

# Iterative Environment Design for Deep Reinforcement Learning Based on Goal-Oriented Specification

Simon Schwan<sup>a</sup> and Sabine Glesner<sup>b</sup>

Software and Embedded Systems Engineering, Technische Universität Berlin, Straße des 17. Juni 135, Berlin, Germany  
{s.schwan, sabine.glesner}@tu-berlin.de

**Keywords:** Reinforcement Learning, Requirements Specification, Iterative Development, Goals, Environment Design.

**Abstract:** Deep reinforcement learning solves complex control problems but is often challenging to apply in practice for non-experts. Goal-oriented specification allows to define abstract goals in a tree and thereby, aims at lowering the entry barriers to RL. However, finding an effective specification and translating it to an RL environment is still difficult. We address this challenge with our idea of iterative environment design and automate the construction of environments from goal trees. We validate our method based on four established case studies and our results show that learning goals by iteratively refining specifications is feasible. In this way, we counteract the common trial-and-error practice in the development to accelerate the use of RL in real-world applications.

## 1 INTRODUCTION

Reinforcement Learning (RL) has emerged as a promising solution for complex control problems such as collision avoidance (Everett et al., 2021) in autonomous driving. RL agents learn through interactions with their environment, being rewarded for desirable behavior. The initial step in the development of RL solutions involves defining an environment in form of a Markov decision process (MDP). Despite the potential of RL, the definition of the environment requires significant experience and expertise and often involves trial-and-error. This results in very high entry barriers for developers that are experts in their application domain, but not RL. Thereby, these barriers limit the application of RL in practice drastically.


Goal-oriented specification (Schwan et al., 2023) enables to specify goals in a tree structure, which allows to abstract from technical details of RL. However, at this point, goal-oriented specification is missing definitions of how to automatically construct environments from goal tree specifications. These definitions are needed to make the approach applicable by enabling the training of RL agents from goal trees. Moreover, it is challenging to develop effective specifications on the first try because of many interdependent design choices. These choices come not only


from defining the environment, but also from training agents on the environment.

With this work, we close the gap and address the challenge of constructing environments from goal trees to enable iterative goal-oriented design. Our key idea is to design RL environments in iterations of three phases: specifying goals, training agents, evaluating the results for future improvements. To achieve this, our two main contributions are as follows:

1. We introduce a method for automatically constructing environments from goal trees. Thereby, we enable the training of RL agents from these specifications.
2. We instantiate our method with definitions for a specific set of goal tree components. By carefully choosing these definitions, we ensure that goal tree refinements lead to an increase of the rewarding feedback to the agent and create the opportunity for iterative improvements.

Together, manually refining specifications and automatically constructing environments from them, enables domain experts to train agents while focusing on the goals rather than the technical details. At the same time, we reduce time-consuming tasks of constructing the environment manually in each iteration. This makes iterative environment design from goal-oriented specifications practical, and we evaluate our method through four case studies from the Farama Gymnasium (Farama Foundation, 2024). First, we

<sup>a</sup>  <https://orcid.org/0009-0002-4085-1777>

<sup>b</sup>  <https://orcid.org/0009-0003-6946-3257>

infer goals from the original environments to enable goal-oriented specification. Second, we define multiple goal trees for each case study. Third, we train agents for each automatically constructed environment and analyze the results.<sup>1</sup>

The paper is structured as follows. We relate our work to existing research (Section 2). Then, we describe preliminaries (3) such as MDPs and goal-oriented specification. Subsequently, we introduce our running example (4) and define our automated construction of environments from goal trees (5). We evaluate our method (6) and conclude (7).

## 2 RELATED WORK

With the increasing use of artificial intelligence (AI) systems, Requirements Engineering for AI (RE4AI) (Ahmad et al., 2023) becomes relevant. The survey identifies the Unified Modeling Language (UML) and goal-oriented requirements engineering (GORE) to be the prevalent modeling languages in RE4AI. Reinforcement learning is a methodology that addresses problems within the context of a specific theoretical framework: the Markov decision process. Our specification method enables requirements engineering that is inspired by GORE (Van Lamsweerde, 2001) but tailored to the specific framework of RL. A survey of Human-in-the-loop (HITL) RL (Retzlaff et al., 2024) examines existing research. It identifies the HITL paradigm to be of utmost importance and proposes that humans, i.e. developers, domain experts and users, interact with the RL system in four sequential phases: agent development, agent learning, agent evaluation, agent deployment. In alignment with the theoretical findings of the survey, we introduce our iterative design approach according to the initial three phases. However, we do not consider the agent deployment.

We base our specification language on goals, which are used in several other RL methods (Schaul et al., 2015; Andrychowicz et al., 2017; Florensa et al., 2018; Jurgenson et al., 2020; Chane-Sane et al., 2021; Okudo and Yamada, 2021; Ding et al., 2023; Okudo and Yamada, 2023). These methods focus on improving the training efficiency and we identify three major directions: (1) using goals to shape and make the reward dense (Okudo and Yamada, 2021, 2023; Ding et al., 2023); (2) the division of a major goal into subgoals (Jurgenson et al., 2020; Chane-Sane et al., 2021) such as following intermediate waypoints on a trajectory; (3) learning goals simultane-

ously (Schaul et al., 2015; Andrychowicz et al., 2017; Florensa et al., 2018) to improve generalizability. In contrast, our approach uses goals to specify requirements and enable iterative environment design instead of focusing on training efficiency.

Furthermore, the idea of goal-oriented specification is to integrate existing RL techniques into the specification and training procedure. In this context, we review existing goal-based methods. Often, goal-based methods can be automatically applied (Andrychowicz et al., 2017; Florensa et al., 2018; Chane-Sane et al., 2021; Jurgenson et al., 2020) to train RL agents. Hindsight experience replay (HER) (Andrychowicz et al., 2017) enables to learn many goals from the same episode by relabeling the target goals of the terminal state. Thus, HER improves generalizability by learning from unsuccessful episodes. While we find HER promising to be integrated into our training, other approaches (Florensa et al., 2018; Jurgenson et al., 2020; Chane-Sane et al., 2021) need to have specifically tailored RL algorithms. These methods stand in contrast to our approach that enables learning from standard, model-free RL algorithms. Subgoal-based reward shaping (Okudo and Yamada, 2021, 2023) relies on the manual specification of ordered subgoals to guide the agent. We may be able to integrate it into goal-oriented specification by developing a corresponding goal-tree operator.

Finally, there are other specification languages for RL that are related to our work. While (Hahn et al., 2019) specifies RL objectives in an  $\omega$ -regular language, most languages (Li et al., 2017; Jothimurugan et al., 2019, 2021; Cai et al., 2021; Hammond et al., 2021) are based on linear temporal logic, which allows them to specify temporal properties. These languages enable the design of reward with theoretical guarantees such as providing policy invariance. However, these theoretical considerations do not guarantee that deep RL algorithms converge because of statistical optimization. In contrast, our method integrates training into the iterative environment design to counteract unpredictable side effects.

## 3 PRELIMINARIES

This section first provides preliminaries for reinforcement learning, followed by goal-oriented specification.

**Reinforcement Learning.** A reinforcement learning problem is formally modeled as a *Markov decision process (MDP)* (Sutton and Barto, 2018) by a tuple  $(S, A, P, R)$  with  $S$  being the space of all states

<sup>1</sup>Code and results are available at <https://doi.org/10.6084/m9.figshare.26408821.v1>

satisfying the Markov property,  $A$  being the space of all actions,  $P(s, a, s') = Pr[s_{t+1} = s' | s_t = s, a_t = a]$  being the transition probability and  $R : S \times A \times S \rightarrow \mathbb{R}$  being the immediate reward. The RL agent interacts with the MDP and collects samples in form of *episodes*  $\tau = (s_1, a_1, r_1, s_{t+1}, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T)$  with states  $s_i \in S$ , actions  $a_i \in A$ , rewards  $r_i = R(s_i, a_i, s_{i+1})$  and the terminal state  $s_T$  at time  $T$ . The objective of RL algorithms is to find a *policy*  $\pi : S \rightarrow A$ , also named *agent*, that maximizes the expected cumulative and discounted reward, i.e. *return*  $R(\tau) = \sum_{i=0}^T \gamma^i r_i$ , with *discount factor*  $\gamma \in [0, 1]$ . Depending on the parameters,  $R(s_t, a_t, s_{t+1})$  denotes the reward and  $R(\tau)$  the return. A popular RL algorithm is *Proximal Policy Optimization (PPO)* (Schulman et al., 2017), a policy gradient method that updates in small steps by a clipped surrogate objective. PPO can optimize for discrete and continuous actions.

There are several methods to specify reward. *Sparse reward*, i.e. rewarding only on success, has the advantage of defining a single and clear objective. However, the agent may not be able to experience this sparse reward because it does not reach the associated success states. In this case, it is possible to shape the reward to a *dense* function or to sparsely reward in intermediate states. If the reward  $R$  consists of multiple components  $R_i$  as in multi-objective RL, a common choice is to scalarize using the weighted linear sum of the components. This enables the use of single-objective RL algorithms (Roijers et al., 2013).

Throughout the paper, we assume state spaces  $S$  to be *feature spaces*  $S : S_A \times S_B \times S_C \times \dots$  consisting of a set of features  $F = \{A, B, C, \dots\}$  where a state  $s \in S$  is a tuple  $s = (s_A, s_B, s_C, \dots)$ .

**Goal-Oriented Specification.** Goal-oriented specification (Schwan et al., 2023) introduces the specification of goals for RL agents in a hierarchical tree structure. The approach formalizes the separation of Markov decision processes into immutable aspects, i.e. the *initial environment*, and the engineered aspects, i.e. *requirements* as depicted in Figure 1. The *initial environment* is a three tuple  $(S^*, A^*, P^*)$  with  $S^*$  being the *initial state space* (e.g. available sensors),  $A^*$  being the *initial action space* (e.g. actuators) and  $P^*$  being the *initial transition probabilities* (e.g. physics of the world or a simulation). These aspects are immutable and cannot be modified during specification. *Requirements* are the counterpart to the initial environment. They include aspects of MDPs that can be designed or manipulated by the engineer to solve a problem with RL. Formally, requirements are a tuple  $(S_G, A_G, T_G, R_G)$  that belong to a goal space  $G \subseteq S_G$ . The state space  $S_G$  contains

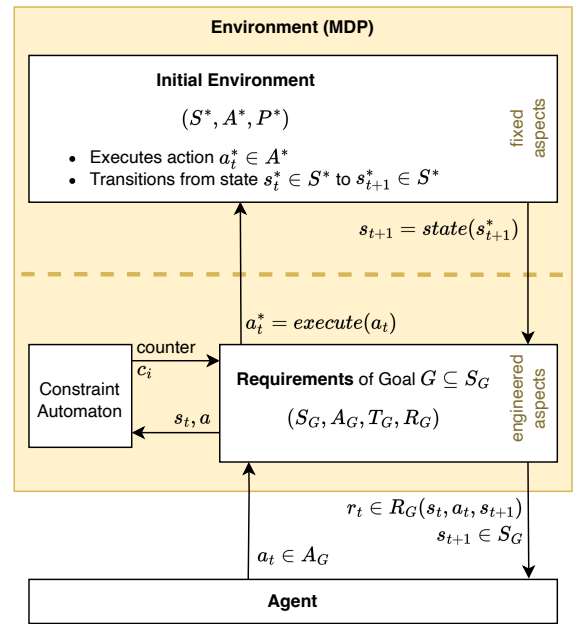


Figure 1: Composed environment (Markov decision process) from the initial environment and requirements.

feature-engineered states. The action space is defined by  $A_G$ , which contains possibly abstracted actions that may differ from the initial environment. The termination space  $T_G : S_G \times A_G$  contains state-action pairs for which episodes in the MDP terminate. This allows to constrain undesired transitions in  $P^*$  for state-actions  $(s, a) \in T_G$ . The reward  $R_G : S \times A \times S$  is a single scalar reward function that implicitly inherits the objective associated with the goal  $G$ .

Figure 1 shows how the initial environment and requirements are combined to construct an environment in form of an MDP. To do so, it is necessary to specify a mapping between the requirements and the initial environment based on two functions. The first function  $state : S^* \rightarrow S_G$  enables the conversion of the initial state space  $S^*$  to the feature-engineered state space of the requirements  $S_G$ . The agent chooses its next action  $a_t \in A_G$  based on the converted states. It is necessary to execute this action in the initial environment, which the definition of  $execute : A_G \rightarrow A^*$  enables. Then, the environment proceeds to its next state.

Furthermore, goal-oriented specification (Schwan et al., 2023) introduces the ability to specify goals in a tree. The idea is to construct a requirements tuple from a goal-tree specification, but exact definitions are not presented in the original work. Each node in the tree contains its own goal space and requirements. The construction of these requirements is defined by the following tree components: leaf nodes, operators, annotations. Leaf nodes are the atomic units for goals

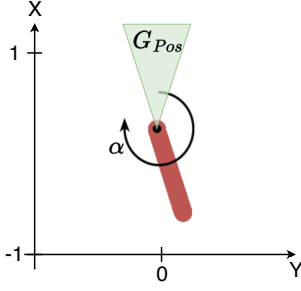


Figure 2: Pendulum environment.

in the tree and include an associated goal space  $G$ . Goals can be hierarchically structured by a generic operator  $(S_G, A_G, T_G, R_G) = \odot_i^N (S_i, A_i, T_i, R_i)$  that allows to refine the parent goal  $G$  into  $N$  subgoal nodes. In addition, nodes can be annotated with distance annotations that guide the agent to the goal space as well as safety constraints. Safety constraints are specified by their constraint state-action space  $C : S \times A$  and the number of allowed constraint violations  $\delta_C \in \mathbb{N}$  where  $cnt_C : S \times A \rightarrow \{0, 1\}$  counts a constraint violation  $(s, a) \in C$  with 1.

## 4 RUNNING EXAMPLE

We illustrate our work using the Pendulum case study from Farama Gymnasium (Farama Foundation, 2024) that is shown in Figure 2. The objective of the case study is learning to stabilize a circular pendulum in an upright position. The state space  $S^* \subseteq \mathbb{R}^3$  contains three features, i.e. the  $X$  and  $Y$  position and the angular velocity  $\alpha_V$ . Actions  $a \in \mathbb{R}$  define the torque that is applied to the pendulum.

The objective of the pendulum is implicitly defined by the given reward function. However, to use goal-oriented specification, we need explicit goals as sections of the state space. Through empirical measurements on trained agents based on the original reward, we examine the terminal states of the episodes. Based on the results, we define the goal space such that a successfully trained agent consistently reaches the goal at the end of an episode. For the Pendulum case study, we define the goal  $G_{Pend}$  with constants  $c_i$  as standing upright at  $(s_X, s_Y) = (1, 0)$  with a tolerance of 15 degrees ( $c_X = \sin(15) \approx 0.966$ ,  $c_Y = \sin(15) \approx 0.259$ ) with low angular velocity below the threshold  $c_{\alpha_V} = 0.25$ :

$$G_{Pend} = \{(s_X, s_Y, s_{\alpha_V}) \mid s_X \geq c_X, |s_Y| \leq c_Y, |s_{\alpha_V}| \leq c_{\alpha_V}\} \quad (1)$$

In the following, we simplify the notation of goal spaces by using  $\{s_X \geq c_X, |s_Y| \leq c_Y, |s_{\alpha_V}| \leq c_{\alpha_V}\}$  analogously.

## 5 AUTOMATED CONSTRUCTION OF ENVIRONMENTS FROM GOAL TREE SPECIFICATIONS

In this section, we present our two main contributions to evolve goal-oriented specification (Schwan et al., 2023) and make iterative design of RL agents from goal trees practical. First, we introduce our goal tree processing algorithm that automatically constructs a single, composite requirements tuple at the root. Our algorithm is generic with respect to leaf nodes, operators and annotations from the goal tree. Thereby, it allows the development of future components integrating further RL methods. Second, we introduce a specific set of definitions for leaf nodes, operators and annotations that instantiate the generic parts of our algorithm. These definitions allow the construction of environments from which RL agents can be directly trained and evaluated. We leverage the fact that goal-oriented specification allows specifying the same goal in a variety of goal trees. According to our definitions, node refinements increase the rewarding feedback to the agent. Therefore, each refined goal tree specification leads to the construction of a unique environment variant. Together, our algorithm and definitions allow us to automatically construct unique environments that can be used to train RL agents and analyze their behavior. Finally, this enables iterative environment design from goal-oriented specification.

Next, we introduce our algorithm followed by the definitions that instantiate the generic construction. For clarity, we use the terminology of *specifying* for manually engineered aspects of the goal tree specification and *constructing* for our automated construction of requirements.

We implement a depth-first traversal to recursively construct the composite requirements as shown in Algorithm 1. The algorithm  $process\_node(node, S, A)$  receives a *node* as input for which we construct the output requirements  $(S_G, A_G, T_G, R_G)$  and the goal space  $G$ . Additionally, it receives a state space  $S$  and an action space  $A$  as input. We create a single requirements tuple for a specification by starting the process at the root node of the tree  $process\_node(root, S_{root}, A_{root})$ . Here,  $S_{root}$  and  $A_{root}$  are the direct result from the specified  $state(\dots)$  and  $execute(\dots)$  functions as presented in Section 3. Our algorithm processes a node in three sequential steps as follows.

First, we construct a requirements tuple for the node under construction. Nodes may be either a leaf or an operator node. Leaf nodes have a specified goal space  $G = goal(node)$ , from which we construct the requirements tuple according to our definitions below.

```

Data:  $node, S, A$ 
Result:  $G, (S_G, A_G, T_G, R_G)$ 
/* Step 1: Process leaf or operator */
if  $node$  is leaf then
   $G = goal(node)$ 
   $(S_G, A_G, T_G, R_G) = leaf(S, A, G)$ 
else
   $r = \emptyset$ 
  for  $c \in children(node)$  do
     $(S_i, A_i, T_i, R_i), G_i = process\_node(c, S, A)$ 
     $r \leftarrow r \cup \{(S_i, A_i, T_i, R_i)\}$ 
  end
   $G, (S_G, A_G, T_G, R_G) = \odot_i^N(S_i, A_i, T_i, R_i), G_i$ 
end
/* Step 2: Process node annotations */
for  $a \in annotations(node)$  do
   $(S_G, A_G, T_G, R_G) \leftarrow build(a, (S_G, A_G, T_G, R_G))$ 
end
/* Step 3: Process root specifics */
if  $node$  is root then
  insert  $G$  into  $T_G$  for all actions
end
return  $G, (S_G, A_G, T_G, R_G)$ 

```

Algorithm 1: Our algorithm  $process\_node(node, S, A)$  implements a recursive depth-first traversal of a goal tree specification to construct composite requirements.

Operator nodes have children, and we recursively construct their requirements  $(S_i, A_i, T_i, R_i)$  depth-first by calling  $process\_node(c, S, A)$ . Subsequently, we combine these requirements using the generic operator  $\odot$ . This generic approach enables us to extend our specification language in the future. However, we introduce the specific definitions of our  $\wedge$ -operator below. Second, we adapt the requirements according to the annotations of the node. We sequentially process these annotations by updating the requirements ( $\leftarrow$ ) according to our definitions as introduced below. Third, we end the training of episodes if the agent enters the root goal space. We do so by inserting the goal space  $G$  into the termination space  $T_G$  at the root.

Following, we introduce definitions for each goal tree component: leaf nodes,  $\wedge$ -operator, distance and constraint annotations. We use these definitions to automatically construct the requirements tuple at the root according to Algorithm 1. The result is a unique environment for each goal tree specifying identical goals. We illustrate all definitions by applying them to our running example from Section 4 using the three goal tree specifications as shown in Figure 3.

**Leaf Nodes.** Leaf nodes are the atomic units of goals in goal-oriented specification. Each leaf node contains a specified goal space  $G \subseteq S_G$ , which is the section of the state space that the agent aims to reach. We construct the requirements  $(S_G, A_G, T_G, R_G)$  for a leaf by calling  $leaf(S, A, G)$  as shown in Algorithm 1.

The observation and action space need to conform with the other components of the tree. For leaf nodes, we externally define these by the structure of the tree:

$$S_G = S \wedge A_G = A$$

$S$  and  $A$  are given as input into the  $leaf$  function.

The reward of a leaf node needs to give feedback to the agent when it reaches the goal space. At the same time, it is possible to specify goal trees that compose multiple leaf goals. We define the reward  $R_G$  as follows:

$$R_G(s_t, a_t, s_{t+1}) = \begin{cases} 1 & s_t \notin G, s_{t+1} \in G \\ -1 & s_t \in G, s_{t+1} \notin G \\ 0 & \text{else} \end{cases}$$

The positive reward for entering the goal space  $S_{t+1} \in G$  may be sufficient if the leaf is the only goal in the tree. However, goal-oriented specification enables the composition of leaf nodes through operators, and it may be possible that an agent again exits the goal space. For this reason, we neutralize the positive reward with a negative reward of the same magnitude to prevent the recurrent collection of positive rewards. The termination space  $T_G$  defines state-action pairs at which episodes are terminated. For the same reason of composing leaf nodes, we do not terminate episodes when reaching the goal of a leaf node. Instead, we initialize the termination space of a leaf node as the empty space:

$$T_G = \emptyset$$

Still, we terminate episodes when the agent enters the composite goal at the root as shown in step three of Algorithm 1.

*The simplest goal tree for our running example is to specify a leaf node with the goal space  $G_{Pend}$  from Eq. 1 at the root (Figure 3.a). This specification results in a reward function with a sparse positive reward when the agent successfully reaches the goal and episodes terminate. Nevertheless, this specification may prove challenging during training. Depending on the size and the dynamics of the environment as well as the exploration strategy, the agent may not be able to reach goal states and experience reward.*

**$\wedge$ -operator Node.** The  $\wedge$ -operator enables the specification of simultaneous subgoals. For this reason, our definitions follow the semantics of intersecting subgoal spaces  $G = \bigcap_i^N G_i$ . We instantiate the generic operator  $\odot$  from Algorithm 1 by defining  $(S_G, A_G, T_G, R_G) = \wedge_i^N(S_i, A_i, T_i, R_i)$ . Our definitions

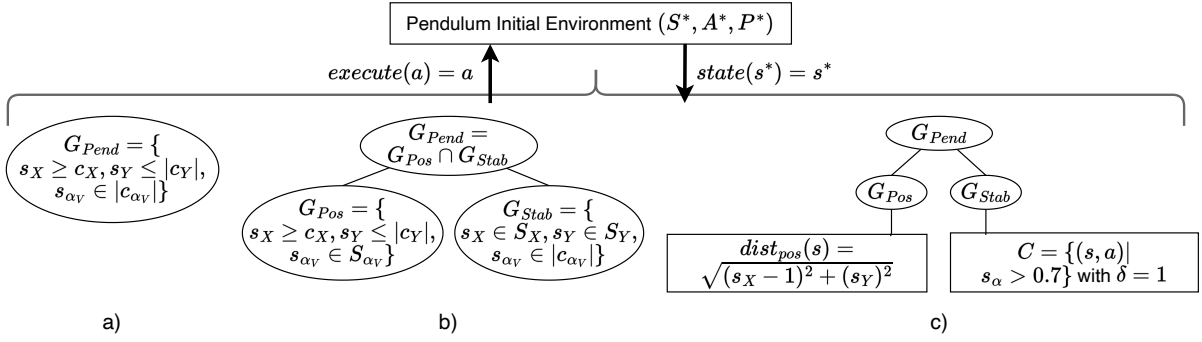


Figure 3: Three goal tree specifications (a-c) for the same goal  $G_{Pend}$  with increasing specification details (left to right) combined with the Pendulum initial environment.

enable the construction of the composite requirements tuple of the parent goal as follows.

To enable the composition of child nodes, we define state and action spaces of the parent and its children to be identical:

$$\forall i. S_G = S_i \wedge A_G = A_i$$

Moreover, this definition allows the intersection of the goal spaces to construct the parent goal space according to our semantics. The reward of our  $\wedge$ -operator node should compose reaching the goals of its children. Our recursive algorithm constructs the requirements tuple of the child nodes first. These requirements include reward components  $R_i$  for each child node. We use these components and define the parent reward  $R_G$  to be the cumulated weighted sum:

$$R_G(s_t, a_t, s_{t+1}) = \sum_i^N \omega_i R_i(s_t, a_t, s_{t+1})$$

By default, we weight the reward components equally with  $\omega_i = 1/N$ . However, this definition introduces the challenge of weighting, which is a non-trivial and often time-consuming manual task in reward design. Nevertheless, it defines a reward shape that increases the feedback to the agent by rewarding the success of reaching intermediate subgoals. For this reason, our  $\wedge$ -operator allows for refinement of a goal and enables the construction of unique requirements while preserving the goal space at the root. We define the termination space of the parent to terminate episodes whenever a child indicates termination:

$$T_G = \bigcup_i^N T_{G_i}$$

For example, the  $\wedge$ -operator enables us to refine the root goal  $G_{Pend}$  of Figure 3.a into two child goals for reaching the position  $G_{Pos} = \{s_X \geq c_X, |s_Y| \leq c_Y, s_{\alpha_V} \in \alpha_V\}$  and stabilizing the pendulum  $G_{Stab} = \{s_X \in X, s_Y \in Y, |s_{\alpha_V}| \leq c_{\alpha_V}\}$

as shown in Figure 3.b. While the goal space at the root remains the same  $G_{Pend} = G_{Pos} \cap G_{Stab}$ , the use of the  $\wedge$ -operator leads to a weighted sparse reward shape with  $R_{Pend} = \omega_{Pend,0} R_{Pos} + \omega_{Pend,1} R_{Stab}$ . We omit the reward parameters  $R(s_t, a_t, s_{t+1})$  for illustration. In contrast to the constructed requirements from Figure 3.a, not only the overall goal of the root is rewarded but also intermediate steps when reaching a child goal space. Thus, the  $\wedge$ -operator increases the feedback to the agent, and we can construct a unique environment variant from the goal tree.

**Annotations.** Tree nodes can be annotated in goal-oriented specification. Thereby it is possible to constrain undesirable behavior or increase feedback to the agent by guiding it towards the desired goal. Each annotation modifies the requirements tuple of its node, which we denote by the left arrow ( $\leftarrow$ ) in step 2 of Algorithm 1. For each additional reward component, we specify a weight for balancing.

In goal-oriented specification, safety constraints are specified by their constraint action-state space  $C : S \times A$ , along with the number of permitted violations  $\delta \in \mathbb{N}$ . However, differing state transitions for identical states violate the Markov property. This may be the case for terminating episodes or penalizing the agent when the violation boundary  $\delta$  is reached. To preserve the Markov property for constraint violations, we make the violation counter transparent to the agent. To achieve this, we construct an extended state space  $S_G \leftarrow S_G \times \mathbb{N}$  by adding a violation counter  $s_C$  with:

$$s_C = \delta - \sum_t^T cnt_C(s_t, a_t)$$

The construction entails updating the state and termination spaces for other tree components with the extended state space to comply with our definitions. For instance, the state spaces are identical for parent and

child nodes according to definitions of the  $\wedge$ -operator. For this reason, we need to update the state and termination spaces of adjacent tree components when adding the constraint violation counter. Furthermore, we integrate the additional state feature into a *constraint automaton* as shown in Figure 1 that counts constraint violations. If an episode contains  $\lambda$  constraint violations, we end it. We do so by modifying the termination space to include states with a violation counter of  $s_C = 0$ :

$$T_G \leftarrow T_G \cup \{(s, a) | s_C = 0\}$$

Finally, we add a penalty to the original reward as follows:

$$R_{Pen}(s_t, a_t, s_{t+1}) = \begin{cases} -1 & (s_t, a_t) \in C \\ 0 & \text{else} \end{cases}$$

Distance annotations of goal-oriented specification enable to guide the agent towards the goal. The allow a dense reward shaping by specifying a Euclidean distance function  $dist : S_G \rightarrow \mathbb{R}$  with

$$dist(s) = \sqrt{(s_x - g_x)^2 + (s_y - g_y)^2}$$

for a goal state  $g \in G$ . From this, we construct a potential-based reward shape (Ng et al., 1999):

$$R_{Dist}(s_t, a_t, s_{t+1}) = dist(s_t) - dist(s_{t+1})$$

Finally, we add the dense reward component  $R_{Dist}$  to the existing reward of the requirements.

Figure 3.c illustrates a third tree specification in which the refined subgoals  $G_{Pos}$  and  $G_{Stab}$  are annotated. The sparse reward of  $G_{Pos}$  is shaped by a dense reward constructed from the specified distance  $dist_{Pos}(s) = \sqrt{(s_x - 1)^2 + (s_y - 0)^2}$  to the top center position. To restrict exploration of the state space with high angular velocities of  $s_\alpha \geq 0.7$ , we annotate the stabilization node by specifying the safety constraint  $C$  with  $\delta = 1$ . Note: we introduce the constraint  $C$  for illustration purpose only and we do not use it in our experiments.

From the annotated specification in Figure 3.c, we construct a structured reward function at our root requirements with weighted components as follows:

$$R_{Pend} = \underbrace{\omega_{Pend,0}(\omega_{Pos,0}R_{Pos} + \omega_{Pos,1}R_{Dist})}_{\text{position goal}} + \underbrace{\omega_{Pend,1}(\omega_{Stab,0}R_\alpha + \omega_{Stab,1}R_{Pen})}_{\text{stabilization goal}}$$

Each reward component strictly belongs to one of the nodes with goal spaces  $G_{Pos}$  and  $G_{Stab}$ . The weights  $\omega_{Pend,i}$  allow to balance between the position

and stabilization goals whereas the weights  $\omega_{Pos,i}$  and  $\omega_{Stab,i}$  balance the proportions of the inner reward. Again, we construct requirements for a goal tree specification with the same goal space at the root. However, our definitions enable us to construct a unique and trainable environment variant.

At this point, we have evolved goal-oriented specification by enabling the automated construction of environments from goal trees. Furthermore, we construct unique requirements that increase the feedback to the agent for each goal tree refinement. In the following section, we use our automated construction to train RL agents on a series of specifications in four case studies and evaluate the results.

## 6 EXPERIMENTS & DISCUSSION

With experiments on four existing case studies from Farama Gymnasium (Farama Foundation, 2024), we examine two key questions for iterative environment design: (1) Can we specify goal trees from which agents are trained to achieve the specified goals? (2) How can the refinement of tree specifications be used in the iterative design of RL environments? To answer these questions, we first present our experimental design and setup in Section 6.1 and discuss the results subsequently in Section 6.2.

### 6.1 Experiment Design and Setup

We evaluate our specification language on four case studies from Farama Gymnasium (Farama Foundation, 2024), encompassing control problems with discrete and continuous action spaces: Acrobot, Pendulum, MountainCarContinuous, LunarLander. Originally, each case study represents a trainable RL environment, which we call baseline. Each baseline includes a reward that implicitly defines the objective. For each case study, we follow our key idea of designing RL environments in iterations of specifying goals, training agents and evaluating the results.

Initially, we focus on training the baseline and identify goals in the state space that are necessary to use goal-oriented specification. For this purpose, we use Proximal Policy Optimization (PPO) (Schulman et al., 2017) because of its versatility in handling both discrete and continuous action spaces along with the minimal tuning effort required. We manually tune hyperparameters for PPO to ensure that the agents can solve the tasks. For fairness, we use these baseline-tuned parameters for all experiments of the case study and reduce bias from hyperparameter tuning. Finally,

Table 1: Complete list of experiment configurations for our case studies.

	Acrobot-v1	Pendulum-v1	MountainCar Continuous-v0	LunarLander-v2
State Space	$S_{Cos,\theta_1} \times S_{Sin,\theta_1} \times S_{Cos,\theta_2} \times S_{Sin,\theta_2} \times S_{\alpha_{Vel},\theta_1} \times S_{\alpha_{Vel},\theta_2} \times S_{Height}^b$	$S_X \times S_Y \times S_\alpha$	$S_X \times S_{Vel}$	$S_X \times S_Y \times S_{X,Vel} \times S_{Y,Vel} \times S_\alpha \times S_{\alpha,Vel} \times S_{Leg_1} \times S_{Leg_2}$
Goals <sup>a</sup>	$G_{Height} :$ $1.0 \leq s_{Height} \leq 3.0$	$G_{Position} :$ $0.966 \leq s_X \leq 1.0$ $-0.259 \leq s_Y \leq 0.259$  $G_{Stabilization} :$ $-0.25 \leq s_\alpha \leq 0.25$	$G_{Position} :$ $0.45 \leq s_X \leq 1.0$ $G_{Velocity} :$ $-0.03 \leq s_{Vel} \leq 0.07$	$G_{Position} :$ $-0.2 \leq s_X \leq 0.2$ $-0.05 \leq s_Y \leq 0.05$ $G_{Velocity} :$ $-0.1 \leq s_{X,Vel} \leq 0.1$ $-0.1 \leq s_{Y,Vel} \leq 0.1$ $G_{Stabilization} :$ $0.1 \leq s_\alpha \leq 0.1$ $-0.1 \leq s_{\alpha,Vel} \leq 0.1$ $G_{LegsGrounded} :$ $s_{Leg_1} = 1, s_{Leg_2} = 1$
Distance Annotations	$dist_{Height}(s) = \frac{1}{\sqrt{(s_{Height} - 1.5)^2}}$	$dist_{Position}(s) = \frac{1}{\sqrt{(s_X - 1)^2}}$	$dist_{Velocity}(s) = \frac{1}{\sqrt{(s_X - 0.07)^2}}$	$dist_{Position}(s) = \sqrt{s_X^2 + s_Y^2}$ $dist_{Velocity}(s) = \sqrt{s_{X,Vel}^2 + s_{Y,Vel}^2}$ $dist_{Stabilization}(s) = \sqrt{s_\alpha^2}$
Constraints	-	-	-	$\delta_{Crash} = 1$ $C_{Crash} = \{(s, a)   (s_X \leq -0.2 \vee s_X \leq 0.2) \wedge s_Y \leq 0.2\}$
PPO Hyper-parameters <sup>c</sup>	-	-	(use_sde=True)	batch_size=32, n_steps=1024, n_epochs=4, gae_lambda=0.98, gamma=0.999, ent_coef=0.01 <sup>d</sup>

<sup>a</sup> We only state the relevant features for each goal. There are no further restrictions on other state space features.

<sup>b</sup> We expand the state space by  $S_{Height}$  with  $height = -\cos(\theta_1) - \cos(\theta_2 + \theta_1)$  through our  $state(...)$  function to enable the specification of a height goal.

<sup>c</sup> If not stated differently, we use the default PPO parameters from Stable Baselines (DLR-RM, 2024b). Most importantly, these are: gamma=0.99, learning\_rate=0.0003, batch\_size=64, n\_steps=2024, n\_epochs=10, gae\_lambda=0.95, ent\_coef=0.0, use\_sde=False

<sup>d</sup> We use optimized hyperparameters from RLZoo (DLR-RM, 2024a), a training framework with published hyperparameters

we extract goal states as described for our running example in Section 4. Table 1 lists the goals and hyperparameters for reproducibility.

Based on the identified goals, we define up to three goal tree specifications for each case study. With each iteration, we increase the specification details refining the previous tree similar to Figure 3. Our first specification consists of a single root leaf node, which includes the goal space that is the intersection of all identified goals. Second, we refine this root node by the  $\wedge$ -operator into subgoal nodes. Finally, we create a third specification by annotating the leaf nodes with distance metrics as given in Table 1. For the Acrobot case study, we have identified only one goal and, therefore, we do not include an  $\wedge$ -operator refined specification. Additionally, for the LunarLander case study, we impose a safety constraint (see Table 1)

representing the crash penalty from the baseline environment.

From each specification, we automatically construct an environment variant. We proceed to train agents for each variant and measure their performance by inspecting the individual success rate for each goal. We do so by defining success based on a goal space  $G$ , counting how often the agent reaches the goal at the terminal state  $s_{T,i}$  over  $N$  episodes:

$$success_G(\pi) = \frac{1}{N} \sum_{\tau_i \sim \pi} \begin{cases} 1 & , s_{T,i} \in G \\ 0 & else \end{cases}$$

Finally, we manually tune the weights of the reward components of those case studies, in which the agents converge to a local maximum and are therefore unable to learn all goals.



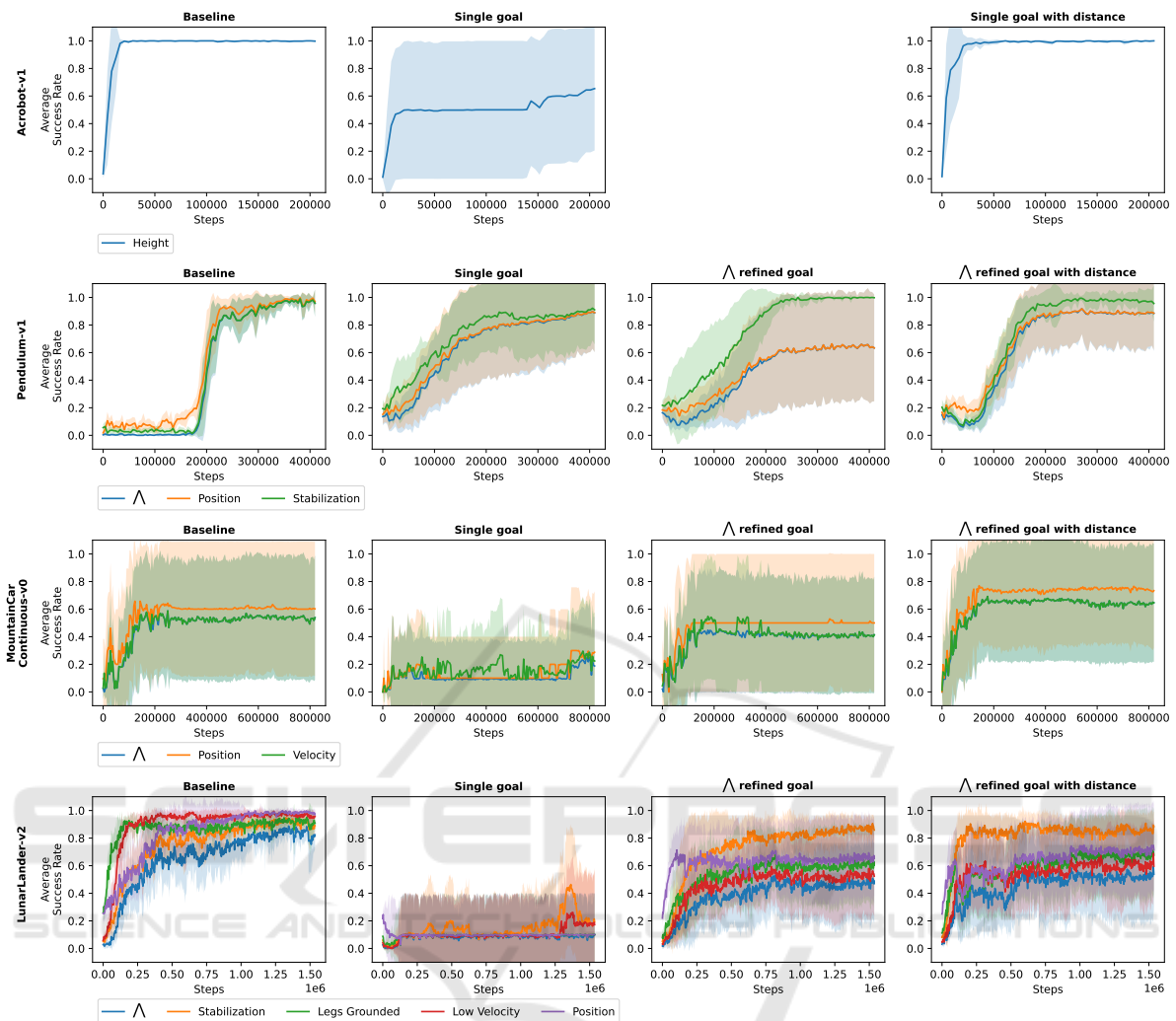


Figure 4: Our results show the success rate of the individual goals with one row per case study. Each column encompasses a single specification scenario with increasing specification details from left to right.

For reproducibility, we adhere to the following setup. Each result is averaged over 10 independent runs with random initialization. We train our agents with PPO from Stable Baselines (DLR-RM, 2024b) with tuned hyperparameters Table 1. We normalize the reward function weights for each environment as follows: for each of the  $N$  reward components of a node, the weight is  $\frac{1}{N}$ ; the hard safety constraint from the LunarLander results in a penalty of  $-1$ ; each distance reward shape is divided by a specified maximum distance  $d$  to the goal.

## 6.2 Discussion of Results

The results of our experiments across the four case studies are depicted in Figure 4. Each row corresponds to one case study, while each column repre-

sents one of the four experimental scenarios: *baseline*, *single goal*,  $\wedge$ -operator refined goal, with *distance*. The graphs illustrate the success metric for each goal listed in Table 1, providing comparability across the scenarios, even though the *baseline* and *single goal* scenario do not encompass these goals directly. The following paragraphs present and discuss the results for each case study individually and we conclude with a summary of our findings.

**Acrobot.** We identify a single goal to reach a specific height. For the Acrobot environment. Thus, we do not apply the  $\wedge$ -operator. The *baseline* scenario shows convergence, achieving the height goal consistently over all runs. The *single goal* scenario reaches about 60 % success rate at the end of training with high variance. This high variance is caused because

in only 6 out of 10 training runs, the agents can reach the goal. The agents do not receive any reward in the remaining four runs due to the sparsity of the reward. Therefore, the agents cannot converge to a solution. Incorporating the *distance* annotation in the last scenario resolves this exploration issue effectively, and we achieve similar performance to the baseline.

**Pendulum.** In the *baseline* of the Pendulum environment, the goals are learned consistently. The *single goal* scenario achieves success in 8 out of 10 runs. The  $\wedge$ -operator refined scenario presents a more intricate result, in which the refined sparse reward facilitates reliable learning of the stabilization goal but reaching the target position is more difficult to achieve. Here, half of the 10 agents converge to a local optimum where they stabilize the pendulum without achieving the position goal. The position *distance* annotation alleviates this problem by providing an additional dense reward. However, only by manually weighting the position and stabilization rewards, we can resolve the conflict between the goals. With this, we achieve more precise convergence to the goals compared to the baseline as shown in Figure 5.

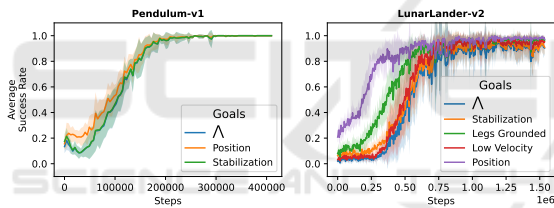


Figure 5: Results with manual weights for the Pendulum ( $\omega_{\text{position}} = 0.8$ ,  $\omega_{\text{stabilization}} = 0.2$ ) and LunarLander ( $\omega_{\text{stabilization}} = 50$ ,  $\omega_{\text{legs\_grounded}} = 20$ ,  $\omega_{\text{low\_velocity}} = 100$ ,  $\omega_{\text{position}} = 200$ ) case studies.

**MountainCarContinuous.** Originally, the MountainCarContinuous case study is built to introduce an exploration challenge. This challenge is evident in the results of the *baseline* scenario, where the goal is achieved in only 4 out of 10 runs. We observe similar difficulties across all three of our specification scenarios. In our  $\wedge$ -operator refined with *distance* scenario, we achieve the goal in 6 out of 10 runs. Improving the exploration and specific tuning of PPO can resolve the exploration challenge for the baseline (Kapoutsis et al., 2023). We strongly believe that our method yields comparable results with similar tuning efforts, although this remains to be tested. Nevertheless, we observe that learning performance improves based on our refinements with increasing specification detail.

Furthermore, during the design of the case study, we have experienced the performance degradation for a different goal tree specification. Specifically, an-

notating the position goal node with the  $x$  distance to the goal states introduces unexpected complexity into the problem. Here, the trained agents consistently learned to stand still at the bottom of the mountain, while avoiding the negative reward required to attempt climbing the mountain at high velocity.

**LunarLander.** The LunarLander case study involves the learning of four goals. The *baseline* scenario reaches approximately 80 % success in achieving the overall  $\wedge$ -operator goal. While the sparse reward proves insufficient in the *single goal* scenario, the  $\wedge$ -operator refined scenario enhances this sparsity by rewarding each goal individually. Despite this, achieving all goals simultaneously remains challenging. Our *distance annotations* scenario mitigates this issue for the stabilization goal. Nonetheless, in most runs, the agents converge to a local optimum, stabilizing without reaching the landing position. Manual weighting corrects this imbalance by prioritizing the position goal as depicted in Figure 5 and we achieve more precise convergence compared to the baseline.

To conclude our evaluation, we identify known limitations of our method. Subsequently, we summarize our results with respect to the two key questions introduced at the beginning of this section.

Our method entails two limitations. First, specifying complex goals in the state space requires a state structure for which it is possible and sufficient to handcraft these goals. While this is theoretically possible, in practice it hinders the use of our method in high-dimensional state spaces such as learning from raw pixels. Second, we have experienced that specific refinements of goals can lead to undesired and unexpected behavior of the trained agents as described in the results of the MountainCarContinuous case study. However, this does not contradict our approach of iterative environment design but rather emphasizes the need for iterations. Nevertheless, it is important to recognize the possibility of degrading results after a refinement.

Finally, we examine the results regarding our two key questions. First, we have shown the ability to specify and learn goal tree specifications sufficiently for all four case studies. Our results show that we can consistently learn the goals in three out of four cases with manually defined weights. For the remaining MountainCarContinuous case study, we have achieved similar results compared to the baseline and we have learned to reach the goals in 6 out of 10 runs. Second, the ability to specify goal trees and automatically construct environments, enables iterations by

evaluating a specification from trained agents. Our findings indicate that single goals with sparse rewards, often, do not provide enough feedback for effective learning. However, the results of our  $\wedge$ -operator consistently improve by providing a refined reward for the goal. Additionally, the  $\wedge$ -operator enables to balance possibly conflicting goals by weighting the inner reward components while the distance annotations help to guide the agent towards otherwise challenging goals. In contrast, we would like to point out that goal tree refinements do not always yield improvements in learning the goals. Therefore, iterations can also include reverting or adapting prior changes to the specification. Nevertheless, our results show that we can iteratively improve on learning the specified goals. In the following section, we conclude and present future work.

## 7 CONCLUSION

In this work, we have introduced iterative environment design for reinforcement learning based on goal-oriented specification (Schwan et al., 2023). We evolve goal-oriented specification and make it practical with two contributions. First, we introduce our automated method to construct RL environments from goal tree specifications. Thereby, we enable the training of agents from these specifications to evaluate their behavior for future improvements. Second, we enable iterative goal tree refinements by introducing definitions for leaf nodes, the  $\wedge$ -operator and annotations. To evaluate our method, we have trained agents in four case studies with up to three specification scenarios each. With manually tuned weights of the reward components, we achieve goal success rates similar to the baselines but with higher precision. Finally, our results show that goal tree refinements can be used to iteratively improve the learning of specified goals. Through iterative environment design, we oppose the common trial-and-error practice to facilitate the application of reinforcement learning.

In future work, we plan on automating the manual weighting of reward components from our  $\wedge$ -operator to further reduce time-consuming manual tasks. Moreover, we aim at enhancing our specification method to be practical for high-dimensional state spaces. Finally, introducing new operators can enable specifying and learning temporal abstractions. With this, we follow our idea to overcome the common trial-and-error practice and facilitate the development of RL solution for domain experts.

## ACKNOWLEDGEMENTS

This work has been partially funded by the Federal Ministry of Education and Research as part of the Software Campus project *ZoLA - Ziel-orientiertes Lernen von Agenten* (funding code 01IS23068).

## REFERENCES

- Ahmad, K., Abdelrazek, M., Arora, C., Bano, M., and Grundy, J. (2023). Requirements engineering for artificial intelligence systems: A systematic mapping study. *Information and Software Technology*, 158:107176.
- Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2017). Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058.
- Cai, M., Xiao, S., Li, B., Li, Z., and Kan, Z. (2021). Reinforcement Learning Based Temporal Logic Control with Maximum Probabilistic Satisfaction. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 806–812.
- Chane-Sane, E., Schmid, C., and Laptev, I. (2021). Goal-Conditioned Reinforcement Learning with Imagined Subgoals. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pages 1430–1440.
- Ding, H., Tang, Y., Wu, Q., Wang, B., Chen, C., and Wang, Z. (2023). Magnetic Field-Based Reward Shaping for Goal-Conditioned Reinforcement Learning. *IEEE/CAA Journal of Automatica Sinica*, 10(12):2233–2247.
- DLR-RM (2024a). RL Baselines3 Zoo: A Training Framework for Stable Baselines3 Reinforcement Learning Agents. <https://github.com/DLR-RM/rl-baselines3-zoo>. [Last accessed on July 17th, 2024].
- DLR-RM (2024b). Stable-Baselines3. <https://github.com/DLR-RM/stable-baselines3>. [Last accessed on July 17th, 2024].
- Everett, M., Chen, Y. F., and How, J. P. (2021). Collision avoidance in pedestrian-rich environments with deep reinforcement learning. *IEEE Access*, 9:10357–10377.
- Farama Foundation (2024). Gymnasium: An API standard for reinforcement learning with a diverse collection of reference environments. <https://gymnasium.farama.org/>. [Last accessed on July 12th, 2024].
- Florensa, C., Held, D., Geng, X., and Abbeel, P. (2018). Automatic goal generation for reinforcement learning agents. In *International conference on machine learning*, pages 1515–1528.
- Hahn, E. M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., and Wojtczak, D. (2019). Omega-Regular Objectives in Model-Free Reinforcement Learning. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 395–412.

- Hammond, L., Abate, A., Gutierrez, J., and Wooldridge, M. (2021). Multi-Agent Reinforcement Learning with Temporal Logic Specifications. In *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems*, pages 583–592. ACM. arXiv:..
- Jothimurugan, K., Alur, R., and Bastani, O. (2019). A composable specification language for reinforcement learning tasks. *Advances in Neural Information Processing Systems*, 32.
- Jothimurugan, K., Bansal, S., Bastani, O., and Alur, R. (2021). Compositional reinforcement learning from logical specifications. *Advances in Neural Information Processing Systems*, 34:10026–10039.
- Jurgenson, T., Avner, O., Groshev, E., and Tamar, A. (2020). Sub-Goal Trees a Framework for Goal-Based Reinforcement Learning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5020–5030.
- Kapoutsis, A. C., Koutras, D. I., Korkas, C. D., and Kosmatopoulos, E. B. (2023). ACRE: Actor-Critic with Reward-Preserving Exploration. *Neural Comput. Appl.*, 35(30):22563–22576.
- Li, X., Vasile, C. I., and Belta, C. (2017). Reinforcement learning with temporal logic rewards. *IEEE International Conference on Intelligent Robots and Systems*, pages 3834–3839.
- Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations : Theory and application to reward shaping. *16th International Conference on Machine Learning*, 3:278–287.
- Okudo, T. and Yamada, S. (2021). Subgoal-Based Reward Shaping to Improve Efficiency in Reinforcement Learning. *IEEE Access*, 9:97557–97568.
- Okudo, T. and Yamada, S. (2023). Learning Potential in Subgoal-Based Reward Shaping. *IEEE Access*, 11:17116–17137.
- Retzlaff, C. O., Das, S., Wayllace, C., Mousavi, P., Afshari, M., Yang, T., Saranti, A., Angerschmid, A., Taylor, M. E., and Holzinger, A. (2024). Human-in-the-Loop Reinforcement Learning: A Survey and Position on Requirements, Challenges, and Opportunities. *J. Artif. Intell. Res.*, 79:359–415.
- Rojijers, D. M., Vamplew, P., Whiteson, S., and Dazeley, R. (2013). A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113.
- Schaul, T., Horgan, D., Gregor, K., and Silver, D. (2015). Universal value function approximators. In *32nd International Conference on Machine Learning*.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. *CoRR*.
- Schwan, S., Klös, V., and Glesner, S. (2023). A Goal-Oriented Specification Language for Reinforcement Learning. In *International Conference on Modeling Decisions for Artificial Intelligence*, pages 169–180. Springer.
- Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.
- Van Lamsweerde, A. (2001). Goal-oriented requirements engineering: A guided tour. *Proceedings of the IEEE International Conference on Requirements Engineering*, pages 249–261.