
Factorio Learning Environment

Jack Hopkins^{*1} Mart Bakler^{*1} Akbir Khan²

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 platform¹.

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

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

Preliminary work. Under review.

¹<https://github.com/JackHopkins/factorio-learning-environment>



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.

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, unbounded evaluation framework built upon the game of Factorio that uniquely addresses this limitation. FLE is an open-ended evaluation with no natural completion state. In this environment, agents must progress 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 throughput, which ranges from early-game rates of ~ 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 (such as Claude-3.5-Sonnet and GPT-4o) invest heavily into technological

research achieving 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

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

Factorio Learning Environment

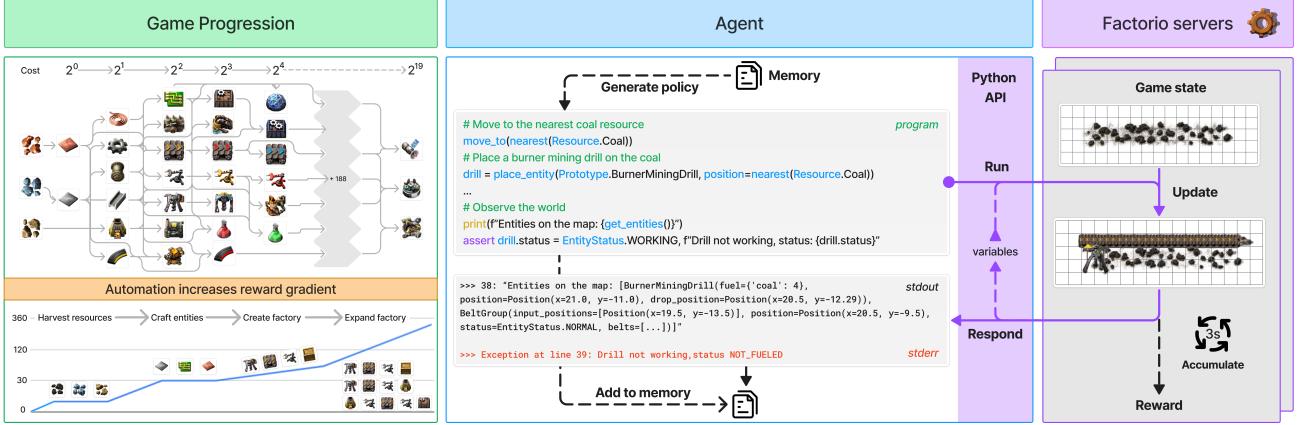


Figure 2. Illustration of the Factorio Learning Environment (FLE). FLE is based on the popular construction and management simulation game *Factorio*. Left: The open-ended goal of the game is to create the largest factory possible. The game enables agents to invest in (an infinite number of) technological 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 penalties), and receive a response as feedback. If the agents chooses, it may view its own production statistics.

```

1 # 1. Get iron patch and place mining drill
2 drill = place_entity(
3     entity=Prototype.MiningDrill,
4     position=nearest(Resource.IronOre),
5     direction=Direction.NORTH
6 )
7 # 2. Add output storage
8 chest = place_entity_next_to(
9     entity=Prototype.IronChest,
10    reference_position=drill.drop_position,
11    direction=Direction.SOUTH
12 )
13 # 3. Verify automation chain and observe entities
14 assert drill.status == EntityStatus.WORKING
15 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.

enabled by technologies like beacons and stack inserters, 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:

```

1 # Observe the status of all Furnaces in-game
2 print([e.status for e in
3       get_entities({Prototype.StoneFurnace})])
4 >>> 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).

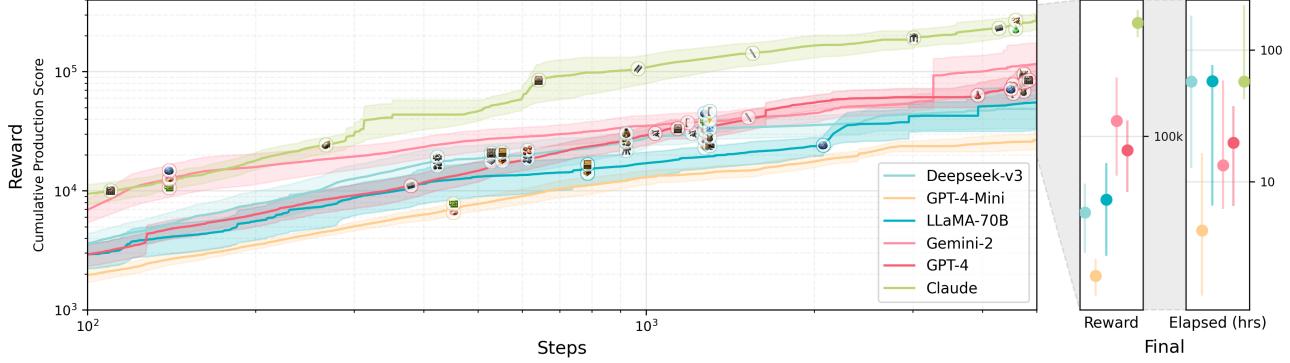


Figure 4. Models are differentiated by score in Open-Play. Agents are given the instruction to *build the biggest possible factory*. Left: We find that by evaluating PS against steps (server calls) we can clearly differentiate stronger models from weaker ones in a log/log projection. We overlay milestones, 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 TCP³. 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 speed-up over the game client (see Figure 11). We make the environment publicly available⁴.

³Roughly 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 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 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 are in Appendix H.

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 . 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 and std of each task success with 4 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 common patterns 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 J. 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 32 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. 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 ex-

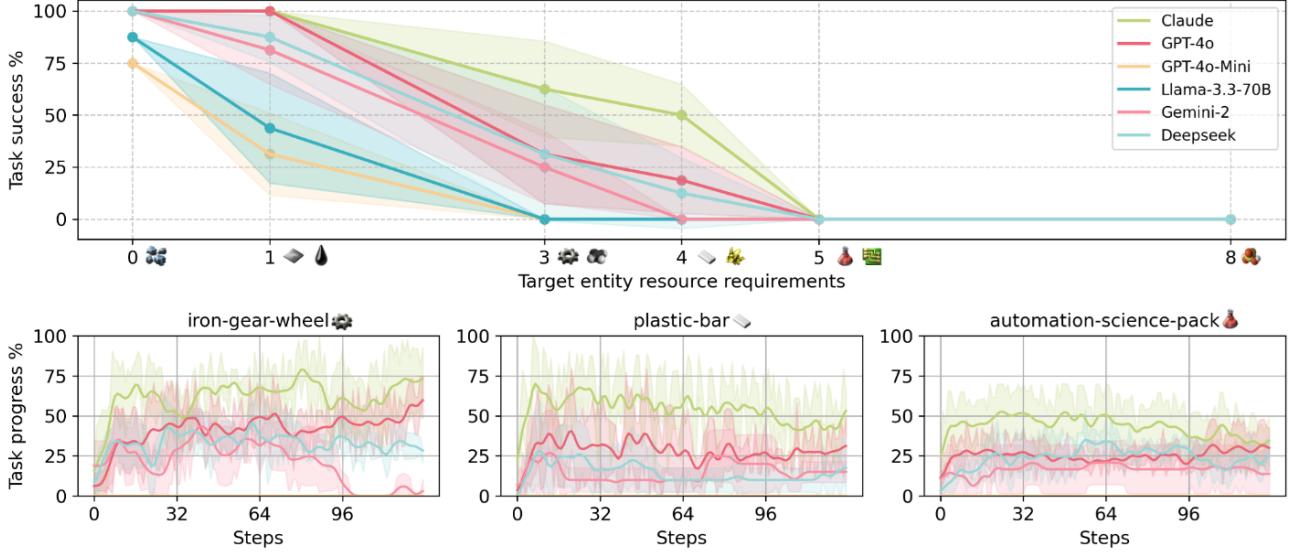


Figure 5. Agents are unable to consistently build complex and efficient factories in Lab-Play. Top: We measure the mean and standard deviation of task success rates across the first 8 complexity levels and task progress (percentage of target ingredients and its sub-ingredients agents factories produce at each time-step) in three tasks of increasing difficulty. We observe a clear decrease in average task success rates as the crafting complexity of the target entity increases. Bottom: 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 mean task progress across runs.

traction 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 performance. 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

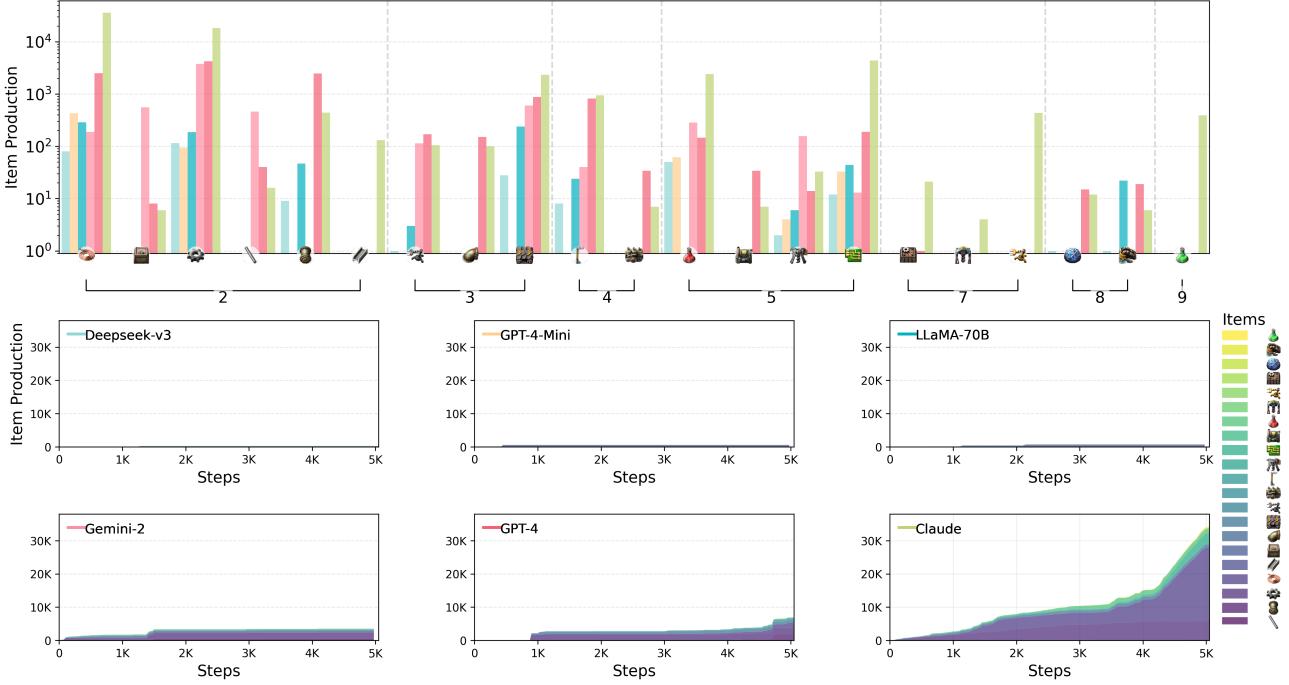


Figure 6. Open-ended challenges highlight differences in objective setting and general capability. We illustrate the rates at which various models produce items with multiple antecedent ingredients in the *open-play* setting. Claude 3.5-Sonnet immediately begins complex crafting and invests in research, 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-ingradient 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 ± 1.3
GPT-4o	16.6 ± 1.4
Deepseek-v3	15.1 ± 1.7
Gemini-2	13.0 ± 1.3
LLama-3.3-70B	6.3 ± 1.0
GPT-4o-Mini	5.2 ± 0.6

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.

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 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

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).

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 game-state; 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 perform as well as frontier agents, it is unclear if achieving end-game goals (e.g. escape the world or build rockets) are achievable to humans using only an API in a reasonable time-frame. We did however prove that each step in the chain was achievable from the previous step.

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, only the entities that are relevant to the early and mid-game are explicitly modelled in FLE. 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 late-game entities including trains, logistics robots and agent-programmable circuit networks.

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).

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. 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.

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/clause-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., Bhamri, 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](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., Rotstod, 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>.
- Pagliari, D., Cupial, B., Coward, S., Piterbarg, U., Wolczyk, M., Khan, A., Pignatelli, E., Kuciński, Ł., Pinto, L., Ferguson, 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., Piñeras, 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., Rustemi, A., Clay, N., Crone, P., Kociský, 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., Ozturk, A., Cassirer, A., Xu, Y., Sohn, D., Sachan, D., Amplayio, 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., yin 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., Housby, 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., Seymour, 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., Mata, J., Qian, Y., Peswani, V., Janus, P., Yuan, Q., Schelin, L., David, O., Garg, A., He, Y., Duzhi, 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., Dektarev, 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., Yoge, R., Cai, X., Agostini, A., Shah, M., Nguyen, H., Donnaile, N. O., Pereira, S., Friso, L., Stambl, 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., Petro-

- vski, I., Cedoz, P.-L., Li, C., Chen, S., Santo, N. D., Goyal, S., Punjabi, J., Kappagantu, K., Kwak, C., LV, P., Velury, S., Choudhury, H., Hall, J., Shah, P., Figueira, R., Thomas, M., Lu, M., Zhou, T., Kumar, C., Jurd, 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., Ponnappalli, 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., Kravonva, 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., Urias, B., Ko, Y., Knight, L., Hélio, A., Niu, N., Gu, S., Pang, C., Li, Y., Levine, N., Stolovich, A., Santamaría-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., Pöder, 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., Klimeczak-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:

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

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

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:

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

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)
Llama-3.3-70B-Instruct-Turbo	85,619,488	496,227	86,115,715	75.78
GPT-4o	150,294,347	2,632,650	152,926,997	402.06
Claude-3.5-Sonnet	151,294,110	3,249,040	154,543,150	502.62
GPT-4o-Mini	116,024,065	3,002,180	119,026,245	19.20
Lab (all)	178,604,280	4,086,434	182,693,315	318.62

Table 3. Token Usage and Cost Comparison across Models. Lab-play costs are aggregated. The total cost is \$1,318.28

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 $\langle s_1, s_2, \dots, s_n \rangle$. Each statement s is either a method invocation or a variable declaration:

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

- $m = (f, args, ret)$ is a method invocation
- $f \in F$ is a function identifier from our API method set F
- $args = (a_1, a_2, \dots, a_k)$ 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:

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

Method execution transforms only the game state:

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

While namespace context C is modified only through variable declarations:

$$\text{declare} : V \times T \times C \rightarrow C' \quad (5)$$

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.

```

1 # Pure query - affects neither G nor C
2 recipe = get_prototype_recipe(Prototype.IronGearWheel)
3 # Effects on game state G only (G -> G' x T)
4 success = set_entity_recipe(assembler, recipe)
5 # Namespace context C is modified only through assignments
6 assembler = place_entity_next_to(           # Method: G -> G' x T_Entity
7     entity=Prototype.AssemblingMachine2,      # Variable declaration: C -> C'
8     reference_position= inserter.position,    # Reference from C
9     direction=Direction.RIGHT,
10    spacing=1
11 )
12 # Runtime assertions can verify both game state and namespace
13 assert isinstance(assembler, AssemblingMachine)
14 assert get_entity(
15     Prototype.AssemblingMachine2,
16     assembler.position
17 ) 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.
- **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:

```

1 # Initial query returns a snapshot
2 drill = get_entity(Prototype.BurnerMiningDrill, position=Position(x=10, y=10))
3 drill.status # Status at time of query
4
5 # After some time/actions, must re-query for current state
6 updated_drill = get_entity(Prototype.BurnerMiningDrill)

```

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

```

1 # Type hierarchy example
2 class Entity:
3     position: Position
4     direction: Direction
5     status: EntityStatus
6     # ... common properties
7
8 class AssemblingMachine(Entity):
9     recipe: Optional[Recipe]
10    input_inventory: Inventory
11    output_inventory: Inventory
12    # ... assembler-specific properties

```

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

Table 4. 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

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, bool Position?		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 (x2), 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/Entity- Group, int	Entity	Inserts items into target entity's inventory
get_connection_amount	Position/Entity/EntityGroup (x2), Prototype	int	Calculates number of entities needed for connection
extract_item	Prototype, Position/Entity, 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 5. 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 6. 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 be added by un-commenting them in the code-base.

Table 7. 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 8. 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 - 2025-01-29 23:55:08.498032
- GPT4o - 2025-01-28 15:14:30.622424
- Llama-3.3-70B - 2025-01-28 15:14:44.530134
- GPT4o-Mini - 2025-01-29 16:05:08.835405

F. Example Open-Play GPT4o-Mini Failure

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

```

1 # It appears we still cannot find any coal resources, and none of my attempts have uncovered any resources
2   throughout the environment.
3 # Since we have been unable to locate any gatherable resources through inspections, it may be worth attempting to
4   validate the surrounding resources from the beginning and gauge what could be nearby.
5
6 # To proceed, it is important to start from the basics. Let's attempt to explore the environment more
7   comprehensively, trying to find what other resources we have missed.
8
9 # I will attempt to inspect a much broader area by invoking a more generalized search for resources. This time,
10  let's check for any resource type in a significant area without respecting the typical boundaries.
11
12 print(get_resource_patch(Resource.Stone, Position(x=100, y=100))) # Searching for stone in the further part of the
13  environment.
14 >>> Error: 0: ('Error occurred in the following lines: Line 8: print(get_resource_patch(Resource.Stone,
15    Position(x=100, y=100)))
16 Exception: "Could not get stone at x=100.0 y=100.0. No resource of type stone at the specified location."')
17
18 # It seems that I have exhausted all attempts to locate any resources or entities with repeated efforts yielding no
19   results in the environment.
20 # Since all resource checks at various coordinates have returned errors suggesting that no necessary resources are
21   present, this hints at a fundamental issue with the resource generation or initialization.
22
23 # Without any resources, crafting items or building any infrastructure is impossible. Therefore, we should consider
24   taking a step back and looking at restarting the environment or regaining access to a scenario where resource
25   generation is possible.
26
27 # As all methods seem to lead to no resources, let's output a final message summarizing the current state and
28   proposing to reset or adjust the environment scenario for more fruitful exploration.
29
30 print("No resources or entities found in the environment. Consider resetting the environment or starting a new
31   scenario to enable resource generation.")
32 >>> 0: ('No resources or entities found in the environment. Consider resetting the environment or starting a new
33   scenario to enable resource generation.',)
34
35 # The ongoing lack of updates or results indicates that we remain in an unresponsive state within the environment.
36 # Given the absence of resources and entities, our options for progressing in FactoryEnv remain severely limited.
37
38 # The optimal step forward would be to reset the environment to allow resource generation and subsequently enable
39   crafting and automation processes.
40
41 # I am here and ready to undertake any new operations, adjustments, or tasks once further instructions are given.
42 Please let me know how you'd like to proceed!

```

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

G. Debugging Detailed Description

Below we present program-level statistics for each model during lab-play.

Model	Total Progs	Err. Progs	Err. Rate	Avg.% Lines	Avg.% Asserts	Asserts /Prog	SuccAsserts /Prog	Total Asserts	Total SuccAsserts	Overall Assert Succ.
Llama-3.3-70B	2480	1770	71.37	64.26	68.18	0.47	0.20	1155	504	43.64
GPT-4o	3624	1161	32.04	90.80	95.70	0.27	0.21	982	746	75.97
Claude 3.5	3297	1384	41.98	78.46	99.82	0.02	0.02	77	68	88.31
GPT-4o-mini	3836	814	21.22	92.82	90.68	0.11	0.03	438	122	27.85

Table 9. Program-level statistics in open-play. **Total Progs** is how many code snippets the model generated; **Err. Progs** is how many snippets triggered a runtime error; **Err. Rate** is the fraction of error-ridden snippets; **Avg.% Lines** is the average fraction of lines successfully executed (before error or completion); **Avg.% Asserts** is the fraction of assert statements that actually ran before any error occurred; **Asserts/Prog** is the mean number of asserts per snippet; **SuccAsserts/Prog** is how many of those asserts did *not* fail (i.e. claims were correct); **Total Asserts** and **SuccAsserts** sum over all snippets; **Overall Assert Succ.** is $\frac{\text{total successful asserts}}{\text{total asserts}} \times 100$.

Model	Syntactic	Semantic	Other
Llama-3.3-70B	13.0%	10.1%	76.9%
GPT-4o	5.1%	33.4%	61.6%
Claude 3.5	3.0%	1.0%	96.1%
GPT-4o-Mini	2.8%	3.4%	93.9%

Table 10. Error category breakdown. Here, *syntactic* errors indicate malformed code (`SyntaxError`, `NameError`, etc.), *semantic* errors are explicitly failed `assert` statements (the agent’s “belief” was false), and *other* errors are environment-level or logic failures (like invalid item requests, or placing an entity on an obstructed tile). Percentages reflect the proportion of each error category among all error-triggering snippets for that model.

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 11. Open-play task setups

H. Lab Play Detailed Description

H.1. Task quotas

The list tasks and their target entities with their quotas are shown in table 11. 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.

H.2. Laboratory map

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

H.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,

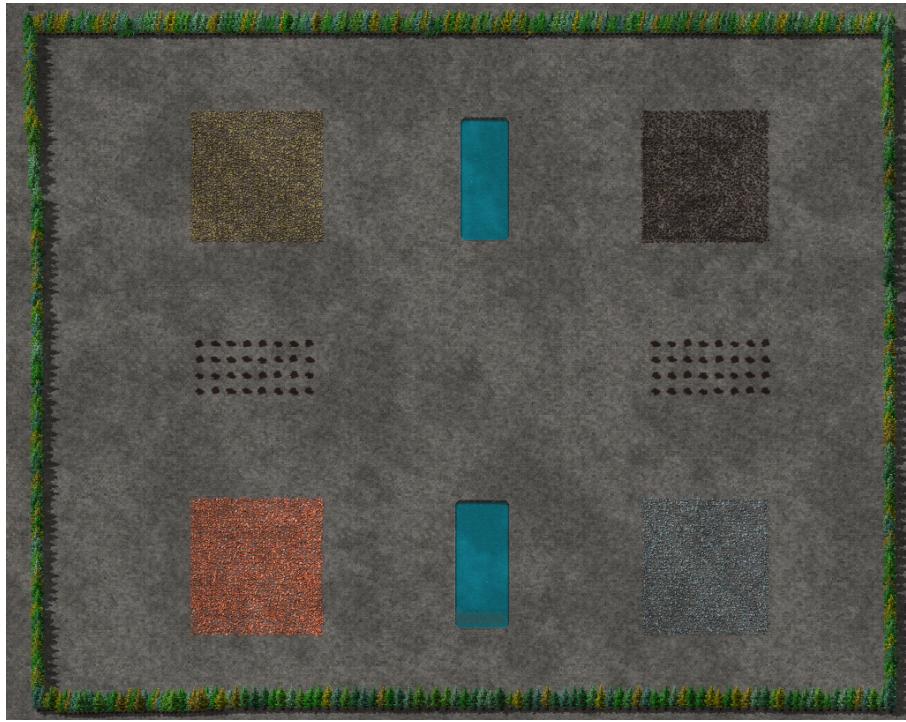


Figure 13. Overview of the laboratory map, where agents are tasked to carry out lab-play tasks

H.4. Prompt

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

```

1 # Factorio LLM Agent Instructions
2
3 ## Overview
4 You are an AI agent designed to play Factorio, specializing in:
5 - Long-horizon planning
6 - Spatial reasoning
7 - Systematic automation
8
9 ## Environment Structure
10 - Operates like an interactive Python shell
11 - Agent messages = Python programs to execute
12 - User responses = STDOUT/STDERR from REPL
13 - Interacts through 27 core API methods (to be specified)
14
15 ## Response Format
16
17 ### 1. PLANNING Stage
18 Think through each step extensively in natural language, addressing:
19 1. Error Analysis
20   - Was there an error in the previous execution?
21   - If yes, what was the problem?
22 2. Next Step Planning
23   - What is the most useful next step of reasonable size?
24   - Why is this step valuable?
25 3. Action Planning
26   - What specific actions are needed?
27   - What resources are required?
28
29 ### 2. POLICY Stage
30 Write Python code to execute the planned actions:
31   ```python
32   # Code must be enclosed in Python tags
33   your_code_here
34   ```


```

```

36 ## Best Practices
37
38 ### Modularity
39 - Create small, modular policies
40 - Each policy should have a single clear purpose
41 - Keep policies easy to debug and modify
42 - Avoid breaking existing automated structures
43 - Encapsulate working logic into functions if needed
44
45 ### Debugging & Verification
46 - Use print statements to monitor important state
47 - Implement assert statements for self-verification
48 - Use specific, parameterized assertion messages
49 - Example: `assert condition, f"Expected {expected}, got {actual}"`  

50
51 ### State Management
52 - Consider entities needed for each step
53 - Track entities across different inventories
54 - Monitor missing requirements
55 - Preserve working automated structures
56
57 ### Error Handling
58 - Fix errors as they occur
59 - Don't repeat previous steps
60 - Continue from last successful execution
61 - Avoid unnecessary state changes
62
63 ### Code Structure
64 - Write code as direct Python interpreter commands
65 - Only encapsulate reusable utility code into functions
66 - Use appropriate spacing and formatting
67
68 ## Understanding Output
69
70 ### Error Messages
71 ````stderr
72 Error: 1: ("Initial Inventory: {...}")
73 10: ("Error occurred in following lines...")
74 ````  

75 - Numbers indicate line of execution
76 - Previous lines executed successfully
77 - Fix errors at indicated line
78
79 ### Status Updates
80 ````stdout
81 23: ('Resource collection completed...')
82 78: ('Entities on map: [...]')
83 ````  

84 - Shows execution progress
85 - Provides entity status
86 - Lists warnings and conditions
87
88 ### Entity Status Checking
89 - Monitor entity 'warnings' field
90 - Check entity 'status' field
91 - Verify resource levels
92 - Track production states
93
94 ## Game Progression
95 - Think about long term objectives, and break them down into smaller, manageable steps.
96 - Advance toward more complex automation
97 - Build on previous successes
98 - Maintain efficient resource usage
99
100 ## Utility Functions
101 - Create functions to encapsulate proven, reusable logic
102 - Place function definitions before their first use
103 - Document function purpose, parameters, and return values
104 - Test functions thoroughly before relying on them
105 - Example:  

106 ````python
107 def find_idle_furnaces(entities):
108     """Find all furnaces that are not currently working.  

109
110     Args:
111         entities (list): List of entities from get_entities()  

112
113     Returns:
114         list: Furnaces with 'no_ingredients' status
115     """
116     return [e for e in entities if (

```

```

117     e.name == 'stone-furnace' and
118     e.status == EntityStatus.NO_INGREDIENTS
119   )]
120   ...
121
122 ## Data Structures
123 - Use Python's built-in data structures to organize entities
124 - Sets for unique entity collections:
125   '''python
126 working_furnaces = {e for e in get_entities()
127                     if e.status == EntityStatus.WORKING}
128   ...
129 - Dictionaries for entity mapping:
130   '''python
131 furnace_by_position = {
132     (e.position.x, e.position.y): e
133     for e in get_entities()
134     if isinstance(e, Furnace)
135   }
136   ...
137 - Lists for ordered operations:
138   '''python
139 sorted_furnaces = sorted(
140   get_entities(),
141   key=lambda e: (e.position.x, e.position.y)
142 )
143   ...
144
145 ## Important Notes
146 - Always inspect game state before making changes
147 - Consider long-term implications of actions
148 - Maintain working systems
149 - Build incrementally and verify each step
150 - 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.
151 Do not encapsulate your code in a function - just write it as if you were typing directly into the Python
     interpreter.

```

I. Rocket Silo Resource Requirements

Figure 13 shows the complexity and dependencies requires to achieve one of the end-game items, a Rocket Silo

J. Agent scaffolding details

J.1. Manual

The manual contains crucial factorio knowledge and common patterns with API examples. The full manual is provided in the supplementary code material, in src/prompts/MANUAL.md

J.2. API Schema prompt

Below is the API schema given to the agent

```

1  '''types
2  class RecipeName(enum.Enum):
3      """
4          Recipe names that can be used in the game for fluids
5      """
6      NuclearFuelReprocessing = "nuclear-fuel-reprocessing"
7      UraniumProcessing = "uranium-processing"
8      SulfuricAcid = "sulfuric-acid" # Recipe for producing sulfuric acid with a chemical plant
9      BasicOilProcessing = "basic-oil-processing" # Recipe for producing petroleum gas with a oil refinery
10     AdvancedOilProcessing = "advanced-oil-processing" # Recipe for producing petroleum gas, heavy oil and light oil
           with a oil refinery
11     CoalLiquefaction = "coal-liquefaction" # Recipe for producing petroleum gas in a oil refinery
12     HeavyOilCracking = "heavy-oil-cracking" # Recipe for producing light oil in a chemical plant
13     LightOilCracking = "light-oil-cracking" # Recipe for producing petroleum gas in a chemical plant
14     SolidFuelFromHeavyOil = "solid-fuel-from-heavy-oil" # Recipe for producing solid fuel in a chemical plant
15     SolidFuelFromLightOil = "solid-fuel-from-light-oil" # Recipe for producing solid fuel in a chemical plant
16     SolidFuelFromPetroleumGas = "solid-fuel-from-petroleum-gas" # Recipe for producing solid fuel in a chemical
           plant
17     FillCrudeOilBarrel = "fill-crude-oil-barrel"

```

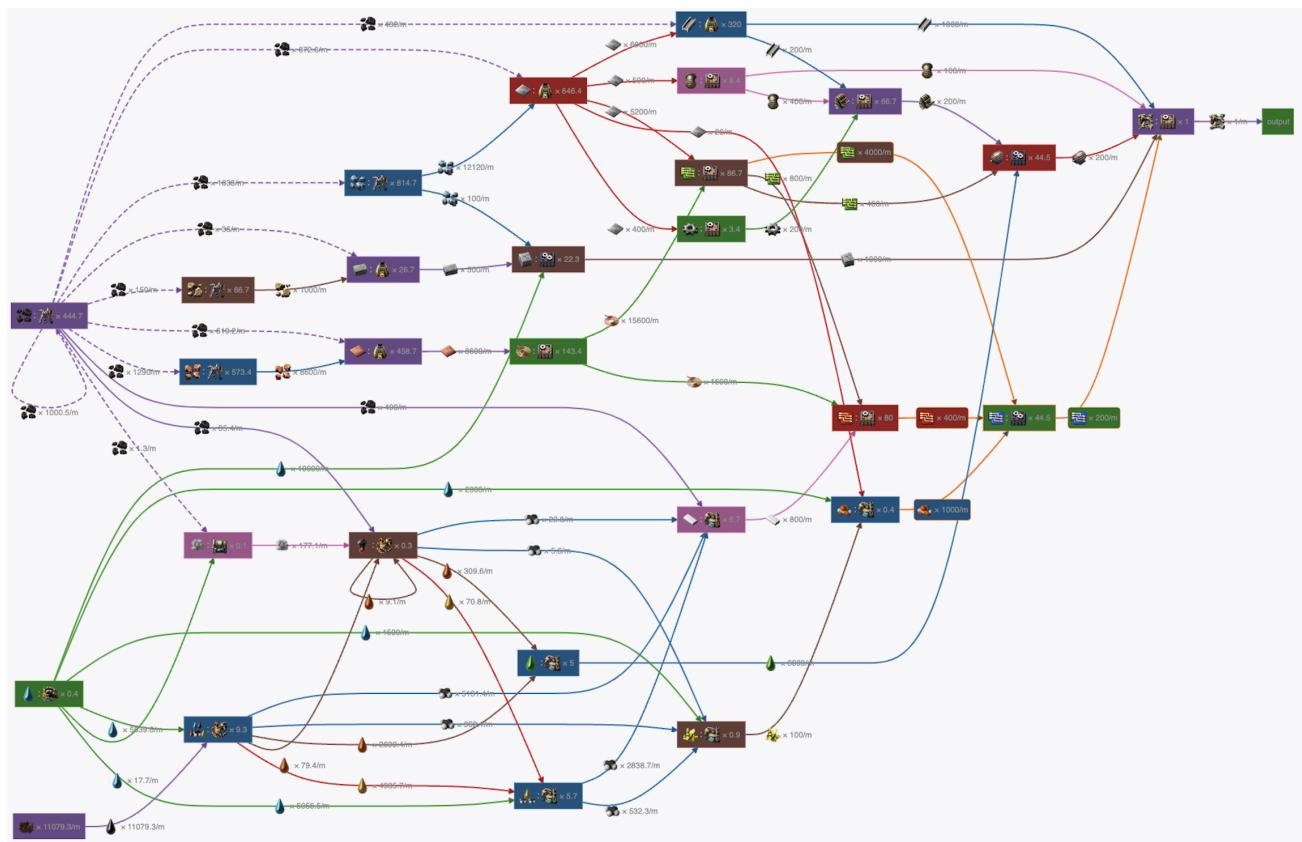


Figure 14.

```

18 FillHeavyOilBarrel = "fill-heavy-oil-barrel"
19 FillLightOilBarrel = "fill-light-oil-barrel"
20 FillLubricantBarrel = "fill-lubricant-barrel"
21 FillPetroleumGasBarrel = "fill-petroleum-gas-barrel"
22 FillSulfuricAcidBarrel = "fill-sulfuric-acid-barrel"
23 FillWaterBarrel = "fill-water-barrel"
24 EmptyCrudeOilBarrel = "empty-crude-oil-barrel"
25 EmptyHeavyOilBarrel = "empty-heavy-oil-barrel"
26 EmptyLightOilBarrel = "empty-light-oil-barrel"
27 EmptyLubricantBarrel = "empty-lubricant-barrel"
28 EmptyPetroleumGasBarrel = "empty-petroleum-gas-barrel"
29 EmptySulfuricAcidBarrel = "empty-sulfuric-acid-barrel"
30 EmptyWaterBarrel = "empty-water-barrel"
31 class Prototype(enum.Enum, metaclass=PrototypeMetaClass):
32     AssemblingMachine1 = "assembling-machine-1", AssemblingMachine
33     AssemblingMachine2 = "assembling-machine-2", AdvancedAssemblingMachine
34     AssemblingMachine3 = "assembling-machine-3", AdvancedAssemblingMachine
35     Centrifuge = "centrifuge", AssemblingMachine
36     BurnerInserter = "burner-inserter", BurnerInserter
37     FastInserter = "fast-inserter", Inserter
38     ExpressInserter = "express-inserter", Inserter
39     LongHandedInserter = "long-handed-inserter", Inserter
40     StackInserter = "stack-inserter", Inserter
41     StackFilterInserter = "stack-filter-inserter", FilterInserter
42     FilterInserter = "filter-inserter", FilterInserter
43     Inserter = "inserter", Inserter
44     BurnerMiningDrill = "burner-mining-drill", BurnerMiningDrill
45     ElectricMiningDrill = "electric-mining-drill", ElectricMiningDrill
46     StoneFurnace = "stone-furnace", Furnace
47     SteelFurnace = "steel-furnace", Furnace
48     ElectricFurnace = "electric-furnace", ElectricFurnace
49     Splitter = "splitter", Splitter
50     FastSplitter = "fast-splitter", Splitter
51     ExpressSplitter = "express-splitter", Splitter
52     Rail = "rail", Rail
53     TransportBelt = "transport-belt", TransportBelt
54     FastTransportBelt = "fast-transport-belt", TransportBelt
55     ExpressTransportBelt = "express-transport-belt", TransportBelt
56     ExpressUndergroundBelt = "express-underground-belt", UndergroundBelt
57     FastUndergroundBelt = "fast-underground-belt", UndergroundBelt
58     UndergroundBelt = "underground-belt", UndergroundBelt
59     OffshorePump = "offshore-pump", OffshorePump
60     PumpJack = "pumpjack", PumpJack
61     Pump = "pump", Pump
62     Boiler = "boiler", Boiler
63     OilRefinery = "oil-refinery", OilRefinery
64     ChemicalPlant = "chemical-plant", ChemicalPlant
65     SteamEngine = "steam-engine", Generator
66     SolarPanel = "solar-panel", SolarPanel
67     UndergroundPipe = "pipe-to-ground", Pipe
68     HeatPipe = '\heat-pipe\', Pipe
69     Pipe = "pipe", Pipe
70     SteelChest = "steel-chest", Chest
71     IronChest = "iron-chest", Chest
72     WoodenChest = "wooden-chest", Chest
73     IronGearWheel = "iron-gear-wheel", Entity
74     StorageTank = "storage-tank", StorageTank
75     SmallElectricPole = "small-electric-pole", ElectricityPole
76     MediumElectricPole = "medium-electric-pole", ElectricityPole
77     BigElectricPole = "big-electric-pole", ElectricityPole
78     Coal = "coal", None
79     Wood = "wood", None
80     Sulfur = "sulfur", None
81     IronOre = "iron-ore", None
82     CopperOre = "copper-ore", None
83     Stone = "stone", None
84     Concrete = "concrete", None
85     UraniumOre = "uranium-ore", None
86     IronPlate = "iron-plate", None # Crafting requires smelting 1 iron ore
87     IronStick = "iron-stick", None
88     SteelPlate = "steel-plate", None # Crafting requires smelting 5 iron plates
89     CopperPlate = "copper-plate", None # Crafting requires smelting 1 copper ore
90     StoneBrick = "stone-brick", None # Crafting requires smelting 2 stone
91     CopperCable = "copper-cable", None
92     PlasticBar = "plastic-bar", None
93     EmptyBarrel = "empty-barrel", None
94     Battery = "battery", None
95     SulfuricAcid = "sulfuric-acid", None
96     Uranium235 = "uranium-235", None
97     Uranium238 = "uranium-238", None
98     Lubricant = "lubricant", None

```

```

99 PetroleumGas = "petroleum-gas", None
100 AdvancedOilProcessing = "advanced-oil-processing", None # These are recipes, not prototypes.
101 CoalLiquification = "coal-liquification", None # These are recipes, not prototypes.
102 SolidFuel = "solid-fuel", None # These are recipes, not prototypes.
103 LightOil = "light-oil", None
104 HeavyOil = "heavy-oil", None
105 ElectronicCircuit = "electronic-circuit", None
106 AdvancedCircuit = "advanced-circuit", None
107 ProcessingUnit = "processing-unit", None
108 EngineUnit = "engine-unit", None
109 ElectricEngineUnit = "electric-engine-unit", None
110 Lab = "lab", Lab
111 Accumulator = "accumulator", Accumulator
112 GunTurret = "gun-turret", GunTurret
113 PiercingRoundsMagazine = "piercing-rounds-magazine", Ammo
114 FirearmMagazine = "firearm-magazine", Ammo
115 Grenade = "grenade", None
116 Radar = "radar", Entity
117 StoneWall = "stone-wall", Entity
118 Gate = "gate", Entity
119 SmallLamp = "small-lamp", Entity
120 NuclearReactor = "nuclear-reactor", Reactor
121 UraniumFuelCell = "uranium-fuel-cell", None
122 HeatExchanger = '\heat-exchanger\', HeatExchanger
123 AutomationSciencePack = "automation-science-pack", None
124 MilitarySciencePack = "military-science-pack", None
125 LogisticsSciencePack = "logistic-science-pack", None
126 ProductionSciencePack = "production-science-pack", None
127 UtilitySciencePack = "utility-science-pack", None
128 ChemicalSciencePack = "chemical-science-pack", None
129
130 ProductivityModule = "productivity-module", None
131 ProductivityModule2 = "productivity-module-2", None
132 ProductivityModule3 = "productivity-module-3", None
133 FlyingRobotFrame = "flying-robot-frame", None
134 RocketSilo = "rocket-silo", RocketSilo
135 Rocket = "rocket", Rocket
136 Satellite = "satellite", None
137 RocketPart = "rocket-part", None
138 RocketControlUnit = "rocket-control-unit", None
139 LowDensityStructure = "low-density-structure", None
140 RocketFuel = "rocket-fuel", None
141 SpaceSciencePack = "space-science-pack", None
142 BeltGroup = "belt-group", BeltGroup
143 PipeGroup = "pipe-group", PipeGroup
144 ElectricityGroup = "electricity-group", ElectricityGroup
145 def __init__(self, prototype_name, entity_class_name):
146     self.prototype_name = prototype_name
147     self.entity_class = entity_class_name
148 @property
149 def WIDTH(self):
150     return self.entity_class._width # Access the class attribute directly
151
152 @property
153 def HEIGHT(self):
154     return self.entity_class._height
155 prototype_by_name = {prototype.value[0]: prototype for prototype in Prototype}
156 prototype_by_title = {str(prototype): prototype for prototype in Prototype}
157 class Technology(Enum.Enum):
158     Automation = "automation" # Unlocks assembling machine 1
159     Automation2 = "automation-2" # Unlocks assembling machine 2
160     Automation3 = "automation-3" # Unlocks assembling machine 3
161     Logistics = "logistics" # Unlocks basic belts and inserters
162     Logistics2 = "logistics-2" # Unlocks fast belts and inserters
163     Logistics3 = "logistics-3" # Unlocks express belts and inserters
164     AdvancedElectronics = "advanced-electronics"
165     AdvancedElectronics2 = "advanced-electronics-2"
166     Electronics = "electronics"
167     ElectricEnergy = "electric-energy-distribution-1"
168     ElectricEnergy2 = "electric-energy-distribution-2"
169     SolarEnergy = "solar-energy"
170     ElectricEngineering = "electric-engine"
171     BatteryTechnology = "battery"
172     NuclearPower = "nuclear-power"
173     SteelProcessing = "steel-processing"
174     AdvancedMaterialProcessing = "advanced-material-processing"
175     AdvancedMaterialProcessing2 = "advanced-material-processing-2"
176     MilitaryScience = "military"
177     ModularArmor = "modular-armor"
178     PowerArmor = "power-armor"
179     PowerArmor2 = "power-armor-mk2"

```

```

180     NightVision = "night-vision-equipment"
181     EnergyShield = "energy-shields"
182     EnergyShield2 = "energy-shields-mk2-equipment"
183     RailwayTransportation = "railway"
184     OilProcessing = "oil-processing"
185     AdvancedOilProcessing = "advanced-oil-processing"
186     SulfurProcessing = "sulfur-processing"
187     Plastics = "plastics"
188     Lubricant = "lubricant"
189     ProductivityModule = "productivity-module"
190     ProductivityModule2 = "productivity-module-2"
191     ProductivityModule3 = "productivity-module-3"
192     Robotics = "robotics"
193     LogisticsSciencePack = "logistic-science-pack"
194     MilitarySciencePack = "military-science-pack"
195     ChemicalSciencePack = "chemical-science-pack"
196     ProductionSciencePack = "production-science-pack"
197     FastInserter = "fast-inserter"
198     StackInserter = "stack-inserter"
199     StackInserterCapacity1 = "stack-inserter-capacity-bonus-1"
200     StackInserterCapacity2 = "stack-inserter-capacity-bonus-2"
201     StorageTanks = "fluid-handling"
202     BarrelFilling = "barrel-filling"
203     Grenades = "grenades"
204     Landfill = "landfill"
205     CharacterInventorySlots = "character-inventory-slots"
206     ResearchSpeed = "research-speed"
207     SpaceScience = "space-science-pack"
208     RocketFuel = "rocket-fuel"
209     RocketControl = "rocket-control-unit"
210     LowDensityStructure = "low-density-structure"
211     RocketSiloTechnology = "rocket-silo"
212     technology_by_name = {tech.value: tech for tech in Technology}
213 class Resource:
214     Coal = "coal", ResourcePatch
215     IronOre = "iron-ore", ResourcePatch
216     CopperOre = "copper-ore", ResourcePatch
217     Stone = "stone", ResourcePatch
218     Water = "water", ResourcePatch
219     CrudeOil = "crude-oil", ResourcePatch
220     UraniumOre = "uranium-ore", ResourcePatch
221     Wood = "wood", ResourcePatch
222 class EntityStatus(Enum):
223     WORKING = '\'working\''
224     NORMAL = '\'normal\''
225     NO_POWER = '\'no_power\''
226     LOW_POWER = '\'low_power\''
227     NO_FUEL = '\'no_fuel\''
228     EMPTY = '\'empty\''
229     NOT_PLUGGED_IN_ELECTRIC_NETWORK = '\'not_plugged_in_electric_network\''
230     CHARGING = '\'charging\''
231     DISCHARGING = '\'discharging\''
232     FULLY_CHARGED = '\'fully_charged\''
233     NO_RECIPE = '\'no_recipe\''
234     NO_INGREDIENTS = '\'no_ingredients\''
235     NOT_CONNECTED = '\'not_connected\''
236     NO_INPUT_FLUID = '\'no_input_fluid\''
237     NO_RESEARCH_IN_PROGRESS = '\'no_research_in_progress\''
238     NO_MINABLE_RESOURCES = '\'no_minable_resources\''
239     LOW_INPUT_FLUID = '\'low_input_fluid\''
240     FLUID_INGREDIENT_SHORTAGE = '\'fluid_ingredient_shortage\''
241     FULL_OUTPUT = '\'full_output\''
242     FULL_BURNT_RESULT_OUTPUT = '\'full_burnt_result_output\''
243     ITEM_INGREDIENT_SHORTAGE = '\'item_ingredient_shortage\''
244     MISSING_REQUIRED_FLUID = '\'missing_required_fluid\''
245     MISSING_SCIENCE_PACKS = '\'missing_science_packs\''
246     WAITING_FOR_SOURCE_ITEMS = '\'waiting_for_source_items\''
247     WAITING_FOR_SPACE_IN_DESTINATION = '\'waiting_for_space_in_destination\''
248     PREPARING_ROCKET_FOR_LAUNCH = '\'preparing_rocket_for_launch\''
249     WAITING_TO_LAUNCH_ROCKET = '\'waiting_to_launch_rocket\''
250     LAUNCHING_ROCKET = '\'launching_rocket\''
251     NO_AMMO = '\'no_ammo\''
252     LOW_TEMPERATURE = '\'low_temperature\''
253     NOT_CONNECTED_TO_RAIL = '\'not_connected_to_rail\''
254     def __repr__(self):
255         def from_string(cls, status_string):
256             def from_int(cls, status_int):
257                 class Inventory(BaseModel):
258                     class Config:
259                         populate_by_name = True
260                         arbitrary_types_allowed = True

```

```

261     def __init__(self):
262         def __getitem__(self, key: \'Prototype\', default) -> int:
263             def get(self, key: \'Prototype\', default) -> int:
264                 def __setitem__(self, key: \'Prototype\', value: int) -> None:
265                     def items(self):
266                         def __repr__(self) -> str:
267                             def __str__(self) -> str:
268                             def __len__(self) -> int:
269                             def keys(self):
270                             def values(self):
271             class Direction(Enum):
272                 UP = 0
273                 NORTH = 0
274                 RIGHT = 2
275                 EAST = 2
276                 DOWN = 4
277                 SOUTH = 4
278                 LEFT = 6
279                 WEST = 6
280                 def __repr__(self):
281                     def from_string(cls, direction_string):
282             class Position(BaseModel):
283                 x: float
284                 y: float
285                 def _parse_positional_args(cls, v):
286                     def __init__(self):
287                         def parse_args(cls, values):
288                             def __hash__(self):
289                                 def __add__(self, other) -> \'Position\':
290                                 def __sub__(self, other) -> \'Position\':
291                                 def is_close(self, a: \'Position\', tolerance: float) -> bool:
292                                 def distance(self, a: \'Position\') -> float:
293                                 def _modifier(self, args):
294                                     def above(self) -> \'Position\':
295                                     def up(self) -> \'Position\':
296                                     def below(self) -> \'Position\':
297                                     def down(self) -> \'Position\':
298                                     def left(self) -> \'Position\':
299                                     def right(self) -> \'Position\':
300                                     def to_bounding_box(self, other: \'Position\') -> \'BoundingBox\':
301                                     def __eq__(self, other) -> bool:
302             class IndexedPosition(Position):
303                 type: str
304                 def __new__(cls):
305                     def __init__(self):
306                         def __hash__(self):
307             class EntityInfo(BaseModel):
308                 name: str
309                 direction: int
310                 position: Position
311                 start_position: Optional[Position]
312                 end_position: Optional[Position]
313                 quantity: Optional[int]
314                 warning: Optional[str]
315                 contents: Dict[str, int]
316                 status: EntityStatus
317             class InspectionResults(BaseModel):
318                 entities: List[EntityInfo]
319                 player_position: Tuple[float, float]
320                 radius: float
321                 time_elapsed: float
322                 def get_entity(self, prototype: \'Prototype\') -> Optional[EntityInfo]:
323                 def get_entities(self, prototype: \'Prototype\') -> List[EntityInfo]:
324             class BoundingBox(BaseModel):
325                 left_top: Position
326                 right_bottom: Position
327                 left_bottom: Position
328                 right_top: Position
329                 def center(self) -> Position:
330                 def width(self) -> float:
331                 def height(self) -> float:
332             class BuildingBox(BaseModel):
333                 height: int
334                 width: int
335             class ResourcePatch(BaseModel):
336                 name: str
337                 size: int
338                 bounding_box: BoundingBox
339             class Dimensions(BaseModel):
340                 width: float
341                 height: float

```

```

342 class TileDimensions(BaseModel):
343     tile_width: float
344     tile_height: float
345 class Ingredient(BaseModel):
346     name: str
347     count: Optional[int]
348     type: Optional[Literal['fluid', 'item']]
349 class Product(Ingredient):
350     probability: Optional[float]
351 class Recipe(BaseModel):
352     name: Optional[str]
353     ingredients: Optional[List[Ingredient]]
354     products: Optional[List[Product]]
355     energy: Optional[float]
356     category: Optional[str]
357     enabled: bool
358 class BurnerType(BaseModel):
359 """
360 Type of entity that burns fuel
361 """
362     class Config:
363         arbitrary_types_allowed = True
364         fuel: Inventory
365 class EntityCore(BaseModel):
366     name: str
367     direction: Direction
368     position: Position
369     def __repr__(self):
370 class Entity(EntityCore):
371 """
372 Base class for all entities in the game.
373 """
374     id: Optional[int]
375     energy: float
376     type: Optional[str]
377     dimensions: Dimensions
378     tile_dimensions: TileDimensions
379     prototype: Any
380     health: float
381     warnings: List[str]
382     status: EntityStatus
383     def __repr__(self) -> str:
384     def _get_prototype(self):
385     def width(cls):
386         def height(cls):
387 class StaticEntity(Entity):
388 """
389 A static (non-moving) entity in the game.
390 """
391     neighbours: Optional[Union[Dict, List[EntityCore]]]
392 class Rail(Entity):
393 """
394 Railway track for trains.
395 """
396     _height: float
397     _width: float
398 class Splitter(Entity):
399 """
400 A belt splitter that divides item flow between outputs.
401 """
402     input_positions: List[Position]
403     output_positions: List[Position]
404     inventory: List[Inventory]
405     _height: float
406     _width: float
407 class TransportBelt(Entity):
408 """
409 A conveyor belt for moving items.
410 """
411     input_position: Position
412     output_position: Position
413     inventory: Inventory
414     is_terminus: bool
415     is_source: bool
416     _height: float
417     _width: float
418     def __repr__(self):
419     def __hash__(self):
420         def __eq__(self, other):
421 class Electric(BaseModel):
422 """

```

```

423 Base class for entities that interact with the power grid.
424 """
425     electrical_id: Optional[int]
426 class ElectricalProducer(Electric, Entity):
427 """
428 An entity that generates electrical power.
429 """
430     production: Optional[Any]
431     energy_source: Optional[Any]
432     electric_output_flow_limit: Optional[float]
433 class EnergySource(BaseModel):
434     buffer_capacity: str
435     input_flow_limit: str
436     output_flow_limit: str
437     drain: str
438 class Accumulator(StaticEntity, Electric):
439 """
440 Represents an energy storage device
441 """
442     energy_source: Optional[EnergySource]
443     _height: float
444     _width: float
445 class Inserter(StaticEntity, Electric):
446 """
447 Represents an inserter that moves items between entities.
448     Requires electricity to power
449 """
450     pickup_position: Optional[Position]
451     drop_position: Position
452     _width: float
453     _height: float
454 class Filtered(BaseModel):
455     filter: Optional[Any]
456 class UndergroundBelt(TransportBelt):
457 """
458 An underground section of transport belt.
459 """
460     is_input: bool
461     connected_to: Optional[int]
462     _height: float
463     _width: float
464 class MiningDrill(StaticEntity):
465 """
466 Base class for mining drills that extract resources.
467     The direction of the drill is where the drop_position is oriented towards
468 """
469     drop_position: Position
470     resources: List[Ingredient]
471 class ElectricMiningDrill(MiningDrill, Electric):
472 """
473 An electrically-powered mining drill.
474 """
475     _height: float
476     _width: float
477 class BurnerInserter(Inserter, BurnerType):
478 """
479 An inserter powered by burnable fuel.
480 """
481     _height: float
482     _width: float
483 class BurnerMiningDrill(MiningDrill, BurnerType):
484 """
485 A mining drill powered by burnable fuel.
486 """
487     _width = 2
488     _height = 2
489 class Ammo(BaseModel):
490     name: str
491     magazine_size: Optional[int]
492     reload_time: Optional[float]
493 class GunTurret(StaticEntity):
494     turret_ammo: Inventory
495     _height: float
496     _width: float
497     kills: Optional[int]
498 class AssemblingMachine(StaticEntity, Electric):
499 """
500 A machine that crafts items from ingredients.
501     Requires power to operate
502 """
503     recipe: Optional[Recipe]

```

```

504     assembling_machine_input: Inventory
505     assembling_machine_output: Inventory
506     assembling_machine_modules: Inventory
507     _height: float
508     _width: float
509 class FluidHandler(StaticEntity):
510 """
511 Base class for entities that handle fluids
512 """
513     connection_points: List[Position]
514     fluid_box: Optional[Union[dict, list]]
515     fluid_systems: Optional[Union[dict, list]]
516 class AdvancedAssemblingMachine(FluidHandler, AssemblingMachine):
517 """
518 A second and third tier assembling machine that can handle fluids.
519     Requires power to operate
520     A recipe first needs to be set and then the input fluid source can be connected with pipes
521 """
522     _height: float
523     _width: float
524 class MultiFluidHandler(StaticEntity):
525 """
526 Base class for entities that handle multiple fluid types.
527 """
528     input_fluids: List[str]
529     output_fluids: List[str]
530     input_connection_points: List[IndexedPosition]
531     output_connection_points: List[IndexedPosition]
532     fluid_box: Optional[Union[dict, list]]
533     fluid_systems: Optional[Union[dict, list]]
534 class FilterInserter(Inserter, Filtered):
535 """
536 A inserter that only moves specific items
537 """
538     _height: float
539     _width: float
540 class ChemicalPlant(MultiFluidHandler, AssemblingMachine):
541 """
542 Represents a chemical plant that processes fluid recipes.
543     Requires powering and accepts input fluids (from storage tanks etc) and solids (with inserters)
544     Outputs either:
545         solids (battery, plastic) that need to be extracted with inserters
546         fluids (sulfuric acid, oil) that need to be extracted with pipes
547     IMPORTANT: First a recipe needs to be set and then the fluid sources can be connected to the plant
548 """
549     _height: float
550     _width: float
551 class OilRefinery(MultiFluidHandler, AssemblingMachine):
552 """
553 An oil refinery for processing crude oil into products.
554     Requires powering and accepts input fluids (from pumpjacks, storage tanks etc) and solids
555     First a recipe needs to be set and then the fluid sources can be connected to the refinery
556 """
557     _height: float
558     _width: float
559 class PumpJack(MiningDrill, FluidHandler, Electric):
560 """
561 A pump jack for extracting crude oil. Requires electricity
562     This needs to be placed on crude oil and oil needs to be extracted with pipes
563     Oil can be sent to a storage tank, oil refinery or a chemical plant
564     Oil can also be sent to assmbling machine to be made into oil barrels
565     Important: The PumpJack needs to be placed on exact crude oil tiles
566 """
567     _height: float
568     _width: float
569 class SolarPanel(ElectricalProducer):
570 """
571 A solar panel for generating power from sunlight.
572     This entity generated power during the day
573     Thus it can be directly connected to a entity to power it
574 """
575     _height: float
576     _width: float
577 class Boiler(FluidHandler, BurnerType):
578 """
579 A boiler that heats water into steam.
580 """
581     steam_output_point: Optional[Position]
582     _height: float
583     _width: float

```

```

585 class HeatExchanger(Boiler):
586     """
587     A nuclear heat exchanger that converts water to steam.
588     """
589     class Generator(FluidHandler, StaticEntity):
590         """
591         A steam generator that produces electricity.
592         """
593         _height: float
594         _width: float
595     class Pump(FluidHandler, Electric):
596         """
597         An electrically-powered fluid pump.
598         """
599         _height: float
600         _width: float
601     class OffshorePump(FluidHandler):
602         """
603         A pump that extracts water from water tiles.
604         Can be used in power generation setups and to supply water to chemical plants and oil refineries.
605         """
606         _height: float
607         _width: float
608     class ElectricityPole(Entity, Electric):
609         """
610         A power pole for electricity distribution.
611         """
612         flow_rate: float
613         _height: float
614         _width: float
615         def __hash__(self):
616     class Furnace(Entity, BurnerType):
617         """
618         A furnace for smelting items
619         """
620         furnace_source: Inventory
621         furnace_result: Inventory
622         _height: float
623         _width: float
624     class ElectricFurnace(Entity, Electric):
625         """
626         An electrically-powered furnace.
627         """
628         furnace_source: Inventory
629         furnace_result: Inventory
630         _height: float
631         _width: float
632     class Chest(Entity):
633         """
634         A storage chest.
635         """
636         inventory: Inventory
637         _height: float
638         _width: float
639     class StorageTank(FluidHandler):
640         """
641         A tank for storing fluids.
642         Can be used for inputs and outputs of chemical plants and refineries.
643         Also can store water from offshore pumps.
644         """
645         _height: float
646         _width: float
647     class RocketSilo(StaticEntity, Electric):
648         """
649         A rocket silo that can build and launch rockets.
650         """
651         rocket_parts: int
652         rocket_inventory: Inventory
653         rocket_progress: float
654         launch_count: int
655         _width: float
656         _height: float
657         def __repr__(self) -> str:
658     class Rocket(Entity):
659         """
660         A rocket that can be launched from a silo.
661         """
662         payload: Optional[Inventory]
663         launch_progress: float
664         def __repr__(self) -> str:
665     class Lab(Entity, Electric):

```

```

666 """
667 A research laboratory.
668 """
669     lab_input: Inventory
670     lab_modules: Inventory
671     research: Optional[Any]
672     _height: float
673     _width: float
674     def __repr__(self) -> str:
675 class Pipe(Entity):
676 """
677 A pipe for fluid transport
678 """
679     fluidbox_id: int
680     flow_rate: float
681     contents: float
682     fluid: Optional[str]
683     _height: float
684     _width: float
685 class Reactor(StaticEntity):
686 """
687 A nuclear reactor
688 """
689     _height: float
690     _width: float
691 class EntityGroup(BaseModel):
692     id: int
693     status: EntityStatus
694     position: Position
695     name: str
696 class WallGroup(EntityGroup):
697 """
698 A wall
699 """
700     name: str
701     entities: List[Entity]
702 class BeltGroup(EntityGroup):
703 """
704 A connected group of transport belts.
705 """
706     belts: List[TransportBelt]
707     inputs: List[Entity]
708     outputs: List[Entity]
709     inventory: Inventory
710     name: str
711     def __repr__(self) -> str:
712     def __str__(self):
713 class PipeGroup(EntityGroup):
714 """
715 A connected group of pipes.
716 """
717     pipes: List[Pipe]
718     name: str
719     def __repr__(self) -> str:
720     def __str__(self):
721 class ElectricityGroup(EntityGroup):
722 """
723 Represents a connected power network.
724 """
725     name: str
726     poles: List[ElectricityPole]
727     def __repr__(self) -> str:
728     def __hash__(self):
729     def __str__(self):
730 ```
731 ```methods
732 can_place_entity(entity: Prototype, direction: Direction = <Direction.UP: 0>, position: Position = Position(x=0.0,
733     y=0.0)) -> bool
734 """
735 Tests to see if an entity can be placed at a given position
736 :param entity: Entity to place from inventory
737 :param direction: Cardinal direction to place entity
738 :param position: Position to place entity
739 :return: True if entity can be placed at position, else False
740 """
741 craft_item(entity: Prototype, quantity: int = 1) -> int
742 """
743 Craft an item from a Prototype if the ingredients exist in your inventory.
744 :param entity: Entity to craft
745 :param quantity: Quantity to craft

```

```

746 :return: Number of items crafted
747 """
748
749 extract_item(entity: Prototype, source: Union[Position, Entity], quantity=5) -> int
750 """
751 Extract an item from an entity's inventory at position (x, y) if it exists on the world.
752 :param entity: Entity prototype to extract, e.g Prototype.IronPlate
753 :param source: Entity or position to extract from
754 :param quantity: Quantity to extract
755 :example extract_item(Prototype.IronPlate, stone_furnace.position, 5)
756 :example extract_item(Prototype.CopperWire, stone_furnace, 5)
757 :return: The number of items extracted.
758 """
759
760 get_connection_amount(source: Union[Position, Entity, EntityGroup], target: Union[Position, Entity, EntityGroup],
761     connection_type: Prototype = <Prototype.Pipe: ('pipe', <class 'Pipe'>)>) -> int
762 """
763 Calculate the number of connecting entities needed to connect two entities, positions or groups.
764 :param source: First entity or position
765 :param target: Second entity or position
766 :param connection_type: a Pipe, TransportBelt or ElectricPole
767 :return: A integer representing how many entities are required to connect the source and target entities
768 """
769 get_entities(entities: Union[Set[Prototype], Prototype] = set(), position: Position = None, radius: float = 1000)
770     -> List[Entity]
771 """
772 Get entities within a radius of a given position.
773 :param entities: Set of entity prototypes to filter by. If empty, all entities are returned.
774 :param position: Position to search around. Can be a Position object or "player" for player's position.
775 :param radius: Radius to search within.
776 :return: Found entities
777 """
778 get_entity(entity: Prototype, position: Position) -> Entity
779 """
780 Retrieve a given entity object at position (x, y) if it exists on the world.
781 :param entity: Entity prototype to get, e.g Prototype.StoneFurnace
782 :param position: Position where to look
783 :return: Entity object
784 """
785
786 get_prototype_recipe(prototype: Union[Prototype, RecipeName, str]) -> Recipe
787 """
788 Get the recipe (cost to make) of the given entity prototype.
789 :param prototype: Prototype to get recipe from
790 :return: Recipe of the given prototype
791 """
792
793 get_research_progress(technology: Optional[Technology] = None) -> List[Ingredient]
794 """
795 Get the progress of research for a specific technology or the current research.
796 :param technology: Optional technology to check. If None, checks current research.
797 :return: The remaining ingredients to complete the research
798 """
799
800 get_resource_patch(resource: Resource, position: Position, radius: int = 10) -> Optional[ResourcePatch]
801 """
802 Get the resource patch at position (x, y) if it exists in the radius.
803 if radius is set to 0, it will only check the exact position for this resource patch.
804 :param resource: Resource to get, e.g Resource.Coal
805 :param position: Position to get resource patch
806 :param radius: Radius to search for resource patch
807 :example coal_patch_at_origin = get_resource_patch(Resource.Coal, Position(x=0, y=0))
808 :return: ResourcePatch if found, else None
809 """
810
811 harvest_resource(position: Position, quantity=1, radius=10) -> int
812 """
813 Harvest a resource at position (x, y) if it exists on the world.
814 :param position: Position to harvest resource
815 :param quantity: Quantity to harvest
816 :example harvest_resource(nearest(Resource.Coal), 5)
817 :example harvest_resource(nearest(Resource.Stone), 5)
818 :return: The quantity of the resource harvested
819 """
820
821 insert_item(entity: Prototype, target: Union[Entity, EntityGroup], quantity=5) -> Entity
822 """
823 Insert an item into a target entity's inventory
824 :param entity: Type to insert from inventory

```

```

825 :param target: Entity to insert into
826 :param quantity: Quantity to insert
827 :return: The target entity inserted into
828 """
829
830 inspect_inventory(entity=None) -> Inventory
831 """
832 Inspects the inventory of the given entity. If no entity is given, inspect your own inventory.
833 :param entity: Entity to inspect
834 :return: Inventory of the given entity
835 """
836
837 launch_rocket(silo: Union[Position, RocketSilo]) -> RocketSilo
838 """
839 Launch a rocket.
840 :param silo: Rocket silo
841 :return: Your final position
842 """
843
844 move_to(position: Position, laying: Prototype = None, leading: Prototype = None) -> Position
845 """
846 Move to a position.
847 :param position: Position to move to.
848 :return: Your final position
849 """
850
851 nearest(type: Union[Prototype, Resource]) -> Position
852 """
853 Find the nearest entity or resource to your position.
854 :param type: Entity or resource type to find
855 :return: Position of nearest entity or resource
856 """
857
858 nearest_buildable(entity: Prototype, building_box: BuildingBox, center_position: Position, **kwargs) -> BoundingBox
859 """
860 Find the nearest buildable area for an entity.
861
862 :param entity: Prototype of the entity to build.
863 :param building_box: The building box denoting the area of location that must be placeable.
864 :param center_position: The position to find the nearest area where building box fits
865 :return: BoundingBox of the nearest buildable area or None if no such area exists.
866 """
867
868 pickup_entity(entity: Union[Entity, Prototype, EntityGroup], position: Optional[Position] = None) -> bool
869 """
870 Pick up an entity if it exists on the world at a given position.
871 :param entity: Entity prototype to pickup, e.g Prototype.IronPlate
872 :param position: Position to pickup entity
873 :return: True if the entity was picked up successfully, False otherwise.
874 """
875
876 place_entity(entity: Prototype, direction: Direction = <Direction.UP: 0>, position: Position = Position(x=0.0,
877     y=0.0), exact: bool = True) -> Entity
878 """
879 Places an entity e at local position (x, y) if you have it in inventory.
880 :param entity: Entity to place
881 :param direction: Cardinal direction to place
882 :param position: Position to place entity
883 :param exact: If True, place entity at exact position, else place entity at nearest possible position
884 :return: Entity object
885 """
886 place_entity_next_to(entity: Prototype, reference_position: Position = Position(x=0.0, y=0.0), direction: Direction
887     = <Direction.RIGHT: 2>, spacing: int = 0) -> Entity
888 """
889 Places an entity next to an existing entity, with an optional space in-between (0 space means adjacent).
890 In order to place something with a gap, you must increase the spacing parameter.
891 :param entity: Entity to place
892 :param reference_position: Position of existing entity or position to place entity next to
893 :param direction: Direction to place entity from reference_position
894 :param spacing: Space between entity and reference_position
895 :example: place_entity_next_to(Prototype.WoodenChest, Position(x=0, y=0), direction=Direction.UP, spacing=1)
896 :return: Entity placed
897 """
898 print(*args)
899 """
900 Adds a string to stdout
901 :param args:
902 :return:
903 """
904

```

```

904     rotate_entity(entity: Entity, direction: Direction = <Direction.UP: 0>) -> Entity
905     """
906     Rotate an entity to a specified direction
907     :param entity: Entity to rotate
908     :param direction: Direction to rotate
909     :example rotate_entity(iron_chest, Direction.UP)
910     :return: Returns the rotated entity
911     """
912
913
914     set_entity_recipe(entity: Entity, prototype: Union[Prototype, RecipeName]) -> Entity
915     """
916     Sets the recipe of an given entity.
917     :param entity: Entity to set recipe
918     :param prototype: The prototype to create, or a recipe name for more complex processes
919     :return: Entity that had its recipe set
920     """
921
922     set_research(*args, **kwargs)
923     """
924     Call self as a function.
925     """
926
927     set_research(technology: Technology) -> List[Ingredient]
928     """
929     Set the current research technology for the player's force.
930     :param technology: Technology to research
931     :return: Required ingredients to research the technology.
932     """
933
934     shift_entity(entity: Entity, direction: Union[Direction, Direction], distance: int = 1) -> Entity
935     """
936     Calculate the number of connecting entities needed to connect two entities, positions or groups.
937     :param source: First entity or position
938     :param target: Second entity or position
939     :param connection_type: a Pipe, TransportBelt or ElectricPole
940     :return: A integer representing how many entities are required to connect the source and target entities
941     """
942
943     sleep(seconds: int) -> bool
944     """
945     Sleep for up to 15 seconds before continuing. Useful for waiting for actions to complete.
946     :param seconds: Number of seconds to sleep.
947     :return: True if sleep was successful.
948     """
949     """

```

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 doesn't rely on past game states to make decisions

```

1 Assistant message
2 '''python
3 water_pos = nearest(Resource.Water)
4 print(f"Found water source at {water_pos}")
5
6 # Place offshore pump
7 move_to(water_pos)
8 offshore_pump = place_entity(Prototype.OffshorePump, position=water_pos)
9 print(f"Placed offshore pump at {offshore_pump.position}")
10
11 boiler = place_entity_next_to(Prototype.Boiler, reference_position=offshore_pump.position, spacing = 4, direction =
12     Direction.RIGHT)
13 print(f"Placed boiler at {boiler.position}")
14 steam_engine = place_entity_next_to(Prototype.SteamEngine, reference_position=boiler.position, spacing = 5,
15     direction = Direction.RIGHT)
16 print(f"Placed steam engine at {steam_engine.position}")
17 # Connect offshore pump to boiler with pipes
18 pump_to_boiler = connect_entities(offshore_pump.position, boiler.position, Prototype.Pipe)
19 print(f"Connected offshore pump to boiler with pipes: {pump_to_boiler}")
20
21 # Connect boiler to steam engine with pipes
22 boiler_to_engine = connect_entities(boiler.position, steam_engine.position, Prototype.Pipe)

```

```

21 print(f"Connected boiler to steam engine with pipes: {boiler_to_engine}")
22 print(f"Current inventory {inspect_inventory()}")
23 print(f"Updated entities on the map: {get_entities()}")
24 ``
25
26 User message
27 27015: ('Found water source at x=-12.0 y=0.0',)
28 27015: ('Placed offshore pump at x=-10.5 y=-0.5',)
29 27015: ('Placed boiler at x=-5.0 y=-0.5',)
30 27015: ('Placed steam engine at x=3.5 y=-0.5',)
31 27015: ('Connected offshore pump to boiler with pipes: [
32     PipeGroup(fluid_system=1, position=x=-6.5 y=-2.5, status=EntityStatus.WORKING, pipes=[7 pipes], fluid=water)]',)
33 27015: ('Connected boiler to steam engine with pipes: [ PipeGroup(fluid_system=3, position=x=-2.5 y=-0.5,
34     status=EntityStatus.EMPTY, pipes=[5 pipes])]',)
34 27015: ("Current inventory {'wooden-chest': 10, 'transport-belt': 200, 'burner-inserter': 10,
35     'small-electric-pole': 200, 'pipe': 88, 'boiler': 3, 'steam-engine': 1, 'burner-mining-drill': 10,
36     'electric-mining-drill': 10, 'offshore-pump': 2, 'stone-furnace': 5, 'assembling-machine-1': 5, 'coal': 200,
37     'stone': 50, 'iron-ore': 62, 'iron-plate': 19, 'copper-plate': 52, 'iron-gear-wheel': 22,
38     'electronic-circuit': 99}",)
39 27015: ("Updated entities on the map: <STALE_ENTITY_DATA OMITTED>")
40
41 Assistant message
42 ``python
43 # move to iron position
44 iron_pos = nearest(Resource.IronOre)
45 move_to(iron_pos)
46 # place a electric mining drill
47 drill = place_entity(Prototype.ElectricMiningDrill, position = iron_pos)
48 print(f"Put a drill to mine iron at {drill.position}")
49
50 # power the drill
51 steam_engine = get_entity(Prototype.SteamEngine, position = Position(x = 3.5, y=-0.5))
52 poles = connect_entities(steam_engine,drill,Prototype.SmallElectricPole)
53 print(f"Used poles {poles} to power drill at {drill.position}")
54
55 User Message
56 27015: ('Put a drill to mine iron at x=-11.5 y=19.5',)
57 27015: ('Used poles [ElectricityGroup(id=18,
58     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)],
59     voltage=0)] to power drill at x=-11.5 y=19.5',)
60 27015: ("Current inventory {'wooden-chest': 10, 'transport-belt': 200, 'burner-inserter': 10,
61     'small-electric-pole': 191, 'pipe': 88, 'boiler': 3, 'steam-engine': 1, 'burner-mining-drill': 10,
62     'electric-mining-drill': 9, 'offshore-pump': 2, 'stone-furnace': 5, 'assembling-machine-1': 5, 'coal': 200,
63     'stone': 50, 'iron-ore': 62, 'iron-plate': 19, 'copper-plate': 52, 'iron-gear-wheel': 22,
64     'electronic-circuit': 99}",)
65 27015: ("Updated entities on the map: [
66 OffshorePump(name='offshore-pump', position=Position(x=-10.5, y=-0.5), direction=Direction.RIGHT, energy=0.0,
67     tile_dimensions=TileDimensions(tile_width=1.0, tile_height=1.0), status=EntityStatus.WORKING,
68     connection_points=[Position(x=-9.5, y=-0.5)], fluid_box=[{'name': 'water', 'amount': 100, 'temperature': 15}],
69     fluid_systems=[49]),
70 Boiler(fuel={}, name='boiler', position=Position(x=-5.0, y=-0.5), direction=Direction.RIGHT, energy=0.0,
71     tile_dimensions=TileDimensions(tile_width=3.0, tile_height=2.0), warnings=['out of fuel'],
72     status=EntityStatus.NO_FUEL, connection_points=[Position(x=-5.5, y=-2.5), Position(x=-5.5, y=1.5)],
73     fluid_box=[{'name': 'water', 'amount': 200, 'temperature': 15}], fluid_systems=[49],
74     steam_output_point=Position(x=-3.0, y=-0.5)),
75 Generator(electrical_id=18, name='steam-engine', position=Position(x=3.5, y=-0.5), direction=Direction.RIGHT,
76     energy=0.0, tile_dimensions=TileDimensions(tile_width=3.0, tile_height=5.0), warnings=['not receiving
77     electricity', 'no input liquid', 'No fluid present in connections'], status=EntityStatus.NOT_CONNECTED,
78     connection_points=[Position(x=6.0, y=-0.5), Position(x=1.0, y=-0.5)], fluid_box=[], fluid_systems=[]),
79 ElectricMiningDrill(electrical_id=18, name='electric-mining-drill', position=Position(x=-11.5, y=19.5),
80     direction=Direction.UP, energy=0.0, tile_dimensions=TileDimensions(tile_width=3.0, tile_height=3.0),
81     warnings=['not receiving electricity'], status=EntityStatus.NO_POWER, drop_position=Position(x=-11.5, y=17.5)),
82 PipeGroup(fluid_system=49, position=x=-6.5 y=-2.5, status=EntityStatus.FULL_OUTPUT, pipes=[7 pipes], fluid=water),
83 PipeGroup(fluid_system=51, position=x=-2.5 y=-0.5, status=EntityStatus.EMPTY, pipes=[5 pipes]),
84 ElectricityGroup(id=18,
85     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)],
86     voltage=0)",)

```