CAPS: Comprehensible Abstract Policy Summaries for Explaining Reinforcement Learning Agents

Joe McCalmone
Wake Forest University
Winston-Salem, NC, USA
mccajl18@wfu.edu

Thai Le
The Pennsylvania State University
University Park, PA, USA
tql3@psu.edu

Sarra Alqahtani
Wake Forest University
Winston-Salem, NC, USA
sarra-alqahtani@wfu.edu

Dongwon Lee
The Pennsylvania State University
University Park, PA, USA
dongwon@psu.edu

ABSTRACT

As reinforcement learning (RL) continues to improve and be applied in situations alongside humans, the need to explain the learned behaviors of RL agents to end-users becomes more important. Strategies for explaining the reasoning behind an agent's policy, called *policy-level explanations*, can lead to important insights about both the task and the agent's behaviors. Following this line of research, in this work, we propose a novel approach, named as CAPS, that summarizes an agent's policy in the form of a directed graph with natural language descriptions. A decision tree based clustering method is utilized to abstract the state space of the task into fewer, condensed states which makes the policy graphs more digestible to end-users. This abstraction allows the users to control the size of the policy graph to achieve their desired balance between comprehensibility and accuracy. In addition, we develop a heuristic optimization method to find the most explainable graph policy and present it to the users. Finally, we use the user-defined predicates to enrich the abstract states with semantic meaning. We test our approach on 5 RL tasks, using both deterministic and stochastic policies, and show that our method is: (1) agnostic to the algorithms used to train the policies, and (2) comparable in accuracy and superior in explanation capabilities to existing baselines. Especially, when provided with our explanation graph, end-users are able to accurately interpret policies of trained RL agents 80% of the time, compared to 10% when provided with the next best baseline. We make our code and datasets available to ensure the reproducibility of our research findings: `https://github.com/mccajl/CAPS`

KEYWORDS

Reinforcement Learning, Explainable AI, XRL, Autonomous

ACM Reference Format:
Joe McCalmone, Thai Le, Sarra Alqahtani, and Dongwon Lee. 2022. CAPS: Comprehensible Abstract Policy Summaries for Explaining Reinforcement Learning Agents. In *Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), Online, May 9–13, 2022*, IFAAMAS, 9 pages.

Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), P. Faliszewski, V. Mascardi, C. Pelachaud, M.E. Taylor (eds.), May 9–13, 2022, Online.

1 INTRODUCTION

Neural networks have been successfully applied to reinforcement learning (RL) problems in areas of control [2] and games [15]. Due to impressive performance in these areas, RL has started appearing in performance-sensitive real-world applications like chip design [14], server management [11], and robotics [19]. However, the black-box nature of neural network decisions increases the need for end-user trust when deploying a new RL system [6]. Explainable reinforcement learning (XRL) attempts to build this trust by explaining the behavior of an RL agent that uses neural networks in its decision making process.

In general, we argue that a truly useful XRL has to satisfy several desiderata. First, XRL has to explain the entire policy (i.e., behavior) of the agent to end-users who likely have insufficient knowledge of RL. Policy-level explanations attempt to reveal the policy of an agent for many different regions of the state space, and "before" the agent has taken any actions [5, 23, 26]. However, existing XRL methods often focus on explaining the decisions of an RL agent in specific instances [8, 18]. Second, XRL needs to be generalizable for both deterministic and stochastic RL policies in continuous or discrete state spaces. However, existing XRL approaches explain only deterministic policies [26] or discrete state spaces [23]. Third, XRL needs to deliver explanations, which end-users can interpret, with as minimal user intervention as possible. Moreover, the end-users should be given control over the size and thoroughness of the explanation based on their needs. Finally, the explanations must accurately describe the agent's policy, which can be evaluated using the fidelity test [13].

In this paper, satisfying these desiderata, we propose a generalizable policy-level explanation approach, named as **Comprehensible Abstract Policy Summaries** (**CAPS**), that explains both stochastic and deterministic policies of an RL agent and displays the policy as a directed graph, embedded with intuitive natural language (NL) explanations. CAPS first collects, from the user, simple NL predicates which describe potential aspects of the agent's state (e.g., "car moves right" or "car stops at the stop sign"). CAPS then collects no more than 500 timesteps from the RL agent trajectories. In order to make the explanation process tractable, CAPS uses a clustering algorithm, CLTree [9], that abstracts the agent's states into a hierarchy of different configurations of clusters (C). Each cluster groups the similar states into one abstract state. A heuristic

optimization technique is developed to select the best configuration of the clusters, which is determined by the accuracy of the state transitions and the end-user interpretability. For each cluster, $c \in C$, CAPS also identifies whether the agent considers the states in the cluster to be *critical*, extending the methodology proposed in [6]. Then, using the generated clusters C, CAPS forms the agent's policy (π) and transition function as a directed graph, where the nodes are the clusters of states, forming abstract states, and the edges represent the actions chosen by π, as well as the probability of transitioning from one abstract state to the next. To enrich the generated graph with more semantic meaning, CAPS labels the abstract states (i.e. graph nodes) with concise NL explanations using the user-defined languange predicates and Boolean algebra. Lastly, by controlling the height of the CLTree, CAPS gives the end-user the choice of generating different policy graphs with different sizes such that each size corresponds to different levels of abstraction.

To demonstrate the utility of CAPS, we use five popular RL environments. We trained the RL agents in the environments using an algorithm for stochastic policies, PPO [21] and an algorithm for deterministic policies, DQN [16]. Our experimental results indicate that CAPS is generalizable with minimal user interventions. To measure how close the explanation graphs are to the agent's policy, we ran fidelity testing for each environment and compared it against two baselines which share the most in common with our approach [23, 26]. The results show that the accuracy of the policy graphs produced by CAPS is comparable to those produced by the baselines. We also conducted a user study to test the "real" explainability of the generated graphs with different abstraction levels against the two baselines. Results show that users presented with CAPS graphs identify the correct state and next action of an agent 80% of the time, compared to just 10% for the next best baseline.

Our contributions are summarized as follows:

- We introduce a policy explanation approach for creating abstract policy graphs with different state abstraction levels, which provides the end-users with more control over the size of the graph.
- We provide an optimization heuristic for maximizing the accuracy of our policy graphs, while minimizing the size of the graph.
- We highlight which abstract states the agent considers most important to the task which builds trust in end-users [6].
- We propose a novel algorithm for generating a single natural language representation for each abstract state. Our method overcomes the issue of previous natural language grounding methods, where too many contradictory state regions can degrade the quality of the explanation as in [5].

2 RELATED WORK

Researchers have highlighted important pixels in a frame of an Atari game [7], [25], [4], and generated contrastive Atari frames which would cause the agent to take another action [18] . [8] use reward decomposition to identify the part of the goal a specific action was meant to achieve. Other methods identify states which an RL agent determines is important for the outcome of the episode. The entropy of the agent's policy and the maximum difference of the value function have been used to determine states in which a certain action needs to be taken to avoid failure [1], [6]. Highlighting such states improves a user's trust in the agent [6].

For policy-level explanations, Liu et al. in [10] generate decision trees that mimic the agent's policy, and then interpret the rules formed by the learned tree.Structured models of the environment have been created in [12] to trace the outcomes of the agent's policy and produce explanations. These models are not generalizable across tasks and are difficult to create in large environments, or in environments where the complete state transition dynamics are unknown. Another method has been proposed in [24] to generate an explanation of a policy by comparing it to a user-generated 'foil' policy. It, however, requires the end-user to have a foil policy in question and know the dynamics of the environment. In [5], policy-level explanations have been generated in response to user queries. They also propose a method for translating state regions to natural language. Our proposed approach improves their methods for translating state regions, and generates policy explanations which encompass more scenarios than just those that the user queries.

An abstracted Markov chain has been built in [23] and is displayed as a directed graph to explain the agent's policy. This approach clearly visualizes the structure of the environment, and the agent's path while traversing it. However, the generated policy graph is not interpretable beyond its shape, because the nodes representing abstract states are not given natural language labels. For stochastic policies and larger environments, the produced graphs can quickly become too large [23].

Zahavy et.al [26] create state abstractions in Atari environments using clustering. Their method produces abstract policy graphs, but requires heavy manual feature engineering and knowledge of the environment. In addition, the quality of the abstract groupings is dependent on the engineered features, and they do not provide a way to interpret their policy graphs beyond environments with image representations. Our method alleviates these weaknesses by clustering the state abstractions before any feature engineering is applied, but the two can be used in conjunction if the user already has extensive knowledge of the environment. The comparison between our proposed CAPS and prior work is listed in Table 1.

3 BACKGROUND

3.1 Reinforcement Learning

An RL agent learns by acting within an environment. Their interaction can be characterized as a Markov Decision Process (MDP) described by the tuple (S, A, P, R), where S is the set of states, A is the set of actions available to the agent, P is the transition function such that $P(s_t, a_t)$ produces a distribution over all possible next states at time t, and R is the set of all possible rewards that an agent can receive for actions taken in states. The goal of an agent is to learn the policy, $\pi(s_t) = a_t$, which would maximize the total discounted reward over the whole task. To help discover the optimal policy, an agent learns to approximate the value function, $v(s_i)$, which is the expected discounted reward that can be gained by being in the state s_i and following the policy, π. Formally, the value function is:

$$v(s_i) = \mathbb{E}\left(\sum_{t=i} \gamma^t R(s_t, \pi(s_t), s_{t+1}) \right) \tag{1}$$

Algorithm	Policy Explanation	Explains the Transition Function	Targets End-users	Assumes Minimal End-user Knowledge	Highlights Critical States	Explains Continuous State Spaces	Explains Stochastic Policies	Optimizes Size of Explanation	Natural Language Explanations
Topin and Veloso [23]	✓	✓		✓					
Hayes and Shah [5]			✓	✓		✓			✓
Zahavy et al. [26]	✓	✓					✓		
Liu et al. [10]			✓			✓			
Madumal et al. [12]	✓	✓	✓						✓
Van der Waa et al. [24]	✓		✓						✓
CAPS	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 1: Summary of Surveyed Related Works vs. CAPS

Note that the state-value function can be obtained from the action-value function: $V_\pi(s_0) = Q_\pi(s_0, \pi(s_0))$. These are used in methods based on Q-Learning, Sarsa(λ), and actor-critic methods [22]. Therefore, the value-function is generally available alongside the policy of a trained agent. In the case of stochastic policies, π produces a probability distribution over A, $\pi(\cdot|s)$, which can then be sampled to choose a_t. The goal of CAPS is to display information about the policy, π, and the transition function, P, to the user as an easily understood directed graph.

3.2 CLTree

CAPS relies on clustering, an algorithm for finding the natural groupings of data points within the entire set. Specifically, we use the clustering algorithm, CLTree [9], which trains a decision tree to separate the data that belongs to the set, from data which is artificially inserted into the set uniformly. By learning the feature boundaries which separate real data from fake data, the nodes in the tree form natural clusters. In addition, these clusters can be combined or separated by traversing up or down the tree. Therefore, by pruning the tree, each cluster can be more inclusive or exclusive according to the user's needs.

4 THE PROPOSED APPROACH: CAPS

The main deliverable from CAPS is a directed graph to describe the agent's policy in NL predicates. We formulate the output graph G as $G = (V, E)$ to explain the RL policy components, states, actions, and transiting probabilities between the states. V is a set of vertices representing the RL agent's states S from the MDP representation of the environment. E is the edges connecting the vertices matching the actions $A = \pi(S)$ in the MDP and defined as $E \subseteq \{(v_1, v_2)|(v_1, v_2) \in V^2$. Since G is a directed graph, we can annotate E with the transition probabilities P as defined in the MDP.

Directly mapping the states S, actions A, and transitions P from the original MDP to V and E in G is intractable for the end-user interpretation. For instance, generating G for an episode of 100 timesteps collected from a policy π could, in the worst case, include 100 vertices to represent each state in the episode. Given the definitions of the RL agent's policy π and the graph G, we frame this problem within CAPS framework as a four step process as depicted in Figure 1 and explained in the following subsections.

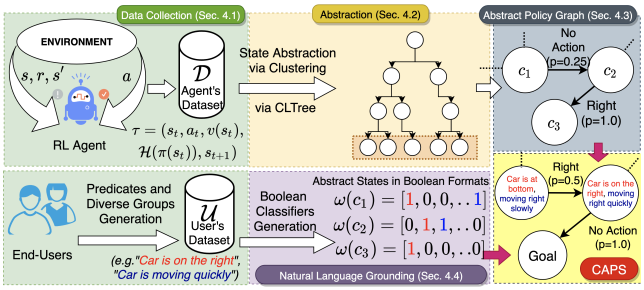


Figure 1: The overall architecture of the CAPS approach.

4.1 Data Collection

In this step, we collect two small datasets, U, from the end-user, and D, from the RL agent's execution. The user is asked to create natural language predicates that concern them about the environment and the agent's task. For example, for a self-driving car agent, the user could give predicates like "the car is moving fast", "there is a stop sign", or "car on the right". Those predicates will be used later to append G with semantic meaning that supports the end-user's interpretation. For the agent's dataset D, we collect a small number (< 500) of timesteps from its execution traces, possibly collected over multiple episodes, consisting of tuples, $\tau = (s_t, a_t, v(s_t), \mathcal{H}(\pi(s_t)), s_{t+1}) \in D$. s_t is the state of the agent at time t, which will be used to form the vertices V of G, while a_t is the action that the agent's policy chooses based on the current state, $a_t = \pi(s_t)$, and will be mapped to the edges E in G. The state value $v(s_t)$ is collected to estimate how good it is for the agent to be in state s_t for the future reward from Eq.1. This value helps groups states which the agent views as similarly important to the task's success (Section 4.2). The policy entropy $\mathcal{H}(\pi(s_t))$ defines the entropy of the probability distribution generated by π in state s_t. We use the policy entropy to highlight clusters of states as critical (Section 4.2.1). For deterministic policies which do not output a probability distribution, we instead use the maximum difference of the Q-function among actions as a proxy for policy entropy, as in [6]. This is defined as $max_{a_i,a_j \in A}(Q(s_t, a_i) - Q(s_t, a_j))$, where $Q(s_t, a_i)$ is the learned value of taking action a_i in state s_t.

4.2 Policy State Abstraction via Clustering

The motivation of CAPS is to generate more human-interpretable graphs than the Markov chains made from the base MDP and the

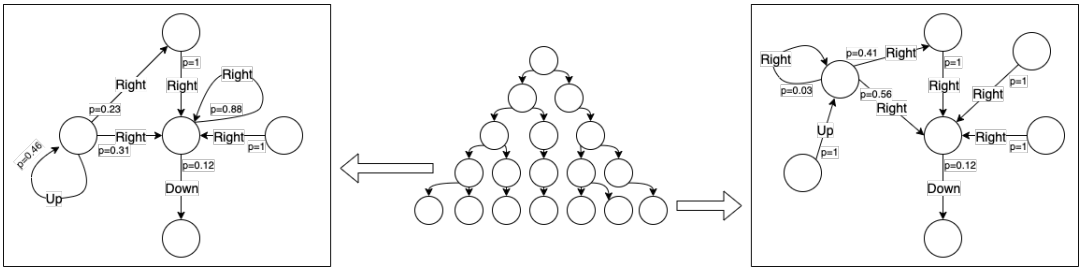


Figure 2: Example of Abstract Policy Graphs produced by CAPS before translation for CLTree of height 4 and 5 (left, right)

RL agent's policy. Therefore, the number of nodes in the resulted graph of CAPS should be much lower than the number of distinct states in the base MDP. To achieve this, we adopt the notion of state abstraction from RL literature [17], [22]. State abstraction involves grouping the grounded states of the agent's MDP into groups of similar states, reducing the size of the entire state space. Prior work [23] has abstracted the states using feature importance or significance. In CAPS, we instead use a clustering algorithm eliminating the need to manually engineer the state features as in previous work. We can directly apply clustering algorithms to abstract the state space if we assume that groups of similar states S_1, S_2 share similar policies $\pi(S_1) = \pi(S_2)$ or value functions $v(S_1) = v(S_2)$ as in [17]. This assumption, however, does not always hold, thus we include the value function $v(s_t)$ and action a_t from the collected tuples τ for more accurate clustering.

Using CLTree algorithm [9], we cluster each $\tau \in D$ into similar groups. The generated leaf clusters C in the tree represent the abstract states. As a result, each abstract state $c \in C$ groups all states which are *interchangeable* under the agent's policy such that either the agent behaves similarly when starting in any $s \in c$, or $\pi(\forall s \in c) = a$ [26], [23], or the agent assigns similar values to each $s \in c$ [23]. The advantage of using CLTree clustering over K-means or a modified K-means [26] is that the user has direct control over the size of the graph's nodes V. From a finished CLTree pruned to a specific depth, traversing the tree can increase or decrease the level of abstraction, giving a dynamic understanding of the policy. As an example, Figure 2 shows two policy graphs, without natural language labels, at two sequential levels of the tree.

4.2.1 Heuristic Optimization for Clustering. To increase the usability of CAPS for users with minimal to no knowledge about RL and CLTree, we develop a heuristic optimization technique to find the optimal set of clusters C^*. Our heuristic is motivated by the fact that as the policy graph G grows in the number of clusters C (i.e. $|V|$), the error between the true environment transition function P from the MDP and the one estimated by G, as well as the error between the agent's true policy π and that presented in the graph, decreases. However, in order for G to be interpretable to a human end-user, we must limit the number of vertices V in the graph [1].

Let L be the set of tree heights, and C_l be the set of clusters at a height, $l \in L$. We wish to choose l such that the clusters in $C^* = C_l$ form an accurate policy graph with respect to the value function $v(s)$ and the policy of the agent π within each cluster. In addition, we want to penalize the size of the graph, so that the policy graph G remains interpretable for humans with less vertices V. To achieve that, we introduce two heuristics; the value score and the cluster policy. Informally, the *value score* represents the error between the environment's transition function $P*$ as estimated by the clusters, and the true transition function P in the MDP. It does this by comparing the true value functions of the states in each cluster $v(s) \forall s \in C$ and the estimated value function according to the estimated transition function. This heuristic is inspired by the Value Mean Square Error(VMSE) evaluation criterion in [26]. The second heuristic is the entropy of the *cluster policy*. The *cluster policy*, $\Pi(c)$ defines the probability of taking an action, a, given a cluster, c, as the proportion of $s \in c$ such that $\pi(s) = a$. We minimize the average entropy of the *cluster policy*, $\mathcal{H}(\Pi(c))$, across all $c \in C$, so that it is clear to the end-user which action the agent will take in the abstract state, c.

Recall, $v(s)$ is the value of the state s, as determined by the agent's value function (Eq.1). We formally define the *value score*, $\zeta(C(l))$, as:

$$\zeta(C(l)) = \frac{1}{|C(l)|} \sum_{c \in C(l)} (V_{gt}(c) - V_{est}(c))^2 \tag{2}$$

where $V_{gt}(c)$ is the ground-truth value of an abstract state, c, found by:

$$V_{gt}(c) = \frac{1}{|c|} \sum_{s \in c} v(s)$$

and $V_{est}(c)$ is the estimated value of an abstract state, c, found by:

$$V_{est}(c) = \gamma \sum_{c_i \in C(l)} P^*(c, c_i) V_{gt}(c_i)$$

where γ is the discount factor from Eq. 1 and $P^*(c, c_i)$ is the probability the agent transitioning from cluster $c \in C_l$ to cluster $c_i \in C_h$, estimated as the proportion of (s_t, a_t) pairs $\in c$ which transition to $s_{t+1} \in c_i$:

$$P^*(c, c_i) = \frac{1}{|c|} \sum_{(s,a,s_{t+1}) \in c} I(s_{t+1}) \tag{3}$$

where I is an indicator function deciding whether s_{t+1} is in c_i.

Then, we formally define the height of the tree, l, pertaining to the optimal clusters as:

$$\arg\min_{l \in L} \left[\zeta(C(l)) + \left(\frac{1}{|C(l)|} \sum_{c \in C(l)} \mathcal{H}(\Pi(c)) \right) + \alpha |C(l)| \right] \tag{4}$$

where α is a parameter controlling the penalty of larger policy graphs and $\mathcal{H}(\Pi(c))$ is the entropy of the cluster policy.

[1] See Appendix B for the analysis study of the heuristic optimization

4.3 Abstract Policy Graph

Given a set of abstract states, we create a policy-level explanation by modeling the policy as a directed graph G. Each vertex $v \in V$ represents an abstract state from the clusters generated by CLTree and the edges E are transitions induced by a single action from one abstract state to another, each accompanied by a transition probability. We estimate the transition function $P*$ from Eq.3. We use Algorithm2 in [23] to append the actions and transition probabilities to our graph G. We, however, modify the graph generation algorithm in [23] to include the stochastic transition functions by attaching the actions taken to the edges with their probabilities, instead of to the nodes. This change increases the readability of CAPS's generated graph since the nodes have the NL predicates to describe the abstract states and the edges describe how likely the agent will take certain actions. Figure 2 shows what an example G looks like before incorporating the NL labels to the nodes, at two different heights of the CLTree.

4.4 Natural Language Grounding

In order to provide interpretable explanations of the agent's abstract policy graph G, we ground the graph nodes (i.e. abstract states) in NL. We use a set of Boolean classifiers that we build based on the user's given predicates in U. We use these classifiers to translate internal knowledge of the MDP states within the abstract states into NL through two levels of translations; MDP state translation and abstract state translation.

4.4.1 MDP State Translation . We modify the method of [5] for grounding state regions in NL to improve the explanation quality. Recall, our collected preliminary data U has the environment-specific user-predicate. Those predicates are used to create binary classifiers evaluating the features of each state. The semantic meaning of the *boolean predicates* should correspond to some aspect of the state space which we are interested in using to explain the agent's action. Unlike [26] which also uses feature engineering to create semantic meaning, in CAPS, the set of predicates does not affect the accuracy of the transition function, or the fidelity of the policy graphs. This is because we apply the NL after clustering instead of clustering the engineered features. Hence, only the quality of the NL explanations given to each abstract state is affected by the user-predicates.

Given a set of user-predicates,F, which are binary functions, we map an MDP state $s \in c$ into binary vector, $\omega(s)$, with length $|F|$, where each element represents whether s satisfies the particular user-predicate, $f \in F$. $\forall c \in C$, and $\forall s \in c$, we map s to $\omega(s)$ through:

$$\hat{\omega(s)} = \left[\begin{cases} 1, & \text{if } f(s) = \text{True} \\ 0, & \text{otherwise} \end{cases} \right] \forall f \in F \tag{5}$$

4.4.2 Abstract State Translation. In Eq.5, each grounded state in the abstract state $\forall s \in c$ is translated into a binary vector $\omega(s)$ of NL predicates. We then condense the set of all binary vectors in the cluster $\{\omega(s) | s \in c\}$, into a single vector of predicate values, $\omega(c)$, which provides a concise explanation of the cluster c (Algorithm 1). Algorithm 1 has three steps; finding the frequent predicates for each abstract state, calculating the correlation between those predicates, and translating them into one NL sentence. We label a predicate in a cluster as frequent or not with Eq.6.

Algorithm 1 Condense Semantic Meanings into a Single, Translated Representation

```
1:  Input
2:  ω: Binary vectors ∀s ∈ c computed by Eq.5
3:  β, δ: Predicate frequency and correlation threshold
4:  DG: The diverse groups for the user-predicates
5:  Output
6:  Exp: The English explanation for each state in the graph G
7:  procedure Generate Explanation
8:      for c in C do
9:          for i in user-predicate do ω(c)[i] ← Eq.6
10:         for i in user-predicate do
11:             if ω(c)[i] == 1 then F ← F ∪ i
12:         end for
13:         for (i, j) in F do
14:             if f_i, f_j ∈ DG then Exp(c) ← Exp(c) ∪ f_i ∪ or ∪ f_j
15:             elif ρ_{i,j} > δ then Exp(c) ← Exp(c) ∪ f_i ∪ and ∪ f_j
16:             else Exp(c) ← Exp(c) ∪ f_i ∪ or ∪ f_j
17:         end for
18:     end for
19:     return Exp
20: end procedure
```

$$\hat{\omega(c)}[i] = \begin{cases} 1, & \text{if } \frac{\sum_{\forall s \in c} \omega(s)[i]}{|c|} > \beta \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

The result, $\omega(c)$, is a binary vector that has 1 at the *ith* predicate if the proportion of each predicate's appearance in all the MDP states, s, in this abstract state, c, is above a threshold β (lines 8-12). Unlike [5], which only presents a predicate as part of the explanation if it appears in all MDP states, our algorithm uses a threshold β to control how frequent a predicate value must appear in the grounded states $\forall s \in c$ to be included in the explanation of the abstract state c. Hence, CAPS guarantees presenting the user with at least some relevant descriptions. We find that in practice, the method of [5] often finds no commonalities between the grounded states, especially as the amount of input data grows (Table.2). We suspect this is because even a single contrastive state can invalidate a predicate following their method.

The second step of Algorithm 1 (lines 13-16) decides which Boolean algebra operators (i.e. or, and) should be used to connect the frequent predicates, so that each node in the graph has a semantically meaningful description. This is done by first requiring the user to group their predicates into *diverse* groups such that no two predicates in the same group can both be true in any grounded state. For example, in a self-driving car, the predicates "car is moving fast" and "car is moving slow" should be in the same group since the car cannot be moving fast and slow simultaneously. If a pair of frequent predicates for an abstract state are part of the same diverse group, we connect those predicates with an "or" operator. Otherwise, we use the Pearson correlation coefficient [20] to find how strong the correlation between the expressive predicates. We use "and" for strong correlations and "or" otherwise. For each pair of predicates

CAPS	Hayes and Shah [5]
"Pole is standing up and cart is either moving left or right and cart is either on the left or in the middle"	No Explanation Produced
"Pole is either stabilizing to the right or standing up and cart is moving right and cart is in the middle"	"Pole is not falling left and cart is moving right and cart is in the middle"
"Pole is either stabilizing to the left or standing up and cart is moving left and cart is in the middle"	"Cart is moving left"

Table 2: Example labels for abstract states produced by CAPS vs. [5] for the Cartpole environment (Appendix A.3 describes this environment and shows its policy graph).

i, j, we compute the Pearson correlation coefficient ρ as:

$$\rho_{i,j} = \frac{cov(i,j)}{std_i std_j} \tag{7}$$

where cov is the sample covariance, std_i and std_j are the sample standard deviations of the predicates i and j in $s, \forall s \in c$. We use a threshold δ to control how correlated two predicates must be in order to justify an "and" relationship between them. Algorithm 1 improves upon [5] by including the largest number of predicates in the explanation as possible, while still presenting combinations of predicates that actually occurred in the input data. Using the diverse groups plays a crucial role, since we can still include two predicates in the explanation which can never coexist in the same state, simply by joining them with an "or" conjunction.

4.5 Highlighting Critical States

Critical states are defined as states in which it is important to take a certain action [6]. To build trust in an end-user, it is helpful to highlight what the agent does in these critical states, as well as what situations the agent believes are critical. For stochastic policies, such as PPO [21], the set of critical states under the policy K_π have been defined as [6]:

$$K_\pi = \{s | \mathcal{H}(\pi(\cdot|s)) < t\} \tag{8}$$

where $\mathcal{H}(\pi(\cdot|s))$ is the entropy of the probability distribution generated by π in state s. Extending this to the setting of abstract states, we define the critical value of an abstract state, $\kappa(c)$, as:

$$\kappa(c) = \frac{1}{|c|} \sum_{s \in c} \mathcal{H}(\pi(\cdot|s)) \tag{9}$$

In this case, a lower critical score, and therefore a lower average entropy, corresponds to a more critical state. We highlight c as a critical abstract state if its critical value is in the bottom n_{th} percentile of the abstract states in its graph. n is arbitrary and only affects the number of critical abstract states which are presented. We chose to use $n = 10$ in our experiments.

5 EXPERIMENTS

5.1 Experimental Settings

We tested CAPS on five environments. Two environments have discrete state spaces, namely Blackjack and Cliffworld [22]. The other three environments - Cartpole, Lunar Lander, and Mountain Car [3] - have continuous state spaces. In addition, Blackjack has a stochastic transition function while the rest of the environments have deterministic transition functions (See the appendix for more details about these environments). To test the generalizability of CAPS, we tested the environments using two different RL agents: one trained with a deterministic algorithm, DQN [16], and one with a stochastic algorithm, PPO [21]. To preserve space, we discuss Mountain Car in detail, and provide policy graphs for the other environments in Appendix A.

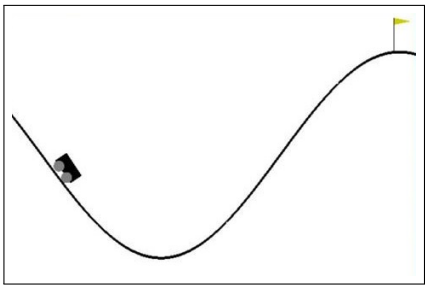


Figure 3: The Mountain Car environment from Open AI Gym

5.2 Results

In the Mountain Car environment, a car starts at the bottom of a valley, and must build up enough speed to reach the top of the hill on the right Figure 3. The car has 3 actions, either accelerate left, accelerate right, or choose to not accelerate. The state space consists of two continuous features, car position, with 0 corresponding to the bottom of the valley, and cart velocity. Figure 4 shows the policy graph for Mountain Car, produced by CAPS. Nodes highlighted in red are selected as critical by CAPS. The user-defined predicates for this environment are broken up into two diverse groups. The first describes the position of the car, and includes the predicates "At the bottom", "On the left slope", "On the right slope", "High up on the left slope", and "High up on the right slope". The second group describes the velocity of the car: "Not moving", "Moving right slowly", "Moving left slowly", "Moving left quickly", and "Moving right quickly".

Given only Figure 3 of the environment and a description of the task, an end-user might suppose the best way to solve the task is to have the car accelerate right until reaching the goal flag. However, such a strategy does not work, because the car fails to build up enough momentum to surmount the hill. The optimal strategy, as discovered by the RL agent, is to first build momentum by alternating its actions, and then to stop accelerating once the

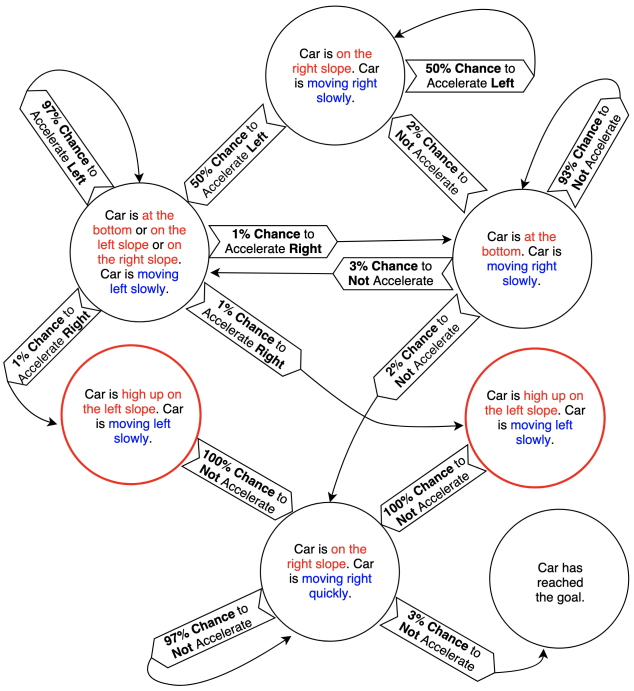


Figure 4: Example of an explanation for Mountain Car produced by CAPS. Abstract states outlined in red are critical.

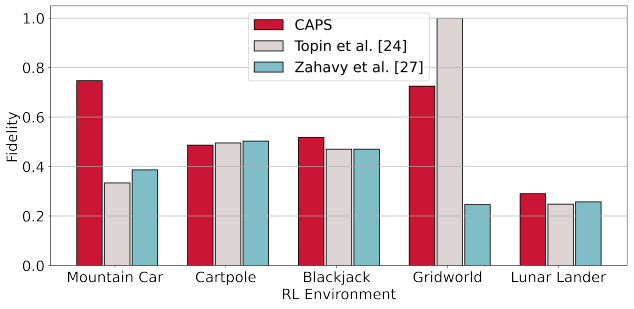


Figure 5: Fidelity for three baselines in the five baseline environments. Data was collected using a PPO policy.

car is high enough up the left hill. Momentum is then enough to carry the car to the goal.

Such a strategy can be seen in the CAPS graph (Figure 4). The car starts at the bottom, and has a stochastic policy which sees it alternate between each action. Once it builds up enough momentum to reach a position high up on the left slope, it identifies its current state as critical, and stops accelerating with high probability. These actions bring the car to a point in which it is rapidly rising up the right slope, and eventually reaches the goal.

Alongside the policy graphs, we perform analysis on the different scores defined in section 4.2.1 with respect to graph size (in nodes). We present the analysis results in Appendix B. We also compare the accuracy of CAPS policy graphs to baselines through a fidelity metric. The *fidelity* of the policy graph is defined as the proportion of actions taken by the trained RL agent that are the same as the

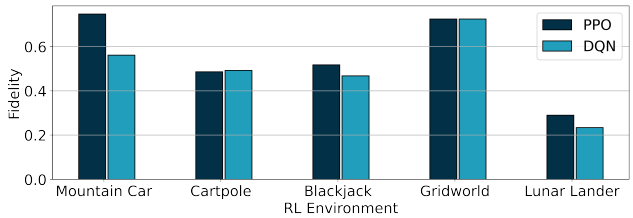


Figure 6: Fidelity in the five baseline environments for two RL agents. One was trained with PPO, and the other with DQN.

action the policy graph says the agent should take [23]. To calculate fidelity for each environment, we simulate about 2000 timesteps of environment interaction. We also average the fidelity of these trials over 10 different generated graphs by each baseline. The results for CAPS and the two baselines which produce policy graphs, [23, 26], are shown in Figure 5. In Figure 5, we see that the fidelity of CAPS is either superior or comparable to [23, 26], except in the case of Gridworld. In the Gridworld environment, the policy is deterministic enough that Topin and Veloso [23], which attempts to memorize (s_t, a_t) pairs, can do so without loss of fidelity. We construct CAPS graphs to favor interpretability and conciseness over perfectly fitting the observed data, but we see that the fidelity accuracy does not suffer under this approach. We also see in Figure 6 that CAPS policy graphs are similar in fidelity when the data was generated with PPO versus with DQN. This indicates that CAPS is invariant to the agent's learning algorithm, and is compatible with deterministic and stochastic policies.

6 USER STUDY

6.1 Design

Distinguished from previous XRL approaches, CAPS includes not only developers and researchers, but also non-technical end-users as its target audience. Therefore, we designed a user-study to quantitatively measure the amount of understanding a CAPS graph provides over two other directly-comparable graph-based policy explanation methods [23, 26] from the perspective of the end-users.

For this study, we present the Mountain Car environment to Amazon Mechanical Turk (AMT) workers, along with a single frame from an episode. Given an explanation graph, we then ask the workers (i) *Q1. Which state represents the current scenario of the environment* and (ii) *Q2. Which action the agent will likely execute next.* We reveal the correct answer to *Q1* at the beginning of *Q2*. The Mountain Car environment is selected because actions of a learned RL agent can be counter-intuitive to a layperson, hence it requires an explanation graph for correct interpretation. In fact, *without* an explanation graph, AMT workers are only able to achieve an overall accuracy of 3% on *Q2* (Table 3). We then manually select three frames (i.e., scenarios) from an episode, and test this same set of frames across *all* baseline (Appendix C). The selected frames represent a sequence of three distinctive locations of the car of the environment (i.e., on the left slope, at the bottom and on the right slope). We show workers either a graph from CAPS, a graph from [23] (*Topin and Veloso*), a graph from [26] (*Zahavy et al.*), or no graph, as a baseline. Note that the two baseline graphs do not

Method	#	Top 20 Longest Responses					Top 30 Longest Responses					All Responses				
	Ans.	Time↓	S↑	A↑	S+A↑	S→A↑	Time↓	S↑	A↑	S+A↑	S→A↑	Time↓	S↑	A↑	S+A↑	S→A↑
Without graph	33	41s	-	0.05	-	-	38s	-	0.03	-	-	37s	-	0.03	-	-
Topin and Veloso [23]	39	159s	0.05	0.40	0.05	1.0 (1/1)	141s	0.03	0.33	0.03	1.0 (1/1)	130s	0.03	0.28	0.03	1.0 (1/1)
Zahavy et al. [26]	45	136s	0.45	0.45	0.10	0.22 (2/9)	125s	0.5	0.47	0.13	0.27 (4/15)	115s	0.42	0.47	0.16	0.37 (7/19)
CAPS (Optimal)	46	139s	0.85	0.95	0.80	0.94 (16/17)	126s	0.77	0.86	0.70	0.91 (21/23)	114s	0.78	0.87	0.70	0.89 (32/36)

Table 3: Comparison of user-study results on (*Time*)–the total time spent on the task, accuracy of selecting the (*S*)–correct abstract state, (*A*)–correct action, (*S+A*)–correct abstract state and action, (*S→A*)–correct action *after* correctly selecting abstract state.

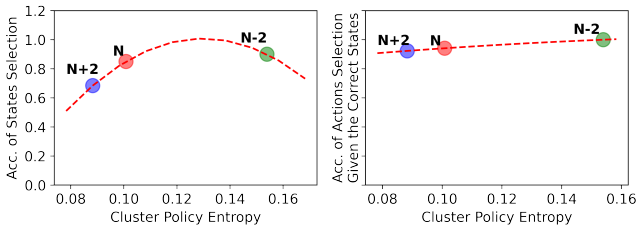


Figure 7: Relationship between comprehensibility and cluster policy entropy with varied CAPS graph sizes.

contain NL explanations. We show each graph used in this study in Appendix C.

We recruited a total of 120 AMT workers who are English speakers located in North America. We recruited general users and *make no assumption* regarding the their knowledge and experience in ML or RL. To ensure the quality of the workers and reduce possible biases, we employ various recruitment criteria and experiment controls and maintain these across all baselines. We refer the readers to Appendix C for more details on the user-study (recruitment criteria, payment, etc.). Our user-study is IRB-approved.

6.2 Results

Table 3 summarizes the results. First, all explanation methods help the end-users to better understand the learned RL agent. Moreover, it is *statistically significant* (*p-value*≤ 0.01) that CAPS performs better than all baselines on average (Table 4). In fact, our method is the most comprehensible, enabling the end-users to accurately interpret *both* the current states and the agent's next actions (S+A) up to 70% accuracy, a four times improvement from the next best baseline (Zahavy et al.) (Table 3). Interestingly, except for the *Topin and Veloso* graph, the more time the AMT workers spent on the tasks, the better their responses became. Particularly, the top 20 longest responses recorded an accuracy of over 85% and 90% accuracy on *Q1* and *Q2*, respectively, when provided with CAPS graph (Table 3). Furthermore, adding either NL texts (CAPS) or visual illustrations (Zahavy et al.[26]) for each abstract state enables the end-users to make decisions more promptly and effectively (Table 3).

Given the assumption that the end-users are involved in the *abstract state translation* step of CAPS (Sec. 4.4.1), they should be fully aware of the mapping from a given scenario of the car to its respective abstract state on the CAPS graph. Table 3 shows that those who fit into this assumption–i.e., users who selected the correct abstract state in *Q1*, are the most accurate in interpreting the agent's next actions, with an overall accuracy of around 90% (S→A). Even though this accuracy is 100% in case of *Topin and Veloso*, there was only 1 response that correctly answered *Q1* (Table 3).

CAPS	>NoGraph	>Topin and Veloso	>Zahavy et al.
p-value (S)	-	1.5e-17	1.6e-4
p-value (A)	2.3e-21	7.8e-10	1.1e-5
p-value (S+A)	-	2.4e-13	1.1e-8

Table 4: *p-values* (all are ≤0.01) of hypothesis tests on whether CAPS can provide the AMT workers with additional explanatory values to accurately select *Q1* (S), *Q2* (A) or both (S+A)

We also evaluate the trade-off between explainability of a generated CAPS graph, its size, and its cluster policy entropy (Section 4.2.1). Figure 7 (Left) shows that the optimal graph size found by CAPS is best positioned in terms of both comprehensibility (the higher the better) and the policy entropy (the lower the better). Once an abstract state is correctly identified, it then becomes relatively easy (≥90% accuracy) for the end-users to correctly identify the agent's next action (Figure 7, Right).

7 CONCLUSION

We introduce a novel method, named as CAPS, for generating comprehensible policy graphs, which can explain the policy of an RL agent to an end-user with minimal knowledge of machine learning. We present a state abstraction strategy that gives us control over the abstraction and size of the policy graph, and allows us to gain additional information by observing how the graph changes as we make it more or less abstract. We also propose a novel method for condensing entire abstract states into concise, natural language descriptions.

ACKNOWLEDGEMENT

The work was in part supported by NSF awards #1950491, #1909702, and #2105007.

REFERENCES

[1] Dan Amir and Ofra Amir. 2018. HIGHLIGHTS: Summarizing Agent Behavior to People. In *AAMAS*.

[2] Hamid Benbrahim and Judy A. Franklin. 1997. Biped dynamic walking using reinforcement learning. *Robotics Auton. Syst.* 22 (1997), 283–302.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. arXiv:arXiv:1606.01540

[4] Samuel Greydanus, Anurag Koul, Jonathan Dodge, and Alan Fern. 2018. Visualizing and Understanding Atari Agents. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, 1792–1801.

[5] Bradley Hayes and J. Shah. 2017. Improving Robot Controller Transparency Through Autonomous Policy Explanation. *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI* (2017), 303–312.

[6] Sandy H. Huang, Kush Bhatia, Pieter Abbeel, and Anca D. Dragan. 2018. Establishing Appropriate Trust via Critical States. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 3929–3936. https://doi.org/10.1109/IROS.2018.8593649

[7] Rahul Radhakrishnan Iyer, Yuezhang Li, Huao Li, Michael Lewis, Ramitha Sundar, and Katia P. Sycara. 2018. Transparency and Explanation in Deep Reinforcement Learning Neural Networks. *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society* (2018).

[8] Zoe Juozapaitis, Anurag Koul, Alan Fern, M. Erwig, and Finale Doshi-Velez. 2019. Explainable Reinforcement Learning via Reward Decomposition.

[9] Bing Liu, Yiyuan Xia, and Philip S. Yu. 2004. Clustering Via Decision Tree Construction.

[10] Guiliang Liu, Oliver Schulte, Wang Zhu, and Qingcan Li. 2018. Toward Interpretable Deep Reinforcement Learning with Linear Model U-Trees. arXiv:1807.05887 [cs.LG]

[11] Ning Liu, Zhe Li, Jielong Xu, Zhiyuan Xu, Sheng Lin, Qinru Qiu, Jian Tang, and Yetang Wang. 2017. A Hierarchical Framework of Cloud Resource Allocation and Power Management Using Deep Reinforcement Learning. 372–382. https://doi.org/10.1109/ICDCS.2017.123

[12] Prashan Madumal, Tim Miller, Liz Sonenberg, and Frank Vetere. 2020. Explainable Reinforcement Learning through a Causal Lens. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2493–2500. https://aaai.org/ojs/index.php/AAAI/article/view/5631

[13] Aniek Markus, Jan Kors, and Peter Rijnbeek. 2021. The role of explainability in creating trustworthy artificial intelligence for health care: A comprehensive survey of the terminology, design choices, and evaluation strategies. *Journal of Biomedical Informatics* 113 (Jan 2021), 103655. https://doi.org/10.1016/j.jbi.2020.103655

[14] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Anand Babu, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. 2020. Chip Placement with Deep Reinforcement Learning. arXiv:2004.10746 [cs.LG]

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602 [cs.LG]

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602 [cs.LG]

[17] A. Moore. 1995. Variable Resolution Reinforcement Learning.

[18] Matthew L. Olson, Roli Khanna, Lawrence Neal, Fuxin Li, and Weng-Keen Wong. 2021. Counterfactual state explanations for reinforcement learning agents via generative deep learning. *Artificial Intelligence* 295 (2021), 103455. https://doi.org/10.1016/j.artint.2021.103455

[19] Jan Peters and Stefan Schaal. 2006. Policy Gradient Methods for Robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2219–2225. https://doi.org/10.1109/IROS.2006.282564

[20] Joe Rodgers and Alan Nicewander. 1988. Thirteen Ways to Look at the Correlation Coefficient. *American Statistician - AMER STATIST* 42 (02 1988), 59–66. https://doi.org/10.1080/00031305.1988.10475524

[21] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG]

[22] Richard S.Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence* 112 (1999), 181–211.

[23] Nicholay Topin and Manuela Veloso. 2019. Generation of Policy-Level Explanations for Reinforcement Learning. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2514–2521. https://aaai.org/ojs/index.php/AAAI/article/view/4097

[24] Jasper van der Waa, Jurriaan van Diggelen, Karel van den Bosch, and Mark Neerincx. 2018. Contrastive Explanations for Reinforcement Learning in terms of Expected Consequences. arXiv:1807.08706 [cs.LG]

[25] Zhao Yang, Song Bai, Li Zhang, and Philip H. S. Torr. 2019. Learn to Interpret Atari Agents. arXiv:1812.11276 [cs.LG]

[26] Tom Zahavy, Nir Ben-Zrihem, and Shie Mannor. 2016. Graying the black box: Understanding DQNs. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 48)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.). PMLR, New York, New York, USA, 1899–1908. https://proceedings.mlr.press/v48/zahavy16.html

CAPS: Comprehensible Abstract Policy Summaries for Explaining Reinforcement Learning Agents

Supplementary Material

A Experimental Environments

We here describe five environments tested and show their corresponding policy graphs generated by CAPS.

A.1 Mountain Car

In the Mountain Car environment, a car starts at the bottom of a valley, and must build up enough speed to reach the top of the hill on the right Figure 1. The car has 3 actions, either accelerate left, accelerate right, or choose to not accelerate. The state space consists of two continuous features, car position, with 0 corresponding to the bottom of the valley, and cart velocity. Figure 2 shows the policy graph for Mountain Car, produced by CAPS. Nodes highlighted in red are selected as critical by CAPS.

The user-defined predicates for this environment are broken up into two diverse groups. The first describes the position of the car, and includes the predicates "At the bottom", "On the left slope", "On the right slope", "High up on the left slope", and "High up on the right slope". The second group describes the velocity of the car: "Not moving", "Moving right slowly", "Moving left slowly", "Moving left quickly", and "Moving right quickly".

Given only Figure 1 of the environment and a description of the task, an end-user might suppose the best way to solve the task is to have the car accelerate right until reaching the goal flag. However, such a strategy does not work, because the car fails to build up enough momentum to surmount the hill. The optimal strategy, as discovered by the RL agent, is to first alternate between left acceleration, right acceleration, and no acceleration to build up momentum, and then to stop accelerating once the car is high enough up the left hill. Momentum is then enough to carry the car to the goal.

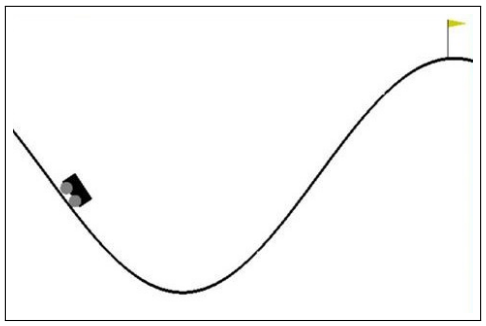


Figure 1: The Mountain Car environment from Open AI Gym

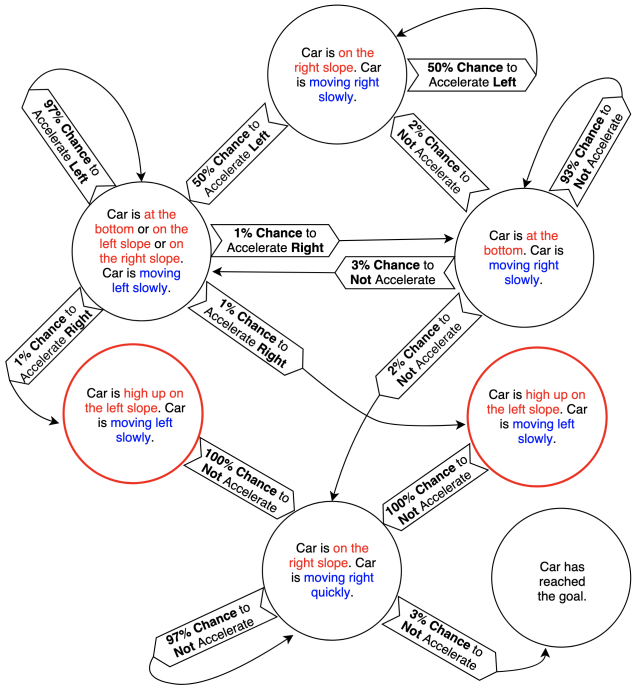


Figure 2: The policy graph for Mountain Car produced by CAPS. Abstract states outlined in red are critical.

Such a strategy can be seen in the CAPS graph (Figure 2). The car starts at the bottom, and has a stochastic policy which sees it alternate between accelerating left, right, and not at all. Once it builds up enough momentum to reach a position high up on the left slope, it identifies its current state as critical, and stops accelerating with high probability. These actions bring the car to a point in which it is rapidly rising up the right slope, and eventually reaches the goal.

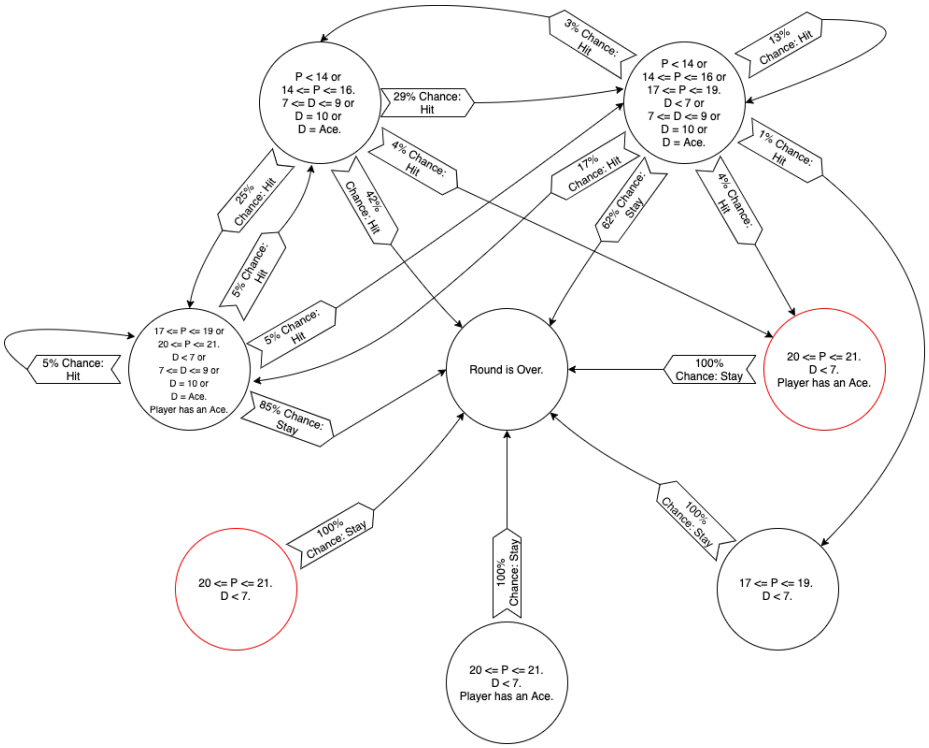


Figure 3: The CAPS explanation for Blackjack [1]

A.2 Blackjack

Blackjack is a popular card game with a discrete state space that has been solved using RL. The objective of the game is to win money by obtaining a point total higher than the dealer's without exceeding 21. Determining an optimal blackjack strategy proves to be a difficult challenge due to the stochastic nature of the game.

The game works by assigning each card a point value. Cards 2 through 10 are worth their face value, while Jacks, Queens, and Kings are worth 10 points. An ace is worth either 1 or 11 points, whichever is the most beneficial. This game is placed with an infinite deck (or with replacement). The game starts with dealer having one face up and one face down card, while player having two face up cards. The player can request additional cards (hit=1) until they decide to stop (stick=0) or exceed 21 (bust). Hence, the possible actions include hitting,standing, splitting, or doubling down. After the player sticks, the dealer reveals their face-down card, and draws until their sum is 17 or greater. If the dealer goes bust the player wins. If neither player nor dealer busts, the outcome (win, lose, draw) is decided by whose sum is closer to 21. The reward for winning is +1, drawing is 0, and losing is -1. The observation of a 3-tuple of: the players

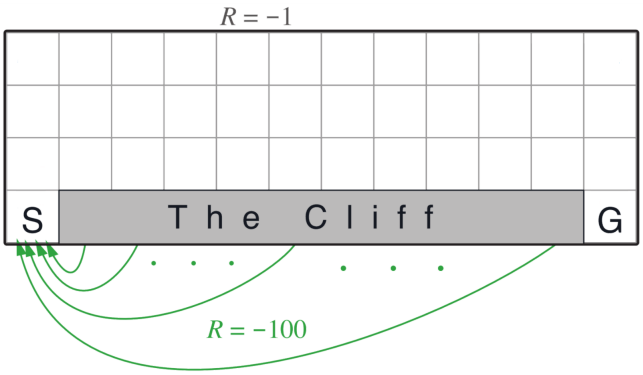


Figure 4: The state space of Cliffworld environment [1]

current sum, the dealer's one showing card (1-10 where 1 is ace), and whether or not the player holds a usable ace (0 or 1). This environment corresponds to the version of the blackjack problem described in Example 5.1 in [1]. The Open AI Gym design of this game is here: `https://github.com/openai/gym/blob/master/gym/envs/toy_text/blackjack.py`

The CAPS graph for Blackjack (Figure 3) can be interpreted as follows. P represents the current sum of the values of the player's cards. D represents the value of the single card the dealer is showing. If the player or dealer has a usable ace, that is displayed as "Player has an Ace" or "D = Ace", respectively. The player has two possible actions, hit or stick. From the top left node, we see that if the player's hand is between 0 and 16, it will choose to hit every time. There are many edges leading out from this node because the environment has a stochastic transition function: choosing to hit from this node does not always transition the environment to the same next node. There are two other nodes where the player's hand is less than 19, but the dealer's hand is good (7-10 or ace). In these nodes, the player will often choose to stay, but sometimes will hit. In all other nodes, the player has a good hand and the dealer does not. The player will stay in these scenarios, with two of them being labeled critical.

A.3 Cliffworld

Cliffworld is from example 6.5 in [1]. It is a standard gridworld with four action in each state (up, down, right, left) which deterministically cause the corresponding state transitions (Figure 4). The reward is -1 for all transitions until the terminal state is reached but if the agent falls from the Cliff then the reward -100. The terminal state is in the bottom right corner, and the starting state is the bottom left corner. The implementation of this environment in Open AI Gym is in `https://github.com/podondra/gym-gridworlds/blob/master/gym_gridworlds/envs/windy_gridworld_env.py`

The CAPS Explanation for Cliffworld can be interpreted as follows. The agent starts as the bottom left cell. It starts by moving up, and then moves

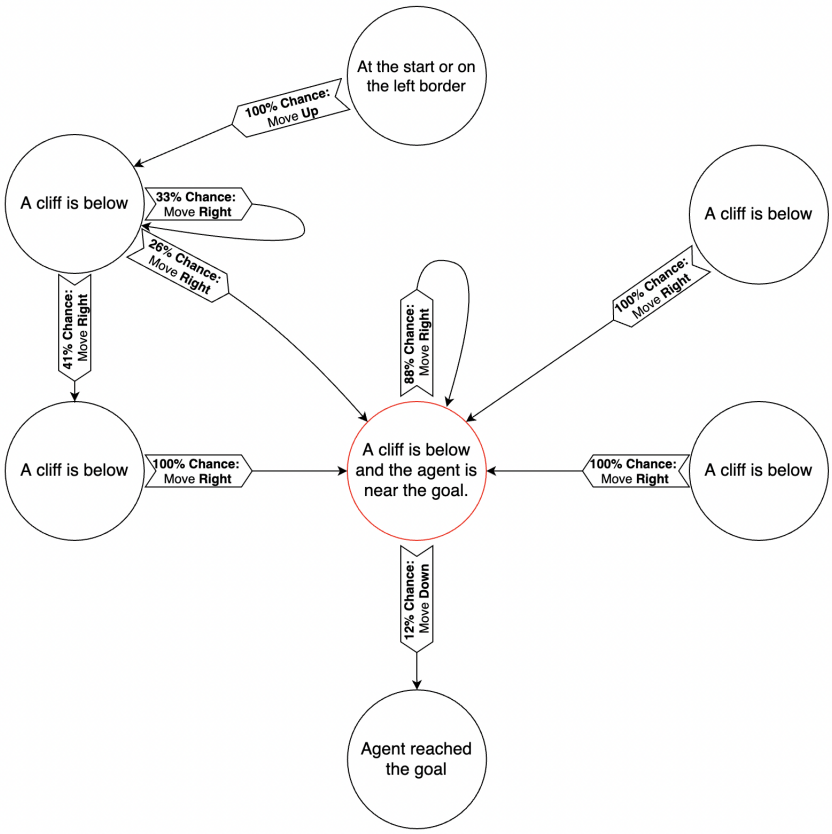


Figure 5: The CAPS Explanation for Cliffworld.

right while the cliff is directly below it. It keeps moving right until it is close to the goal (the cell above the goal), then moves down.

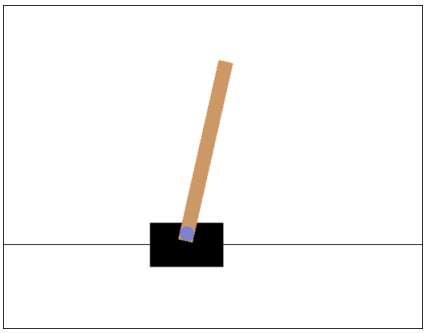


Figure 6: The Cartpole environment

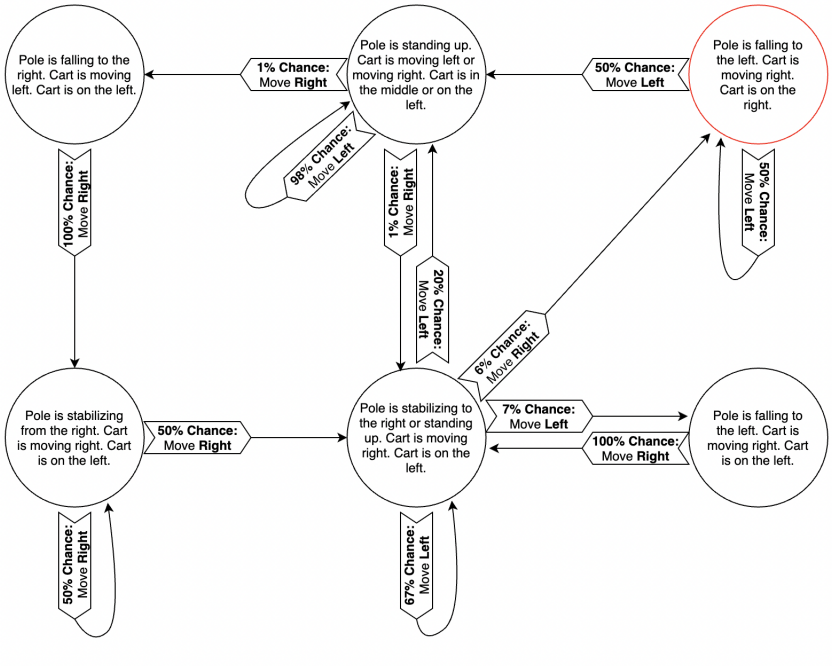


Figure 7: The CAPS Explanation for Cartpole.

A.4 Cartpole

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track (Figure 6). The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical,

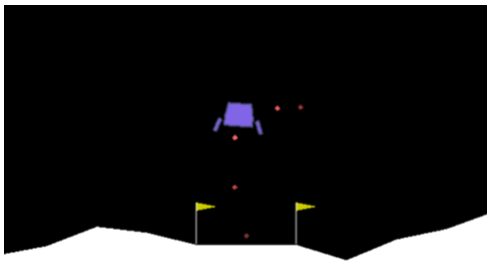


Figure 8: The Lunar Lander environment.

or the cart moves more than 2.4 units from the center. The actions are push cart to the left or push cart to the right. The observation includes; the cart position, velocity, pole angle, and angular velocity. `https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py`

The CAPS explanation for Cartpole can be briefly summarized as follows. The two center cells, which contain states where the pole is standing up, are "stable" states, in the sense that if the agent ends up in a different abstract state, it always returns back to one of these two abstract states. With low probability, the agent can transition from one of these stable states to one of the other states, which contain situations in which the pole is falling to the left or right. One of these states has been labeled as critical. In each of these states, the agent will take the same action (either right or left), until it transitions back to a stable state. In addition, we see that there is no cell where the pole has fallen over, indicating that the agent did not fail the task in the collected timesteps of experience.

A.5 Lunar Lander

This game simulates the situation where a lander needs to land at a specific location under low-gravity conditions, and has a well-defined physics engine implemented. The main goal of the game is to direct the agent to the landing pad as softly and fuel-efficiently as possible. The state space is continuous as in real physics, but the action space is discrete. The observation state has; coordinate of the lander, the horizontal velocity, the vertical velocity, the orientation in space, the angular velocity,the left leg touching the ground (Boolean), the right leg touching the ground (Boolean). The action space includes; do nothing, fire left orientation engine, fire right orientation engine, and fire main engine. `https://github.com/openai/gym/blob/master/gym/envs/box2d/lunar_lander.py`

The CAPS explanation for Lunar Lander can be interpreted as follows. The agent starts on top of and higher than the goal, and activates its boosters so that it moves slightly up, presumably so that it does not hit the ground moving too fast. It identifies these states as critical. This causes it to move right slightly

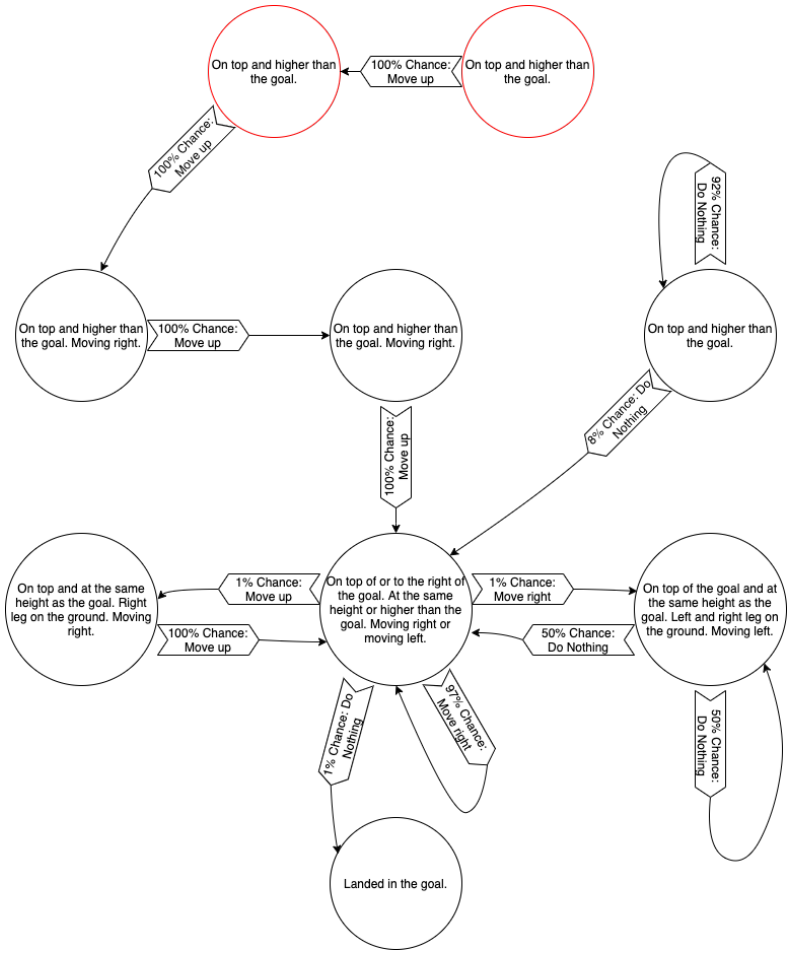


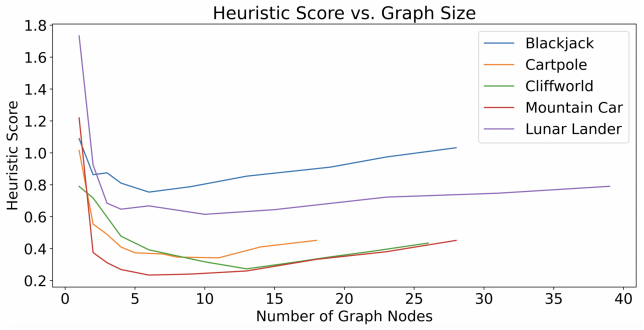
Figure 9: The CAPS Explanation for Lunar Lander

because of orientation, falling until it reaches a state where it is at the same height as the goal. From this state, it eventually transitions to states where its left or right leg, or both, are on the ground, and it either moves up or right to re-position itself into the goal zone, or does nothing until it has landed in the goal.

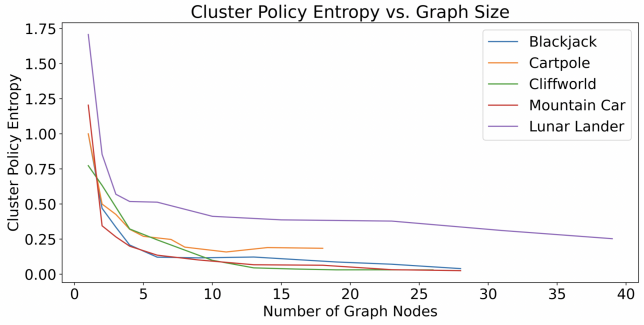
B Heuristic Optimization: Analysis Study

We see in figure 10 that the *value score* decreases as graph size increases, indicating that larger graphs can represent the transition function of the environment more effectively. Also, we see that the entropy of the *cluster policy* decreases as graph size increases. This is natural since we cluster the data in part based off

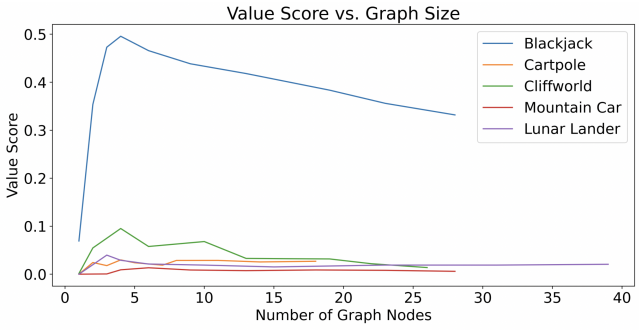
of the taken action.



(a) Heuristic Score vs. Size of CAPS Graph



(b) Cluster Policy Entropy vs. Size of CAPS Graph



(c) Value Score (Eq.2) vs. Size of CAPS Graph

Figure 10: Plots of the different heuristics used in Eq.4 versus the number of nodes in a CAPS graph.

In figure 10c, we plot the relationship between our score heuristic and the graph size. We observe an inflection point where the increase in accuracy is no longer worth the extra nodes on the graph. That point is returned as the optimal graph size.

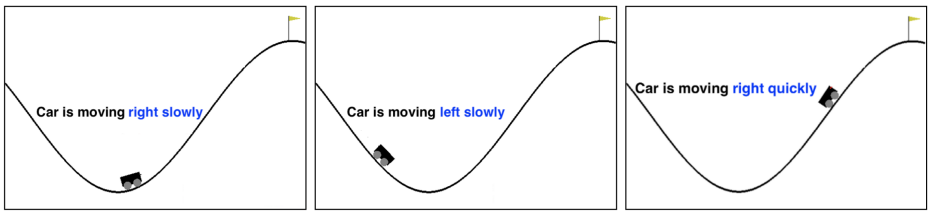


Figure 11: Three selected distinctive frames of an episode of the *Mountain Car* RL environment that were used for the user-study. The AMT workers are presented with a single scenario–i.e., frame, at a time.

C User-Study

C.1 Recruitment and Experimental Controls

To ensure the quality of the workers, we only recruit those who have (i) HIT approval rate $\geq$ 98%, (ii) have number of HITs approved $\geq$ 1000, and (iii) are classified as "master" workers by AMT–i.e., who demonstrated excellent performance for a wide range of tasks in the past. Moreover, we also utilized a count-down timer to ensure a minimum attention span period of 60 and 30 seconds for *Q1* and *Q2*, respectively. To further motivate the workers to well examine the given tasks, we instructed and gave each worker an additional +50% of the original payment amount for each correct response to either Q1 or Q2, or double payment amount if they correctly answer both questions. *Excluding* the potential bonus, we pay each worker \$10–\$12/hour (v.s. 2021 federal minimum wage of \$7.25), which corresponds to a minimum commitment of 1.5 minutes working time, up to 3 minutes for both questions. To remove possible bias, we also ensure that the pools of workers across the tasks of different baselines are not overlapped. We maintained the same recruitment criteria and experiment controls as described across all baselines. Our user-study was approved by the Institutional Review Board.

C.2 User-Study Interface

Figure 11 shows the three selected frames for the user-study with distinctive current locations (right, left, bottom) and behaviors (moving slowly, quickly) of the car in the Mountain Car environment. In Figure 12, given a scenario, AMT workers are asked to identify the abstract state from an explanation graph generated by CAPS. In Figure 13, given a scenario, AMT workers are asked to identify the next probable action of the agent from an explanation graph generated by CAPS. Figures 14 and 15 show the explanation graphs generated by *Topin et al.* [2] and *Zahavy et al* [3], respectively, for the Mountain Car environment that was used in the user-study.

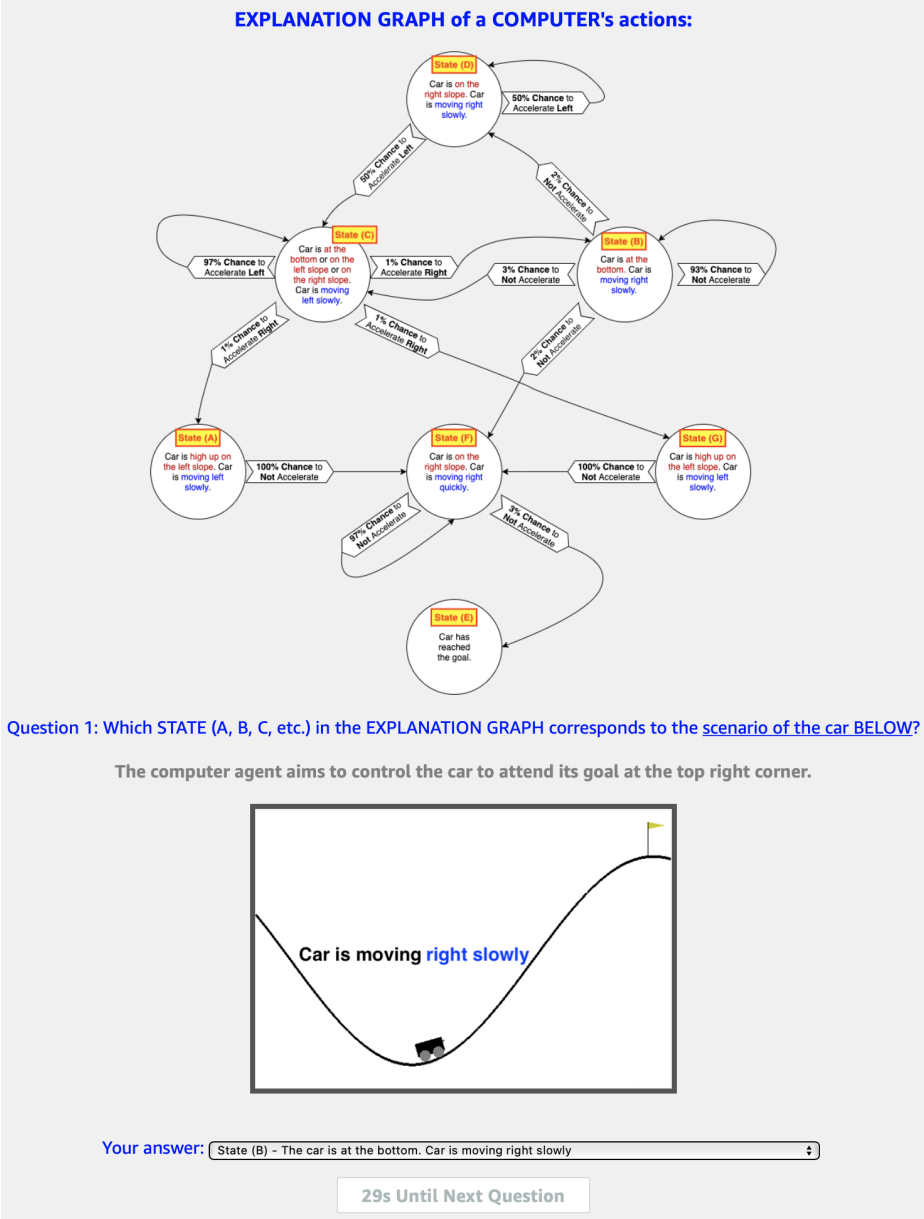


Figure 12: First question of the user-study interface on AMT on the *Mountain Car* RL environment. The worker is asked to identify the abstract state (in circle shape) from the explanation graph that corresponds with the current scenario of car. A count-down timer is utilized to ensure a *minimum* attention span from the worker before she or he moves to the next question.

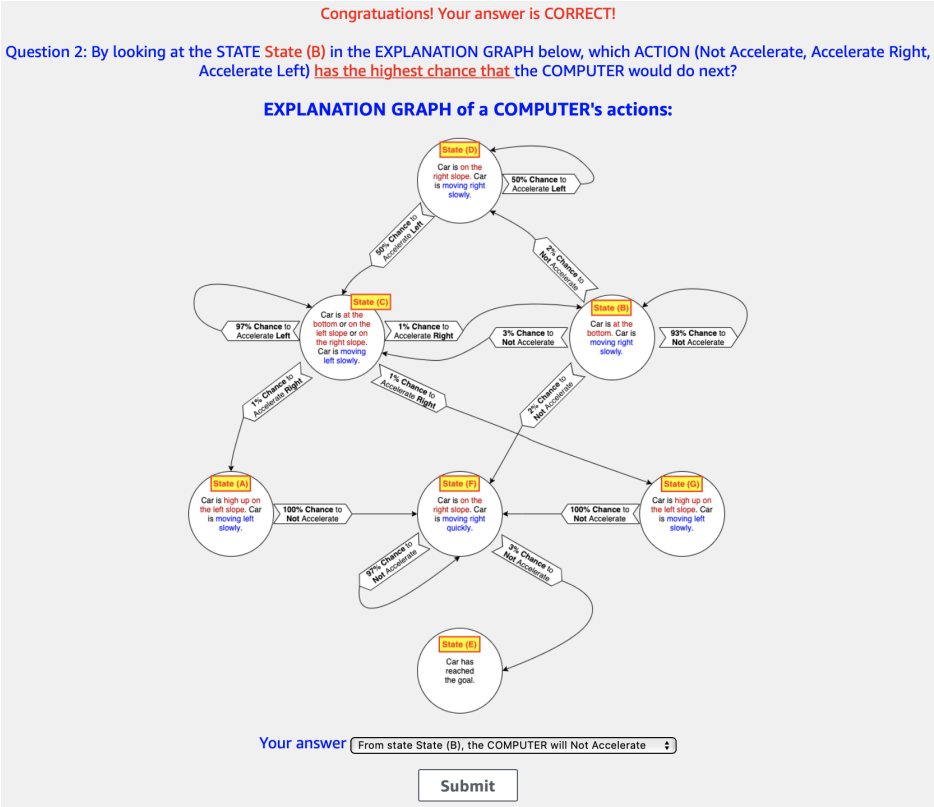


Figure 13: Second question of the user-study interface on AMT on the *Mountain Car* RL environment. The worker is asked to identify the next probable action that the RL agent would carry out given the current scenario of the car. If the worker incorrectly answers the first question (Figure 12), she/he will be informed with the correct abstract state of the explanation graph before answering this question.

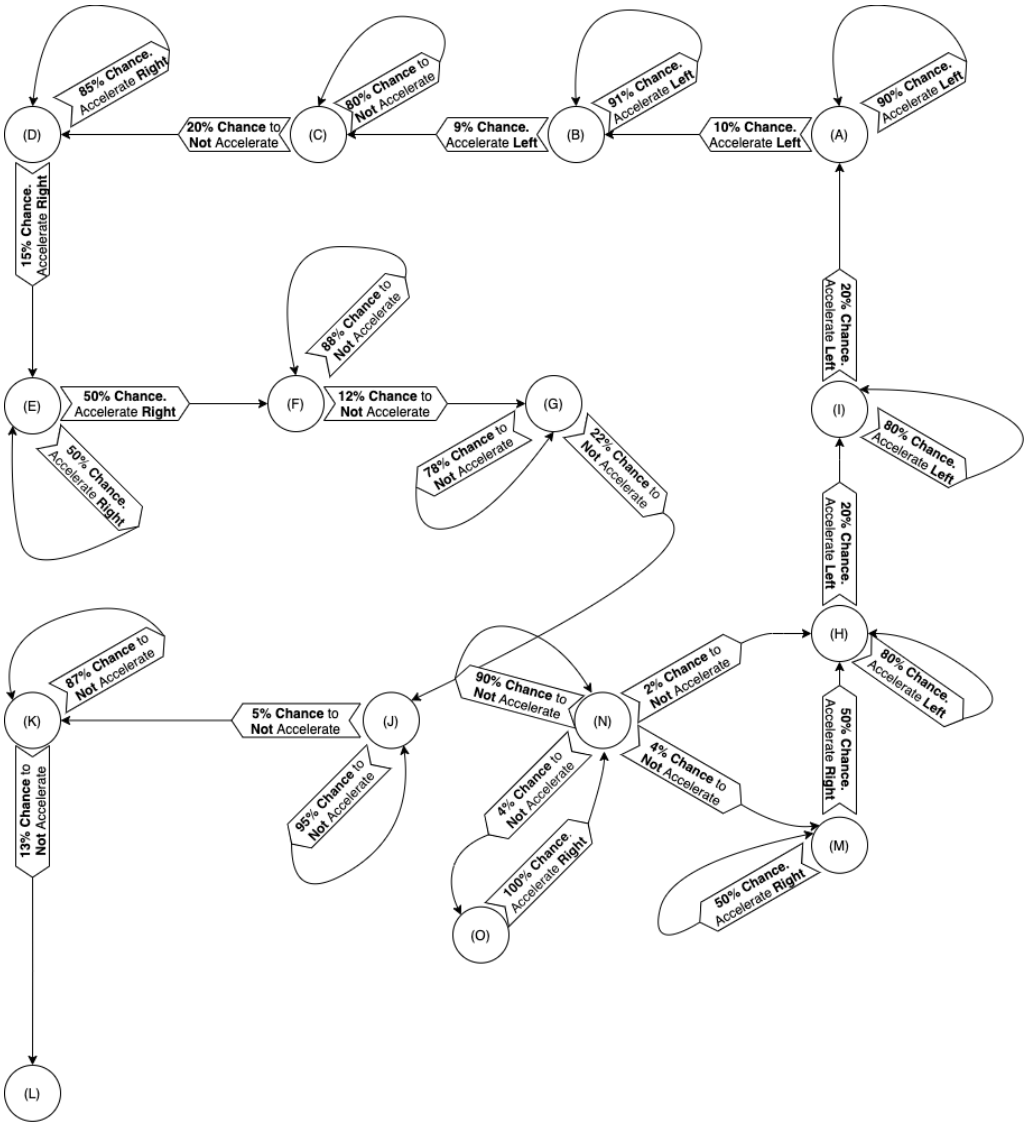


Figure 14: Explanation graph generated by *Topin et al.* [2] for the Mountain Car environment that was used in the user-study.

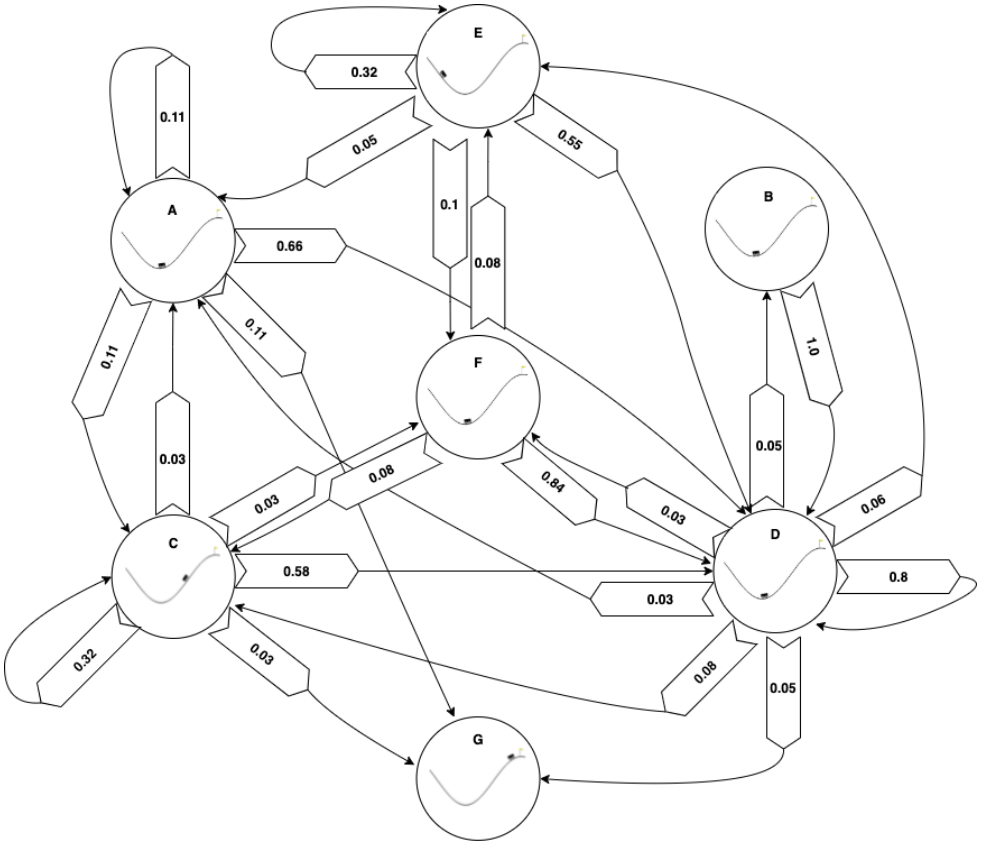


Figure 15: Explanation graph generated by *Zahavy et al.* [3] for the Mountain Car environment that was used in the user-study.

References

[1] R. S.Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, pp. 181–211, 1999.

[2] N. Topin and M. Veloso, "Generation of policy-level explanations for reinforcement learning," 2019.

[3] T. Zahavy, N. B. Zrihem, and S. Mannor, "Graying the black box: Understanding dqns," 2017.