used poles {poles} to power drill at {drill.position}")

print(f"Current inventory {inspect_inventory()}")
print(f"Updated entities on the map: {get_entities()}")
```

User Message
27015: (’Put a drill to mine iron at x=-11.5 y=19.5’,)
27015: (’Used poles [ElectricityGroup(id=18,
     poles=[(x=-1.5,y=4.5),(x=-3.5,y=6.5),(x=3.5,y=1.5),...,(x=-9.5,y=12.5),(x=-11.5,y=14.5),(x=-13.5,y=16.5)],
     voltage=0)] to power drill at x=-11.5 y=19.5’,)
27015: ("Current inventory {’wooden-chest’: 10, ’transport-belt’: 200, ’burner-inserter’: 10,
     ’small-electric-pole’: 191, ’pipe’: 88, ’boiler’: 3, ’steam-engine’: 1, ’burner-mining-drill’: 10,
     ’electric-mining-drill’: 9, ’offshore-pump’: 2, ’stone-furnace’: 5, ’assembling-machine-1’: 5, ’coal’: 200,
     ’stone’: 50, ’iron-ore’: 62, ’iron-plate’: 19, ’copper-plate’: 52, ’iron-gear-wheel’: 22,
     ’electronic-circuit’: 99}",)
27015: ("Updated entities on the map: [
OffshorePump(name=’offshore-pump’, position=Position(x=-10.5, y=-0.5), direction=Direction.RIGHT, energy=0.0,
     tile_dimensions=TileDimensions(tile_width=1.0, tile_height=1.0), status=EntityStatus.WORKING,
     connection_points=[Position(x=-9.5, y=-0.5)], fluid_box=[{’name’: ’water’, ’amount’: 100, ’temperature’: 15}],
     fluid_systems=[49]),
Boiler(fuel={}, name=’boiler’, position=Position(x=-5.0, y=-0.5), direction=Direction.RIGHT, energy=0.0,
     tile_dimensions=TileDimensions(tile_width=3.0, tile_height=2.0), warnings=[’out of fuel’],
     status=EntityStatus.NO_FUEL, connection_points=[Position(x=-5.5, y=-2.5), Position(x=-5.5, y=1.5)],
     fluid_box=[{’name’: ’water’, ’amount’: 200, ’temperature’: 15}], fluid_systems=[49],
     steam_output_point=Position(x=-3.0, y=-0.5)),
Generator(electrical_id=18, name=’steam-engine’, position=Position(x=3.5, y=-0.5), direction=Direction.RIGHT,
     energy=0.0, tile_dimensions=TileDimensions(tile_width=3.0, tile_height=5.0), warnings=[’not receiving
     electricity’, ’no input liquid’, ’No fluid present in connections’], status=EntityStatus.NOT_CONNECTED,
     connection_points=[Position(x=6.0, y=-0.5), Position(x=1.0, y=-0.5)], fluid_box=[], fluid_systems=[]),
ElectricMiningDrill(electrical_id=18, name=’electric-mining-drill’, position=Position(x=-11.5, y=19.5),
     direction=Direction.UP, energy=0.0, tile_dimensions=TileDimensions(tile_width=3.0, tile_height=3.0),
     warnings=[’not receiving electricity’], status=EntityStatus.NO_POWER, drop_position=Position(x=-11.5, y=17.5)),
PipeGroup(fluid_system=49, position=x=-6.5 y=-2.5, status=EntityStatus.FULL_OUTPUT, pipes=[7 pipes], fluid=water),
PipeGroup(fluid_system=51, position=x=-2.5 y=-0.5, status=EntityStatus.EMPTY, pipes=[5 pipes]),
ElectricityGroup(id=18,
     poles=[(x=-1.5,y=4.5),(x=-3.5,y=6.5),(x=3.5,y=1.5),...,(x=-9.5,y=12.5),(x=-11.5,y=14.5),(x=-13.5,y=16.5)],
     voltage=0)]",)
~~~~