HDDLGym: A Tool for Studying Multi-Agent Hierarchical Problems Defined in HDDL with OpenAI Gym

Ngoc La[1], **Ruaridh Mon-Williams**[2]
[1]MIT
[2]University of Edinburgh
ntmla@mit.edu, ruaridh.mw@ed.ac.uk

Abstract

In recent years, reinforcement learning (RL) methods have been widely tested using tools like OpenAI Gym, although many tasks in these environments could also benefit from hierarchical planning. However, there is a lack of tools that enable the seamless integration of hierarchical planning with RL. Hierarchical Domain Definition Language (HDDL), used in classical planning, introduces a structured approach well-suited for model-based RL to address this gap. To facilitate this integration, we introduce HDDLGym, a Python-based tool that automatically generates OpenAI Gym environments from HDDL domains and problems. HDDLGym serves as a link between RL and hierarchical planning, supporting multi-agent scenarios and enabling collaborative planning among agents. This paper provides an overview of HDDLGym's design and implementation, highlighting the challenges and design choices involved in integrating HDDL with the Gym interface and applying RL policies to support hierarchical planning. We also provide detailed instructions and demonstrations for using the HDDLGym framework, including how to work with existing HDDL domains and problems from International Planning Competitions, such as the Transport domain. Additionally, we offer guidance on creating new HDDL domains for multi-agent scenarios and demonstrate the practical use of HDDLGym in the Overcooked domain. By leveraging the advantages of HDDL and Gym, HDDLGym aims to be a valuable tool for studying RL in hierarchical planning, particularly in multi-agent contexts.

Code — https://github.com/HDDLGym/HDDLGym

1 Introduction

Hierarchical planning is essential for addressing complex, long-horizon planning problems by decomposing them into smaller, manageable subproblems. In reinforcement learning (RL), hierarchical strategies can guide exploration along specific pathways, potentially enhancing learning efficiency. However, implementing RL policies within hierarchical frameworks often requires custom modifications to the original environments to incorporate high-level actions (Wu et al. 2021; Liu et al. 2017; Xiao, Hoffman, and Amato 2020). For example, in a Bayesian inference study using the Overcooked game, subtasks are integrated as high-level actions through specific rules embedded in the system codebase (Wu et al. 2021). Similarly, several RL studies use author-defined high-level actions, or macro-actions, to organize complex tasks (Liu et al. 2017; Xiao, Hoffman, and Amato 2020). While these studies highlight the benefits of hierarchical approaches in complex scenarios, the additional programming required to integrate hierarchical layers can make it challenging for external users to modify or implement alternative high-level strategies. This limitation reduces users' flexibility to implement diverse hierarchical strategies tailored to their specific requirements.

The Hierarchical Domain Definition Language (HDDL) (Höller et al. 2020) is an extension of Planning Domain Definition Language (PDDL) (McDermott et al. 1998) that incorporates hierarchical task networks (HTN) (Erol, Hendler, and Nau 1994). HDDL provides a standardized language for hierarchical planning systems and is supported by extensive documentation as well as a variety of domains and problems. Many of these resources are sourced from the hierarchical task network tracks of the International Planning Competitions (IPC-HTN) (IPC 2023 HTN Tracks). HDDL's intuitive and flexible design also allows users to define or modify problem-solving approaches by adjusting the hierarchical task networks to suit their specific needs. To leverage HDDL's strengths in studying RL within hierarchical planning problems, we present HDDLGym — a framework that integrates HDDL with OpenAI Gym (Brockman et al. 2016), a standardized RL interface. HDDLGym is a Python-based tool that automatically generates Gym environments from HDDL domain and problem files.

Multi-agent contexts are a key area in automated planning and hierarchical planning research. While HDDL is not inherently designed for multi-agent systems, multi-agent features have been explored in planning formalisms like MA-PDDL (Kovacs 2012) and MA-HTN (Cardoso, Bordini et al. 2017). However, to utilize the extensive, well-documented HDDL domains and problems from IPC-HTN, HDDLGym is designed to work closely with the HDDL defined by Höller et al. (2020). We introduce a new protocol for extending HDDL domains and problems to support HDDLGym with multi-agent features. This includes making minor modifications to existing HDDL files from IPC-HTN.

Main contributions This paper makes the following three key contributions:

- We introduce HDDLGym, a novel framework that automatically bridges reinforcement learning and hierarchical planning by automatically generating Gym environments from HDDL domains and problems.
- We provide a protocol for modifying HDDL domains to support multi-agent configurations within HDDLGym, thereby extending hierarchical planning techniques to complex multi-agent environments.
- We detail HDDLGym's design and usage, demonstrating its effectiveness with examples from the Transport domain (in IPC-HTN) and the Overcooked environment (as Figure 1a and 1b, respectively).

The remainder of this paper is organized as follows: Section 2 provides background information on HDDL and OpenAI Gym, the two foundational frameworks on which our system is built. Section 3 discusses relevant prior works, thus highlights our contributions to the field. Section 4 then introduces the formal framework of HDDLGym, detailing how HDDL is modified to align with the agent-centric design of this tool. Section 5 covers the design and implementation details of HDDLGym. Following this, section 6 demonstrates the use of HDDLGym with examples from the Transport domain, representing domains from IPC-HTN, and Overcooked, representing customized environments. Section 7 discusses the key benefits and current limitations of the HDDLGym tool, along with future developments to address these limitations and expand its applications within artificial intelligence research. Finally, Section 8 concludes the paper.

2 Background

2.1 HDDL

HDDL (Höller et al. 2020) is an extension of PDDL (McDermott et al. 1998). Höller et al. (2020) define the domain and problem as follows.

Definition of Planning Domain: A planning domain D is a tuple (L, T_P, T_C, M) defined as follows.

- L is the underlying predicate logic.
- T_P and T_C are finite sets of primitive and compound tasks, respectively.
- M is a finite set of decomposition methods with compound tasks from T_C and task networks over the name $T_P \cup T_C$

Definition of Planning Problem: A planning problem $\mathcal{P}$ is a tuple (D, s_I, tn_I, g), where:

- $s_I \in S$ is the initial state, a ground conjunction of positive literals over the predicates assuming the closed world assumption.
- tn_I is the initial task network that may not necessarily be ground.
- g is the goal description, being a first-order formula over the predicates (not necessarily ground).

In other words, beyond the action definition in PDDL, which establishes the rules of interaction with the environment, HDDL introduces two additional operators: *task* and *method*. In HDDL, a *task* represents a high-level action, while a *method* is a strategy for accomplishing a task. Multiple methods can exist to perform a single task. Essentially, a method is a task network that decomposes a high-level task into a partially or totally ordered list of tasks and actions.

In HDDL, task is defined with its parameters, and method is defined with parameters, the associated task, preconditions, a list of subtasks with their ordering or a list of ordered-subtasks. Examples of task and method definitions from the original HDDL work (Höller et al. 2020) are:

```
(:task get-to :parameters (?l - location))
(:method m-drive-to-via
    :parameters (?li ?ld - location)
    :task (get-to ?ld)
    :precondition ()
    :subtasks (and
        (t1 (get-to ?li))
        (t2 (drive ?li ?ld)))
    :ordering (and
        (t1 < t2)))
```

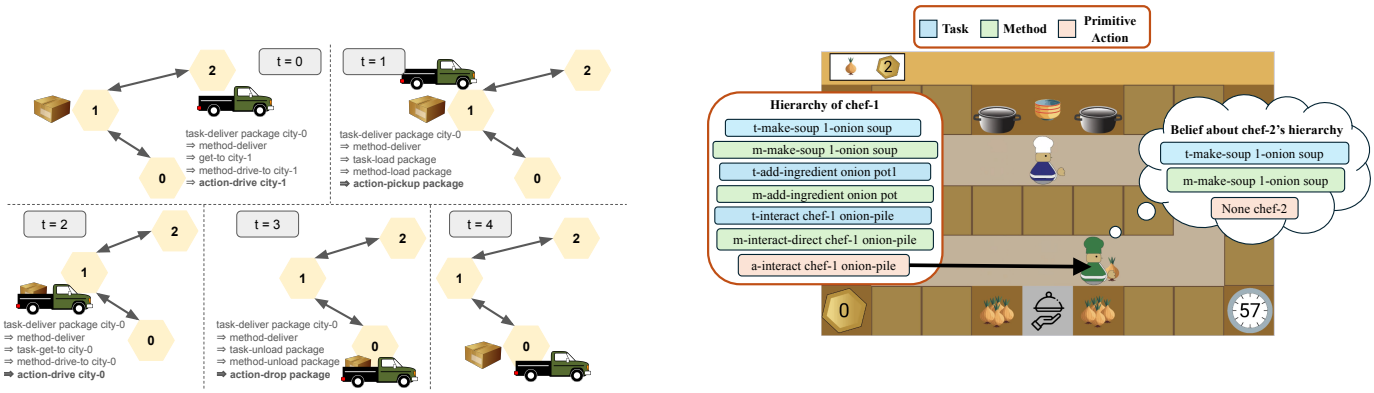
In HDDL, state-based goal definition is optional. Goals is instead defined as a list goal tasks in the HDDL problem file. An example of the goal in a transport problem is as follows.

```
(:htn
    :tasks (and
        (deliver package-0 city-loc-0)
        (deliver package-1 city-loc-2))
    :ordering ()
    :constraints ())
```

More details of HDDL domain and problem files can be found in Höller et al. (2020). In addition to the original format of HDDL, some modifications are required to make the HDDL domains and problems work smoothly with HDDLGym. Details of the modifications are discussed in Section 4.

2.2 OpenAI Gym

OpenAI Gym (Brockman et al. 2016) has become a widely adopted toolkit that offers a standardized interface for benchmarking and developing reinforcement learning (RL) algorithms. Its consistent application programming interface (API) includes key methods for environment initialization, resetting, and interaction, allowing researchers to focus on advancing RL algorithms without handling environment-specific implementation details. The toolkit includes a diverse set of environments, ranging from simple control tasks to complex simulations like Atari games, providing a common platform that enhances reproducibility and enables direct comparisons across different RL methodologies. Therefore, integrating OpenAI Gym with HDDL enables the development of a unified framework for designing and evaluating hierarchical RL approaches, combining the adaptive learning strengths of RL with the structured decision-making of hierarchical planning.



(a) Transport scenario

(b) Overcooked scenario

Figure 1: Environments used to demonstrate HDDLGym

3 Related Work

PDDLGym (Silver and Chitnis 2020) constructs Gym environments from PDDL domains and problems, serving as a valuable reference for our work. However, HDDL significantly differs from PDDL, particularly in managing hierarchical task networks or task and method operators. Additionally, PDDLGym operates under a single-action-per-step model, which suits many PDDL domains but lacks the complexity needed for advanced applications, such as multi-agent contexts. In contrast, our framework, HDDLGym, is designed to accommodate multi-agent environments, enabling the study of RL policies in more complex settings.

Similarly, PyRDDLGym (Taitler et al. 2022) integrates a planning domain language, Relational Dynamic Influence Diagram Language (RDDL) (Sanner et al. 2010), with Gym. RDDL is adept at modeling probabilistic domains with intricate relational structures; however, it does not inherently support multi-level actions. This limitation requires significant adjustments when defining hierarchical problems within PyRDDLGym. Users must creatively structure RDDL descriptions to represent sequences of actions, which can complicate the modeling of hierarchical tasks.

NovelGym is a versatile platform that supports hybrid planning and learning agents in open-world environments (Goel et al. 2024). It effectively combines hierarchical task decomposition with modular environmental interactions to facilitate agent adaptation in unstructured settings. Nevertheless, its hierarchical structure is relatively straightforward, primarily relying on primitive and parameterized actions defined in PDDL. Conversely, HDDLGym offers more advanced hierarchical capabilities through HDDL, granting users greater flexibility and complexity in specifying high-level strategies and problem-solving approaches.

In conclusion, while prior Gym-based frameworks like PDDLGym, PyRDDLGym, NovelGym, etc. provide valuable foundations for working with planning and learning agents, HDDLGym contributes to the field with its ability to manage complex multi-agent hierarchical environments.

4 Formal Framework

Due to various differences in original formalities and purposes between HDDL and Gym, some modifications in HDDL domain files are required to enable HDDLGym working smoothly. In this section, we introduce the agent-centric extension of HDDL, modified from the standard HDDL by Höller et al. (2020). The agent-centric extension only includes changes to the HDDL domain. The agent-centric planning domain is defined below:

Definition 1. *An agent-centric planning domain $\mathcal{D}$ is a tuple $\mathcal{D} = \langle t_a, L, T_P, T_C, M \rangle$, where:*

- *t_a is an agent type hierarchy in the domain.*
- *L is the underlying predicate logic.*
- *T_P is a finite set of primitive tasks, also known as actions. Actions can be further classified into agent actions and environment actions.*
- *T_C is a finite set of compound tasks.*
- *M is a finite set of decomposition methods with compound tasks from T_C and task networks over the name $T_P \cup T_C$.*

We next discuss the elements in Def. 1 that are different from the definition of planning domain in Sec. 2.

Agent type hierarchy t_a One major difference compared to the standard HDDL (Höller et al. 2020) is the addition of t_a. t_a is used to specify which types are classified as agent types within the domain. In an HDDL domain, this classification is done by defining the type "agent" within the `:types` block. For instance, in the Transport domain, the "vehicle" is designated as an agent type, as shown in the line 5 of the types block below. This approach allows the domain to clearly differentiate agent types from other entities, enabling more structured interactions within hierarchical planning tasks.

```
(:types
        location target locatable - object
        vehicle package - locatable
        capacity-number - object
        vehicle - agent)
```

Primitive Task Set T_P The primitive task set, T_P, encompasses all actions defined within the domain, classified as either agent actions or environment actions. Agent actions include one or more agents as parameters, while some actions—initially defined without agent parameters due to the nature of their predicates—must be modified to include agents if these actions are performed on behalf of agents. Additionally, in reinforcement learning, particularly in multi-agent settings, it is essential to ensure that the domain includes a *none* action for each agent, enabling an agent to choose no action for a given step. Therefore, the HDDL domain file should incorporate the following action block to support the *none* action functionality.

```
(:action none
    :parameters (?agent - agent)
    :precondition ()
    :effect ())
```

On the other hand, environment actions exclude agents from their parameters, making them non-agent actions. These actions execute automatically as soon as their preconditions are met immediately after all agents have completed their actions, enabling flexible environment dynamics.

Compound Task Set T_C The compound task set, T_C, includes all high-level tasks, aligning with the standard HDDL structure as described by Höller et al. (2020). However, in HDDLGym's implementation, additional task definition details are required. Specifically, to ensure task completion, HDDLGym checks the current world state against the defined task effects. Thus, task definitions must include explicit effects. In the following example from the Transport domain, the highlighted text indicates the additions made to the original HDDL task definition.

```
(:task get-to
    :parameters (?agent - agent ?dest - location)
    :effect (at ?agent ?destination))
```

The remaining components in the tuple, *L* and *M*, are consistent with the standard HDDL formulation as defined by Höller et al. (2020).

5 HDDLGym Framework

This section explains design and implementation of HDDLGym. It covers (1) details of HDDLEnv as a Gym environment, (2) the definition of Agent class, (3) observation and action space details, (4) RL policy, (5) planning for multi-agent scenarios, and (6) the overall high-level architecture of HDDLGym.

5.1 Gym and HDDLEnv

In the HDDLGym framework, we introduce HDDLEnv, a Python class that extends the Gym environment to support hierarchical planning with HDDL. Details of the HDDLEnv implementation are available in the `hddl_env.py` file.

Initialize and reset functions HDDLEnv is initialized using HDDL domain and problem files, together with an optional list of policies for all agents. During initialization, the HDDL files are converted into an environment dictionary, setting the initial state and goals. Agents are then initialized with their associated policies.

The reset function optionally accepts a new list of agents' policies and resets the environment to its initial state and goal tasks as specified in the HDDL problem file. It also reinitializes agents with their associated policies.

Step function The step function in HDDLEnv accepts an action dictionary from the agents and returns the new state, reward, 'done' flag (indicating win or loss), and debug information, similar to the format of OpenAI Gym's step function. After executing agents' actions, it also checks and applies any valid environment actions. Environment actions are actions that are not associated with any agent. This design enables the environment to change independently from agents' behaviors.

5.2 Agent

HDDLGym is designed as an agent-centric system. It inherently focuses on the interactions and actions of agents within the environment. Therefore, defining an Agent class, as in Definition 2 below, is critical in implementing HDDLEnv.

Definition 2. *An agent A is defined with a tuple $\langle N, P, B, H, U \rangle$ where:*

- *N is the agent's name,*
- *P is a policy,*
- *B B is the set of agents, representing the agent's belief about the configuration of other agents in the environment.*
- *H is a list of tasks, methods, and actions representing the action hierarchy of the agent,*
- *U is a function to update the agent's hierarchy based on the current state of the world.*

Initialize an agent All agents in the environment are initialized when an HDDLEnv instance is created or reset. Each agent is initialized with a name N and a policy P. The agent's name N is derived directly from the HDDL domain and problem files. The policy P refers to an RL strategy that the agent employs to support its hierarchical planning process. This initialization framework enables the agent to operate autonomously, with clearly defined parameters and behaviors, while accounting for both its own actions and the predictions of others within a multi-agent environment.

Update agent's hierarchy H An important method in the Agent class is the update hierarchy function U. This method checks whether any tasks or actions in the agent's hierarchy H have been completed by comparing their effects with the current state of the world. Once tasks or actions are completed, they are removed from both the agent's hierarchy H and the agent's belief about other agents' action hierarchies (B). U is called for each agent after the environment's step function is executed, ensuring that the agents are prepared for planning the next step.

5.3 Observation and Action Spaces

In general multi-agent problems, each agent can be assumed to have knowledge about the current state of the world, its own hierarchical actions, and other agents' previous actions. While different RL methods may have different designs regarding which information should be included in the models, in this work, we set the observation of each agent to include information of (1) current state of the world, (2) goal tasks, (3) the agent's action hierarchy, and (4) other agents' previous primitive actions.

The first element of the observation, the current state of the world, is represented by dynamic grounded predicates. Predicates are considered dynamic if they appear in at least one action, meaning they can be added to or removed from the current state of the world. For the other three elements, using grounded operators to capture agents' action hierarchies would result in an extremely large observation space. To address this, our approach represents the last three elements of the observation as a one-hot encoded vector based on all possible lifted operators and associated objects.

Unlike PDDL or non-hierarchical planning problems, HDDLGym outputs not only primitive actions but also action hierarchies that reflect the high-level strategies guiding these actions. Therefore, similar to the last three elements of the observation, an agent's action hierarchies are represented as a one-hot encoded vector encompassing all possible lifted operators and objects. The observation elements and action elements are also illustrated in the *Observation* box and *Operators* box of Figure 2, respectively. Further discussion on our choices for observation and action spaces can be found in Appendix A. Defining and implementing observation and action spaces are handled in the file `learning_methods.py`, making it easy for users to modify and experiment with different representation choices for observations and actions.

5.4 RL Policy

The RL policy plays a crucial role in the HDDLGym framework by supporting the search for an optimal hierarchical plan for each agent. The policy takes the observation as input, which includes information about dynamic grounded predicates, goal tasks, and previous action hierarchies. Its output is the probabilities of lifted operators and objects, which are then used to compute the probabilities of grounded operators. These probabilities guide the search for action hierarchies within the HDDLGym planner, as discussed in Sec. 5.5.

In this work, we implemented Proximal Policy Optimization (PPO) (Schulman et al. 2017) for discrete domains to effectively explore the application of RL in hierarchical planning problems. All components of the RL policy (including input-output spaces, training methods, and evaluation processes) are defined in the `learning_methods.py` file. By modifying this Python file, users can easily experiment with and implement alternative RL frameworks, supporting further research in multi-agent hierarchical planning.

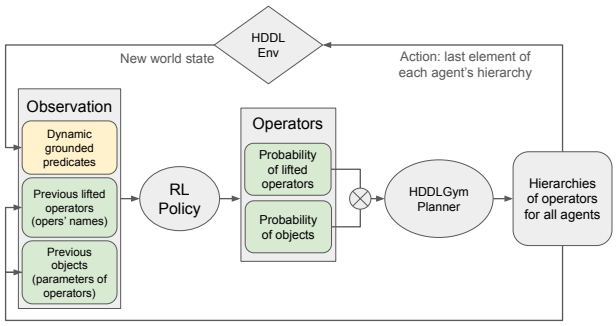


Figure 2: HDDLGym high-level architecture.

5.5 Planning for Multi-agent Scenarios

HDDLGym is designed to work in multi-agent settings; therefore, the planner also considers collaboration between agents. The HDDLGym planner is designed in a centralized format. In the decentralized version, the centralized planner is executed with the real agent and its belief about other agents.

In each step, the HDDLGym planner uses the outputs of the RL policy to find the action hierarchy for each agent. This work assumes that multiple agents in the environment have similar priorities when choosing actions. Therefore, the planner searches and updates each layer of the action hierarchies of all agents in the same iteration. Particularly, in each iteration, the planner generates a list of valid operators for each agent. Then, based on the probabilities given by the RL policy, the planner chooses valid joint operators and assigns the next layer of the agents' action hierarchies. Details of the HDDLGym Planner, including its algorithm, can be found in Appendix B.

5.6 HDDLGym Architecture

The high-level architecture of HDDLGym is demonstrated in Figure 2. As discussed in section 5.3, the input of the RL policy is a one-hot encoded vector that includes (1) dynamic grounded predicates, (2) lifted operators, and (3) objects representing the goal tasks and previous action hierarchies of all agents. The output of the RL policy is the probabilities of the lifted operators and objects. From this list of probabilities, we calculate the probabilities of the grounded operators by averaging the log-probabilities of the lifted operators and the objects involved in the grounded operators. The results are used to direct the HDDLGym planner's search for the best action hierarchy for each agent.

6 Applications

In this section, we discuss the two classes of domains that are supported in HDDLGym: the IPC-HTN and OpenAI Gym-based domains. We also use one representative example from each domain class; Transport from IPC-HTN and Overcooked from OpenAI Gym.[1] Additionally, an IPython

[1]More domains are included in the codebase of the system.

notebook tutorial is included in the codebase to walk users through key aspects of the HDDLGym tool, including: (1) generating and modifying HDDL domain and problem files, (2) running basic features, including random search and simple policy execution, (3) designing and implementing an RL policy, (4) training RL policies, and (5) deploying policies including visualization tools for result analysis.

6.1 IPC-HTN Domains

As previously discussed, Gym defines interactions between agents and the environment. Therefore, not all HDDL domains from IPC-HTN are directly compatible with HDDLGym. Since agent specification within a domain is necessary, this requirement may not be feasible or appropriate for every IPC-HTN domain (IPC 2023 HTN Tracks). HDDLGym is particularly well-suited to domains with agent-focused systems, such as Transport (where the vehicle serves as the agent), Rover (with the rover as the agent), and Satellite (with the satellite as the agent). To better illustrate the applications of these agent-centric environments, we provide a detailed explanation of how to modify HDDL domain files for the Transport environment in the following section.

Transport domain The goal of a Transport problem is to deliver one or more packages from their original locations to designated locations.

To run the Transport domain with HDDLGym, several modifications as described in Section 4 should be followed first. In the Transport domain, the agent is designated as 'vehicle'. All actions in the original Transport domain file originally contain vehicle in their parameters. Particularly, the block `:type` should include the line "`vehicle - agent`" to specify that the agent is a super type of the vehicle type. Then, add the none action for the agent as described in 4.2. Next is to add effects to all tasks. An example is the bold text in the following task definition.

```
(:task deliver
    :parameters (?p - package ?l - location)
    :effect (at ?p ?l))
```

At this point, the Transport domain and problem files are ready for use in HDDLGym to find a hierarchical plan. The resulting action hierarchy is illustrated in Figure 1a. In this scenario, the truck completes the "delivery package" goal task after four actions. At each step, the truck's action hierarchy begins with the goal task and concludes with a specific action. The hierarchy updates after each step, following a sequence of tasks in "method-deliver" to accomplish the "delivery package" goal.

To evaluate the capability of handling collaborative interactions in the Transport domain, we embed the collaborative task, method, and action to the Transport domain. Specifically, task `transfer`, method `m-deliver-collab`, and action `transfer-package` are added in the domain to enable the packages to be transferred from one vehicle to another when the vehicles are at adjacent locations. Details of these collaborative operators can be found in the codebase. Following this template, users can explore more interesting interactions and modify the Transport domain to study heterogeneous multi-agent problems.

Above is an example of how to modify an existing IPC-HTN domain to study with HDDLGym and explore more interesting features for multi-agent hierarchical planning. A similar process can be applied to other domains such as Rover, Satellite, and Barman-BDI, to plan with HDDLGym in single or multi-agent contexts. In our codebase, we include the modified HDDL domain files of Transport, Rover, Satellite, and Barman-BDI to run with HDDLGym. Other domains from IPC-HTN can also be modified as instructed to use with HDDLGym.

6.2 OpenAI Gym-based Domains

Writing HDDL domains and problems for an environment is not trivial, especially domains with complicated interaction rules. While there are many ways to do so, we would suggest starting with the goal task, then design methods to achieve the goal task, then come up with other intermediate tasks and methods for them, and gradually work to the primitive action. Here is an example of how HDDLgym is applied in support planning in the OpenAI Gym's Overcooked environment (Carroll et al. 2019).

Overcooked Overcooked (Carroll et al. 2019) is a popular Gym-based environment for studying reinforcement learning (RL), modeled after the cooperative and fast-paced mechanics of the original game. In Overcooked, players work together to complete cooking tasks under time constraints. In this scenario, two chefs must collaborate to prepare an onion soup. To do so, they need to place an onion in a pot, interact with the pot to start cooking, pour the cooked soup into a bowl, and deliver the bowl to the serving station (see Figure 1b).

In typical Overcooked scenarios, each agent can perform six primitive actions: moving in a 2D gridworld (up, down, left, right), interacting with objects, and doing nothing. Although the entire Overcooked scenario could be fully defined using HDDL, we found it more efficient to utilize HDDLGym for high-level planning and then apply A* (Duchoň et al. 2014) for motion planning to find the primitive actions as listed above. The core HTN for Overcooked domain is entailed in Figure 3. In the HDDL domain, we define the following tasks: make-soup, add-ingredient, cook, deliver, wait, and task-interact. Each of them has one or more methods to complete the tasks. Figure 3 only lists several key HTNs of the domain, though all HDDL domain and problem files of Overcooked environment can be found from the codebase. Additionally, Figure 1b demonstrates an example of a hierarchy of an agent and its belief about the other agent's hierarchy.

The following videos help visualize the result of combining HDDLgym for task planning and using A* for motion planning in various Overcooked layouts.

Bottleneck — https://tinyurl.com/hddlBottleNeckRoom
Coord. ring — https://tinyurl.com/hddlCoordinationRing
Left isle — https://tinyurl.com/hddlLeftIsle
Counter circuit — https://tinyurl.com/hddlCounterCircuit
Cramped room — https://tinyurl.com/hddlCrampedRoom

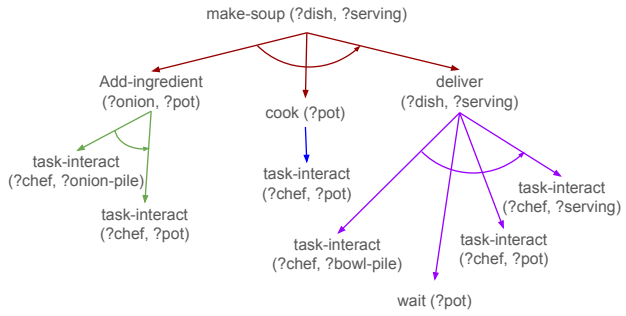


Figure 3: HTNs of the Overcooked scenario.

7 Discussion and Future Work

7.1 Discussion

In this work, we introduced HDDLGym, which has the capability to transform HDDL-defined hierarchical problems into Gym environments, enabling the application of reinforcement learning (RL) policies within hierarchical planning systems. By designing observation and action spaces that prioritize scalability, HDDLGym makes trade-offs that may slightly reduce RL model accuracy in exchange for the ability to tackle more complex hierarchical problems. This flexibility is particularly valuable when working with intricate task structures that require scalable solutions. Additionally, HDDLGym supports multi-agent environments, allowing for dynamic interactions between agents. This multi-agent functionality enriches the framework, facilitating the study of collaborative dynamic in hierarchical planning, thereby creating more engaging scenarios for RL research.

HDDLGym currently operates under certain limitations that we aim to address in future developments. First, it can only handle discrete state and action spaces, which restricts its application to scenarios that require continuous or hybrid spaces. Additionally, HDDLGym's multi-agent structure is symmetric, meaning all agents have equal roles and no agent has priority over another in task selection. This is a simplification that does not always align with real-world collaborative multi-agent systems, where some agents may have dominant roles or specific priorities. Furthermore, HDDLGym assumes a deterministic transition function, meaning that action effects are predictable and do not account for probabilistic outcomes. This limits its applicability to environments where uncertainty and stochastic outcomes are common. Lastly, similar to standard RL setups, the RL policy trained for HDDLGym problems is specific to a particular problem file within a domain and may not generalize effectively to other problem files. Changes such as a varying number of agents, different objects, or altered initial state conditions require retraining the policy, which hinders scalability and adaptability across diverse scenarios within the same domain. Addressing these limitations will broaden HDDLGym's usability in complex, real-world settings.

7.2 Future Work

While converting existing HDDL domains for use with HDDLGym is relatively straightforward, translating native Gym environments into HDDL domain and problem files is significantly more complex. Current efforts focus on converting more agent-centric environments, such as Overcooked, to the HDDL format to leverage HDDLGym's advantages. This ongoing work aims to expand the compatibility of agent-based Gym environments with HDDLGym, enabling more complex multi-agent hierarchical planning applications. In the future, HDDL domains can also be learned autonomously leveraging the recent advances in the field of learning HDDL domains from observations (Maxence Grand 2022).

As discussed, HDDLGym has limitations that could be addressed to better support complex multi-agent hierarchical problems. One improvement is enabling HDDLGym to handle multiple pairs of HDDL domain and problem files for different agents within a single Gym environment. Inspired by how multi-agent features are added to PDDL and HTNs through MA-PDDL (Kovacs 2012) and MA-HTN (Cardoso, Bordini et al. 2017), respectively, this approach would allow each heterogeneous agent to operate with its own unique pair of HDDL domain and problem files. This capability would enhance HDDLGym's ability to manage complex multi-agent dynamics beyond simple collaboration, supporting scenarios with competition, agent privacy, and distributed context information.

8 Conclusion

In this work, we introduced HDDLGym, which is a valuable tool for applying reinforcement learning to hierarchical planning by transforming HDDL-defined problems into Gym environments. We hope its flexible structure, enabling users to design their RL policy flexibly, can support addressing challenges in scalability and functionality when studying RL in complex, real-world, multi-agent scenarios.

Acknowledgments

We gratefully acknowledge the financial support of the Office of Naval Research under the ONR award grant #6000014476. Additionally, we extend our sincere gratitude to Pulkit Verma for his valuable insights and constructive feedback on the project.

References

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym.

Cardoso, R. C.; Bordini, R. H.; et al. 2017. A multi-agent extension of a hierarchical task network planning formalism. *Advances in Distributed Computing and Artificial Intelligence Journal*.

Carroll, M.; Shah, R.; Ho, M. K.; Griffiths, T.; Seshia, S.; Abbeel, P.; and Dragan, A. 2019. On the utility of learning about humans for human-ai coordination. *Advances in neural information processing systems*, 32.

Duchoň, F.; Babinec, A.; Kajan, M.; Beňo, P.; Florek, M.; Fico, T.; and Jurišica, L. 2014. Path planning with modified a star algorithm for a mobile robot. *Procedia engineering*, 96: 59–69.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1994. UMCP: A Sound and Complete Procedure for Hierarchical Task-network Planning. In *Aips*, volume 94, 249–254.

Goel, S.; Wei, Y.; Lymperopoulos, P.; Churá, K.; Scheutz, M.; and Sinapov, J. 2024. NovelGym: A Flexible Ecosystem for Hybrid Planning and Learning Agents Designed for Open Worlds. *arXiv preprint arXiv:2401.03546*.

Höller, D.; Behnke, G.; Bercher, P.; Biundo, S.; Fiorino, H.; Pellier, D.; and Alford, R. 2020. HDDL: An extension to PDDL for expressing hierarchical planning problems. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, 9883–9891.

IPC 2023 HTN Tracks. 2023. International Planning Competition 2023 HTN Tracks. Available at https://ipc2023-htn.github.io/.

Kovacs, D. L. 2012. A multi-agent extension of PDDL3.1. In *ICAPS 2012 Proceedings of the 3rd Workshop on the International Planning Competition*.

Liu, M.; Sivakumar, K.; Omidshafiei, S.; Amato, C.; and How, J. P. 2017. Learning for multi-robot cooperation in partially observable stochastic environments with macro-actions. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1853–1860. IEEE.

Maxence Grand, H. F., Damien Pellier. 2022. An Accurate HDDL Domain Learning Algorithm from Partial and Noisy Observations. In *ICAPS 2022 Workshop on Knowledge Engineering for Planning and Scheduling*.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D. S.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Comp. Vision and Control.

Sanner, S.; et al. 2010. Relational dynamic influence diagram language (rddl): Language description. *Unpublished ms. Australian National University*, 32: 27.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, T.; and Chitnis, R. 2020. PDDLGym: Gym Environments from PDDL Problems. arXiv:2002.06432 [cs].

Taitler, A.; Gimelfarb, M.; Jeong, J.; Gopalakrishnan, S.; Mladenov, M.; Liu, X.; and Sanner, S. 2022. pyrddlgym: From rddl to gym environments. *arXiv preprint arXiv:2211.05939*.

Wu, S. A.; Wang, R. E.; Evans, J. A.; Tenenbaum, J. B.; Parkes, D. C.; and Kleiman-Weiner, M. 2021. Too many cooks: Bayesian inference for coordinating multi-agent collaboration. *Topics in Cognitive Science*, 13(2): 414–432.

Xiao, Y.; Hoffman, J.; and Amato, C. 2020. Macro-action-based deep multi-agent reinforcement learning. In *Conference on Robot Learning*, 1146–1161. PMLR.

Appendices

A Observation and Action Spaces – Explanation and Discussion

A.1 Observation Space

Figure 2 demonstrates the roles of observation and action in the whole HDDLGym architecture. This section provides more details on our choice of observation and action spaces in designing the RL policy. The implementation for these spaces and RL policy is included in the file `learning_methods.py`. Therefore, this design can be flexibly changed to adapt to any other learning methods for solving hierarchical planning problems.

In our current design, the observations (inputs) provided to HDDLGym's RL policy include dynamic grounded predicates, goal tasks, and the current action hierarchies of all agents. Dynamic grounded predicates represent a subset of all possible grounded predicates within the environment. In HDDL, and PDDL more broadly, predicates can either be static or dynamic. Static predicates define unchanging world conditions (e.g., spatial relationships between locations), while dynamic predicates represent changing world conditions (e.g., agent positions). Dynamic predicates can be added or removed from the world state by actions.

Goal tasks are specified in the HDDL problem file under the `:htn` section. Each agent's action hierarchy is structured as a list, starting with a goal task and ending with a primitive action. Figure 1 illustrates examples of action hierarchies in the two environments, Transport and Overcooked, that are further discussed in Sec. 6.

Our approach focuses on using dynamic grounded predicates rather than all possible grounded predicates to minimize the observation space. However, this may restrict the generalization capability of the RL policy, as it is tailored to a specific HDDL problem file and may not generalize to other problems with different agents, objects, and static world conditions.

To represent goal tasks and action hierarchies, a practical method is to one-hot encode grounded operators using a comprehensive list of all possible grounded operators. However, this approach results in a large observation space due to additional operators for tasks and methods in hierarchical problems. Each grounded operator (whether a task, method, or action) can be decomposed into a lifted version paired with relevant objects. This allows agents' goal tasks and action hierarchies to be combined and represented as a one-hot encoded vector based on all possible lifted operators and associated objects.

A.2 Action space

In hierarchical planning problems, the action space should include information about high-level strategies in addition to primitive actions. However, as with the observation space, the list of all possible grounded operators can become exceptionally large in complex problems, posing a scalability challenge. To address this, an agent's action hierarchy is represented as a one-hot encoded vector that captures all possible lifted operators and associated objects.

This approach significantly reduces the size of both the observation and action spaces by omitting certain details, such as the order of action hierarchies and the association of objects with specific operators. Nevertheless, HDDLGym compensates with a hierarchy operator validation feature that aids in generating valid action hierarchies based on the policy's action output.

Another approach to reduce the size of the action space is explored in PDDLGym (Silver and Chitnis 2020), where they introduce a distinction between *free* and *non-free* parameters. Free parameters convey the essential information of an action, while non-free parameters are included due to their presence in precondition or effect expressions. Consequently, PDDLGym's action space consists of combinations of lifted operators with their free parameters. Although this approach works well in PDDLGym, it is challenging to implement within the HDDLGym framework, as identifying free parameters for tasks and methods is not trivial.

B HDDLGym Planner – Algorithm and Explanation

Algorithm 1 outlines the approach of the HDDLGym Planner, where agents determine their action hierarchies by iteratively updating through valid operator combinations. Particularly, HDDLGym Planner's inputs are a list $\mathcal{A}$ of all agents with uncompleted hierarchies, policy P, and deterministic flag d. The HDDLGym planner is a centralized planner. In case of decentralized planning, the list $\mathcal{A}$ includes a real agent and that agent's belief about other agents. The deterministic flag d determines whether the selection process should follow a deterministic or probabilistic approach when choosing operators to form agents' action hierarchies. The policy P is used to guide the search for a suitable hierarchy according to the flag d. In decentralized planning context, P is the policy of the real agent.

The planner begins by initializing an empty list, *Done*, to keep track of agents whose hierarchies end with an action (line 1). The while loop from lines 2 to 28 continues until all agents have completely updated their hierarchies. Within this loop, an empty list, $O_{\mathcal{A}}$, is initialized to store the valid operators of all agents (line 3). Next, the for-loop from lines 4 to 17 iterates to find all valid operators O_a for each agent a. To do this, the algorithm first checks if a is in *Done*, meaning its hierarchy is complete (line 5). If so, then O_a is set as a list containing the agent a's final action (line 6). Otherwise, the while loop from lines 8 to 14 runs until it finds a non-empty O_a. Initially, the list of valid operators for a is checked in line 9; if no valid operators are found (line 10), the last operator in a's hierarchy is removed, and the loop is rerun. However, if a's hierarchy is already empty, the `none` action is added to O_a (line 12).

Algorithm 1: HDDLGym Planner

Input: list of agents $\mathcal{A}$, deterministic flag *d*, policy *P*
Output: updated list of agents $\mathcal{A}$

Initialize an empty list *Done* to keep track of agents whose hierarchies reached action.
while not all agents in *Done* **do**
  Initialize an empty list $O_{\mathcal{A}}$ for valid operators of all agents
  for agent *a* in $\mathcal{A}$ **do**
    if *a* in *Done* **then**
      $O_a \leftarrow$ [action of agent *a*]
    else
      while O_a not empty **do**
        $O_a \leftarrow$ a list of valid operators for *a*
        if O_a is empty **then**
          Remove the last operator of agent *a* hierarchy from its hierarchy
          If no more operator from *a*'s hierarchy to remove, add `none` action to O_a
        end if
      end while
    end if
    Add O_a to $O_{\mathcal{A}}$
  end for
  C $\leftarrow$ Generate all possible combinations of joint operators from $O_{\mathcal{A}}$
  Remove any invalid combinations from *C*
  $P_O \leftarrow$ get probability of each combination in *C* with policy *P*
  if *d* is True **then**
    $c \leftarrow argmax_{c \in C} P_O$
  else
    $c \leftarrow$ Randomly choose a combination from C with weights be P_O
  end if
  Update hierarchies of all agent $\mathcal{A}$ with operators in c
  Check each agent's hierarchy and update *Done* if any hierarchy ends with action
end while
return $\mathcal{A}$ (updated)

The operator list O_a for each agent is then added to $O_{\mathcal{A}}$, the list of operators for all agents (line 16). This list, $O_{\mathcal{A}}$, is subsequently used to generate all combinations of joint operators, *C* (line 18). Line 19 details the pruning of invalid combinations in *C*. A combination is invalid if it violates either of two conditions: first, no agent should perform multiple different actions; and second, no action in the combination should have effects that conflict with the preconditions of other actions. After this pruning, *C* contains only valid operator combinations.

Lines 20 to 25 describe how the policy *P* is applied to select a combination *c* from the list of valid combinations, *C*. The probability list, P_O, corresponding to *C* is generated using policy *P*. Depending on the deterministic flag *d*, the chosen combination *c* is selected in either a deterministic manner (line 22) or probabilistically (line 24).

With the combination of operators determined, the next step is to use it to update each agent's hierarchy (line 26). The list *Done* is then updated if any agents have completed hierarchies (line 27). This process is repeated until all agents in $\mathcal{A}$ have completed their hierarchies. At this point, the HDDLGym planner returns the list of fully updated agents, as shown in line 29.