The Knowledge Engineering Review, Vol. 00:0, 1–999. © 2004, Cambridge University Press
DOI: 10.1017/S000000000000000 Printed in the United Kingdom

Leveraging human knowledge in tabular reinforcement learning: A study of human subjects

Ariel Rosenfeld[1], Moshe Cohen[2], Matthew E. Taylor[3], and Sarit Kraus[2]

[1] *Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel.*
E-mail: arielros1@gmail.com
[2] *Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel.*
[3] *Department of Computer Science, Washington State University, Pullman, Washington, USA.*

Abstract

Reinforcement Learning (RL) can be extremely effective in solving complex, real-world problems. However, injecting human knowledge into an RL agent may require extensive effort and expertise on the human designer's part. To date, human factors are generally not considered in the development and evaluation of possible RL approaches. In this article we set to investigate how different methods for injecting human knowledge are applied, *in practice*, by human designers of varying levels of knowledge and skill. We perform the first empirical evaluation of several methods, including a newly proposed method named SASS which is based on the notion of similarities in the agent's state-action space. Through this human study, consisted of 51 human participants, we shed new light on the human factors that play a key role in RL. We find that reward shaping seems to be the most natural method for most designers, both expert and non-expert, to speed up RL. However, we further find that the our proposed method SASS can be effectively and efficiently combined with basic reward shaping and provides a natural alternative to using only a single speedup method with minimal human designer effort overhead.

1 Introduction

Reinforcement Learning Sutton and Barto (1998) (RL) has had many successes solving complex, real-world problems. However, unlike supervised machine learning, there is no standard framework for non-experts to easily try out different methods (e.g., Weka Witten et al. (2016)) which may pose a barrier to wider adoption of RL methods. While many frameworks exist, such as RL-Glue[1], RLPy[2], PyBrain[3], AIGym[4] and others, they all assume some, and sometimes substantial, amount of RL knowledge and therefore substantial effort is required to add new tasks or instantiate different techniques within these frameworks. Another barrier to wider adoption of RL methods, which is in the focus of this article, is the fact that injecting human knowledge, which can significantly improve the speed of learning, can be difficult for a human designer.

When designing an RL agent, human designers must face the question about how much human knowledge to inject to the system and more importantly, which approach to use to inject the desired knowledge. From the AI research perspective, the more that can be learned autonomously, the more interesting and beneficial the agent. From the engineering or more practical perspective, more human knowledge is desirable as it can help improve the agent's learning. However, this knowledge is only useful as long as it is practical to gather and leverage by the human designer. In order for RL methods to move

[1] `http://glue.rl-community.org`
[2] `https://rlpy.readthedocs.io/`
[3] `http://pybrain.org/`
[4] `https://gym.openai.com/`

beyond requiring developers to fully understand the "black arts" of generalization, approximation, and biasing, it is critical that the community better understand if and how both expert and non-expert humans can provide useful information. This article, rather than requiring the authors to provide such knowledge, takes the problem to field and focuses on human designers who have background in AI and coding, but varying experience in RL.

There are many approaches for leveraging human knowledge in an RL learner. Perhaps the most prominent one is function approximation, Busoniu et al. (2010) (FA), which allows a designer to inject her knowledge by abstracting the domain appropriately and thereby allowing the agent to generalize its experience quickly. Many successful RL applications have used highly engineered state features to bring about successful learning performance (e.g., 'the distance between the simulated robot soccer player with the ball to its closest opponent' and 'the minimal angle with the vertex at the simulated robot soccer player with the ball between the closest teammate and any of the opponents', Stone et al. (2006)). Another popular method is the use of Reward Shaping (RS) Mataric (1994)To do: Talk About RS!!!!@ which its core idea is to allow the RL learner to exploit the designer knowledge during the learning process, by provide it with a continuous feedback for its actions. With that method, the designer modify the reward signal so it will guide the learner to prefer the actions that seems better, to the designer opinion. Formally, the domain's reward signal received from being in state s, taking action a and moving to state s' is denoted by $R(s, a, s')$; the shaping value for that transition by $F(s, a, s')$, so the total reward the learner receives for the transition from s to s' by action a is $R(s, a, s') + F(s, a, s')$. However, incautious use of this method could easily lead to sub-optimal learned policy, so the most common approach is to use the Potential Based Reward Shaping, as presented at Ng et al. (1999). **MOSHE: done**

While different methods of leveraging human knowledge in RL learners have been thoroughly investigated with respect to their theoretical properties and empirical performance in various settings, their deployment often requires extensive engineering and expertise on the designers' part, which is generally not considered (e.g., methods are not evaluated in terms of the amount of time a developer must invest to fine-tune parameters, select appropriate state representations, etc. and how different developers are able to effectively use the different approaches).

The baseline approach in this study is to allow no generalization: an agent's interactions with its environment will immediately affect only its current state in a tabular representation. We will compare this baseline with two widely common speedup approaches: FA and RS. We further propose and evaluate a novel approach which we name SASS, standing for State Action Similarity Solutions, which relies on handcoded state-action similarities. The SASS approach is based on the well-established psychological theory of constructivism Bruner (1957), by which a technically-able designer can define and refine both complex and simplistic constructs (or *similarities* in our terminology) in the state-action space, according to the designer's abilities, knowledge and beliefs, using handcoded similarities to simplify the RL agent's task. To avoid many of the possible confounding factors, we consider a simple temporal difference RL algorithm: Q-learning Watkins (1989) with a tabular representation.

We test our SASS approach, along with the baseline and FA and RS, in this first-of-its-kind human study consisting of three experts (highly experienced programmers with an RL background, but not co-authors in this paper) and 48 To do: howmany? **MOSHE: done** non-expert computer science students. To that end, three RL tasks of varying complexities are considered:

1. The "toy" task of simple robotic soccer To do: cite origin paper, providing a basic setting for the evaluation of the proposed SASS approach.
2. A large grid-world task named Pursuit To do: cite origin paper, investigating the three methods on a moderately challenging task.
3. The popular game of Mario To do: cite origin paper, exemplifying the complexities of instantiating speedup approaches in complex tasks. **ARIEL: Is this a correct way to characterize Mario and Pursuit?** **MOSHE: seems ok to me**

This article argues that in order to bring about a wider adoption of RL techniques, specifically of generalization techniques, it is essential to both investigate and develop RL techniques appropriate for

both expert and non-expert designers. We hope that this study will encourage other researchers to invest increased effort in the human factors behind RL and investigate their RL solutions in human studies.

The remainder of the article is organized as follows: In Section 2 we review some preliminaries on RL and survey recent related work. In Section 3 we present the QS-learning algorithm that incorporates similarities within the basic Q-learning framework and discuss its theoretical foundations. We further propose three notions of state-action similarities and discuss how these similarities can be defined. In Section 4 we present an extensive empirical evaluation of the three tested approaches in three RL tasks. Finally, in Section 5 we provide a summary and list future directions for this line of work.

2 Preliminaries and Background

An RL agent generally learns how to interact with an unfamiliar environment Sutton and Barto (1998). We define the RL task using the standard notation of a Markov Decision Process (MDP). An MDP is defined as $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$ where:

- $\mathcal{S}$ is the state-space;
- $\mathcal{A}$ is the action-space;
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ defines the transition function where $\mathcal{T}(s, a, s')$ is the probability of making a transition from state s to state s' using action a;
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function;
- $\gamma \in [0, 1]$ is the discount factor, which represents the significance of future rewards compared to present rewards.

We assume $\mathcal{T}$ and $\mathcal{R}$ are initially unknown to the agent. In discrete time tasks, the agent interacts in a sequence of time steps. On each time step, the agent observes its state $s \in \mathcal{S}$ and is required to select an action $a \in \mathcal{A}$. The agent then receives a reward r according to the unknown $\mathcal{R}$ function and arrives at a new state s' according to the unknown $\mathcal{T}$ function. We define an agent's *experience* as a tuple $\langle s, a, r, s' \rangle$ where action a is taken in state s, resulting in reward r and a transition to next state s'. The agent's objective is to maximize the accumulated discounted rewards throughout its lifetime. Namely, the agent seeks to find a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ that maximizes the expected total discounted reward (i.e., expected return) from following it.

Temporal difference RL algorithms such as Q-learning Watkins (1989) approximate an action-value function $Q : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{R}$, mapping state-action pairs to the expected real-valued discounted return. Q-learning updates the Q-value estimation according to the temporal difference update rule

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma max_{a'} Q(s', a') - Q(s, a))$$

where α is the learning rate. If both $\mathcal{S}$ and $\mathcal{A}$ are finite sets, the Q function can be easily represented in a table, namely in an $|\mathcal{S}| \times |\mathcal{A}|$ matrix, where each state-action pair is saved along with its discounted return estimation. In this case, the convergence of Q-learning has been proven in the past (under standard assumptions). See Sutton and Barto Sutton and Barto (1998) for more details.

In this work we focus on the Q-learning algorithm with a tabular representation of the Q function. This scheme is, perhaps, the most basic and commonly applied in RL tasks. RL can often suffer from slow learning speeds. To address this problem, designers infuse domain-specific knowledge into the agent's learning process in different ways, enabling better generalization across small numbers of samples. One may consider this approach as a human designer providing advice to an RL learner as opposed to the common framework in which agents advise people (e.g., Rosenfeld and Kraus (2016); Rosenfeld et al. (2016, 2015a,b); Azaria et al. (2015); Levy and Sarne (2016)). The most common approach is to use FA. When using FA, the designer needs to abstract the state-action space in a sophisticated manner such that the (presumed) similar states or state-action pairs will be updated together and dissimilar states or state-action pairs are not. **ARIEL: need an example here for the successful use of FA in something recent...** With the recent successes of DeepRL Mnih et al. (2015) **ARIEL: isn't there something newer to add here? Maybe mention AlphaGo?**, convolutional neural networks were shown to successfully learn features directly from pixel-level representations. However, such features are not necessarily optimal;

significant amounts of designer time are necessary to define the deep neural network's architecture, and significant amounts of data are required to learn the features.

ARIEL: Need to talk about RS here (we talked about FA above)**ARIEL: need an example here for the successful use of FA in something recent...**

Our proposed method, SASS, leverages a human designer's constructivism Bruner (1957). Constructivism is a well-established psychological theory where people make sense of the world (situations, people, etc.) by making use of constructs (or clusters), which are perceptual categories used for evaluation by considering members of the same construct as *similar*. It has been shown that people who have many different, possibly overlapping, and abstract constructs have greater flexibility in understanding the world, and are usually more robust against inconsistent signals. The SASS approach is inspired by constructivism, allowing a designer to define both complex as well as simplistic constructs of similar state-action pairs, according to one's knowledge, abilities and beliefs, and refine them as more experience is gained. This approach is in contrast to more complex types of generalization (e.g., specifying the width of a tile, the number of tiles, and the number of tilings in a CMAC Albus (1981) or specifying the number of neurons, number of layers, and activation functions in a deep net).

The notion of generalization through *similarity* is also common in other techniques that allow the learning agent to provide predictions for unseen or infrequently visited states. For example, Texplore Hester and Stone (2013) uses supervised learning techniques to generalize the effects of actions across different states. The assumption is that actions are likely to have similar effects across states. Tamassia et al. Tamassia et al. (2016) suggest a different approach: dynamically selecting state-space abstraction by which different states that share the same abstraction features are considered similar. Sequeira et al. Sequeira et al. (2013) and Girgin et al. Girgin et al. (2007) have presented variations of this notion by online identifying associations between different states in order to define a state-space metric or equivalence relation. However, all of these methods assume that an expert RL designer is able to iteratively define and test the required similarities without explicit cost.

Another related line of research investigates providing direct biasing from non-expert humans, such as incorporating human-provided feedback Knox and Stone (2010); Peng et al. (2016) or demonstrations Brys et al. (2015). For example, one may ask a non-expert user to teleoperate the agent or provide on-line feedback to the agent's actions. **ARIEL: we can significantly extend this.. ** These methods do not require significant technical abilities from the part of the human (e.g., programming is not needed). In this article, we consider a possibly complementary approach, leveraging *technically-able human designers'* knowledge and technical abilities; either in term of providing an abstraction, a reward shaping function or a similarity function, to improve the agent's performance. The investigated approaches can be integrated with direct biasing as well. We leave the examination of non-technical methods and non-technical users to future work. **ARIEL: I'm not sure the difference between the non-technical input and non-technical users is clear. What do you think? **

Lastly, alternative updating approaches such as eligibility traces Sutton and Barto (1998), where multiple states can be updated based on time since visitation, and Dyna Sutton (1991), where the agent learns from both direct environmental interaction and by planning over an approximate model learned from experience, are popular as well. For ease of analysis, this study does not directly address such methods, but we note that the SASS approach is compatible with these and other additional speed-up techniques.

3 QS-learning

In order to integrate our SASS generalization approach within the Q-learning framework, we adopt a previously introduced technique Ribeiro (1995) where Q-learning is combined with a spreading function that "spreads" the estimates of the Q-function in a given state to neighboring states, exploiting an assumed spatial smoothness of the state-space. Formally, given an experience $\langle s, a, r, s' \rangle$ and a spreading function $\sigma : \mathcal{S} \times \mathcal{S} \mapsto [0, 1]$ that captures how *"close"* states s and s' are in the environment, a new update rule is used:

$$Q(\tilde{s}, a) = Q(\tilde{s}, a) + \alpha\sigma(s, \tilde{s})\delta \quad (1)$$

where δ is the temporal difference error term ($r + \gamma max_{a'}Q(s', a') - Q(s, a)$). The update rule in Eq. 1 is applied to all states in the environment after each experience. The resulting variation is denoted as QS-learning (S stands for spreading). This method was only tested with author-defined spreading functions in simple grid worlds.

Note that standard Q-learning is a special case of QS-learning by setting the function σ to the Kronecker delta ($\delta(x, y) = 1$ if $x = y$, otherwise $\delta(x, y) = 0$).

Proposition 1 *QS-learning converges to the optimal policy given the standard condition for convergence of Q-learning and either: 1) σ which is fixed in time; or 2) σ that converges to the Kronecker delta over the state-action space at least as quickly as the learning rate α converges to zero.*

Proof The proposition is a combination of two proofs available in Szepesvári and Littman (1999) and Ribeiro and Szepesvári (1996). Both were proven for the update rule of Eq. 1 without loss of generality, and therefore apply to the QS-learning update rule of Eq. 2 as well. □

The SASS Approach

According to constructivism literature, and specifically personal construct psychology Kelly (1955), while designing and testing an RL agent, the human designer himself learns the traits of the domain at hand by identifying patterns and domain-specific characteristics. To accommodate both prior knowledge and learned insights (which may change over time), it is necessary to allow the designer to easily explore and refine different similarity hypotheses. For example, a designer may have an initial belief that the state-action pair s, a has the same expected return as $\tilde{s}, \tilde{a}$. Using FA, this can easily be captured by mapping both pairs into a single meta state-action pair. However, after gaining some experience in the domain, the designer refines his belief and presumes that the two pairs are merely *similar* (they would have close expected returns if they were to be modeled separately). This difference can have a significant effect on both the learning efficiency and the resulting policy (which may be suboptimal).

In this study we assume that the similarity function is defined and refined by a human designer during the development of the RL agent as follows.

Definition 1 *Let $\mathcal{S}$, $\mathcal{A}$ be a state-space and an action-space, respectively. A similarity function $\sigma : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ maps every two state-action pairs in $\mathcal{S} \times \mathcal{A}$ to the degree to which we expect the two state-action pairs to have a similar expected return. σ is considered valid if $\forall\langle s, a\rangle$, $\sigma(s, a, s, a) > 0$.*

Similarity functions can be defined in multiple ways to capture various assumptions and insights about the state-action space. As shown in constructivism literature Bruner (1957), some people may use simplistic, crude similarities that allow quick generalizing of knowledge across different settings. Others may use complex and sophisticated similarity functions that will allow a more fine-grained generalization. Although people can easily identify similarities in real-life, they are often incapable of articulating sophisticated rules for defining such similarities. Therefore, in the following, we identify and discuss three notable similarity notions that were encountered repeatedly in our human study (Section 4), covering the majority of human-designed similarity functions in our tested domains.

1) Representational Similarity from the tasks' state-action space. FA is perhaps the most popular example of the use of this technique. The function approximator (e.g., tile coding, neural networks, abstraction, etc.) approximates the Q-value and therefore implicitly forces a generalization over the feature space. A common method is using a factored state-space representation, where each state is represented by a vector of features that capture different characteristics of the state-space. Using such abstraction, one can define similarities using a metric over the factored state-action (e.g., Sequeira et al. (2013); Brys et al. (2015)). Defining representational similarities introduces the major engineering concern of choosing the right abstraction method or FA that would work well across the entire state-action space, while minimizing

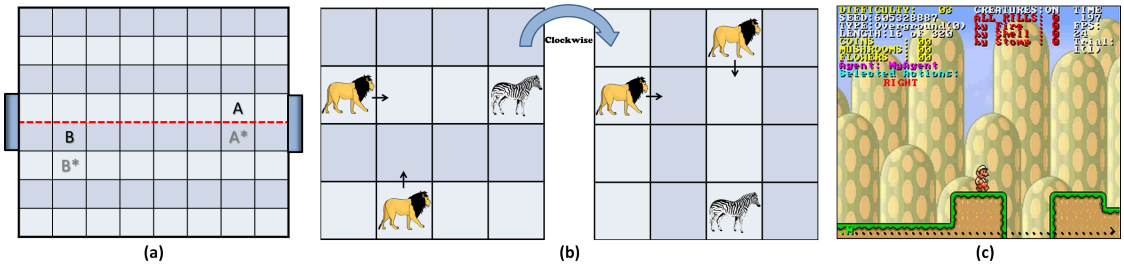


Figure 1: (a) Players in the simple robotic soccer task are A and B; one cell down (A* and B*) are considered similar. (b) Two similar state-action pairs in the Pursuit domain. (c) A state in the Mario AI task where walking or running right are similar (i.e., falling into the gap).

generalizing between dissimilar state-actions. Representational similarity has repeatedly shown its benefit in real world applications, but no one-size-fits-all method exists for efficiently representing the state-action space. See Figure 1 (a) for an illustration.

2) Symmetry Similarity seeks to consolidate state-action pairs that are identical or completely symmetrical in order to avoid redundancies. Zinkevich and Balch Zinkevich and Balch (2001) formalized the concept of symmetry in MDPs and proved that if such consolidation of symmetrical state-actions is performed accurately, then the optimal Q function and the optimal policy are not altered. However, automatically identifying symmetries is computationally complex Narayanamurthy and Ravindran (2008), especially when the symmetry is only *assumed*. For example, in the Pursuit domain, one may consider the 90°, 180° and 270° transpositions of the state around its center (along with the direction of the action) as being similar (see Figure 1 (b)). However, as the predators do not know the prey's (potentially biased) policy, they can only assume such symmetry exists.

3) Transition Similarity can be defined based on the idea of *relative effects* of actions in different states. A relative effect is the change in the state's features caused by the execution of an action. Exploiting relative effects to speed up learning was proposed Jong and Stone (2007); Leffler et al. (2007) in the context of model learning. For example, in the Mario domain, if Mario *walks right* or *runs right*, outcomes are assumed to be similar as both actions induce similar relative changes to the state (see Figure 1 (c)). In environments with complex or non-obvious transition models, it can be difficult to intuit this type of similarity.

SASS in the Q-learning Framework

We use the designer-provided similarity function $\sigma(s, a, s', a')$. We then use σ instead of the spreading function needed by QS-learning. In words, for each experience $\langle s, a, r, s' \rangle$ that the agent encounters, depending on the similarity function σ, we potentially update more than a single $\langle s, a \rangle$ entry in the Q table. Multiple updates, one for each entry $\langle \tilde{s}, \tilde{a} \rangle$ for which $\sigma(s, a, \tilde{s}, \tilde{a}) > 0$, are performed using the following update:

$$Q(\tilde{s}, \tilde{a}) = Q(\tilde{s}, \tilde{a}) + \alpha\sigma(s, a, \tilde{s}, \tilde{a})\delta \tag{2}$$

which, as discussed in Section 3, does not compromise the theoretical guarantees of the unadorned Q-learning.

The update rule states that as a consequence of experiencing $\langle s, a, r, s' \rangle$, an update is made to other pairs $\langle \tilde{s}, \tilde{a} \rangle$ as if the real experience was actually $\langle \tilde{s}, \tilde{a}, r, s' \rangle$ (discounted by the similarity function).

In order to avoid a time complexity of $O(|S||A|)$ per step, QS-learning should be restricted to update state-action pairs for which the similarity is larger than 0. In our experiments (see Section 4) we found only a minor increase in time-complexity for most human provided similarity functions.

In the interest of clarity, from this point forward we will use the term QS-learning using the above Q-learning-with-$SASS$ interpretation. Namely, using a designer-defined similarity function σ and the

Algorithm 1 QS-learning Algorithm

Require: State-space $\mathcal{S}$, Action-space $\mathcal{A}$, discount factor γ, learning rate α, similarity function $\sigma(s, a, s', a')$
initialize Q arbitrarily (e.g. $Q(s, a) = 0$)
for t=1,2,... **do**
s is initialized as the *starting* state
repeat
choose an action $a \in A(s)$ based on an exploration strategy
perform action a
observe the new state s' and receive reward r
$\delta \leftarrow r + \gamma \cdot \max_{a' \in A} Q(s', a') - Q(s, a)$ calculate temporal difference error
for each $\tilde{s}, \tilde{a} \in \mathcal{S} \times \mathcal{A}$ such that $\sigma(s, a, \tilde{s}, \tilde{a}) > 0$ **do**
$Q(\tilde{s}, \tilde{a}) = Q(\tilde{s}, \tilde{a}) + \alpha\sigma(s, a, \tilde{s}, \tilde{a})\delta$
$s \leftarrow s'$
until s' is a terminal state

update rule of Eq. 2, we will modify the classic QS-learning algorithm yet keep its original name due to their inherent resemblance. See Algorithm 1 for the QS-learning Algorithm as used in this study.

4 Evaluation

Our human subject study comprises of three experimental settings: First, we examine the SASS approach against the baseline and FA approaches, in the simple robotic soccer task with 16 non-expert developers. Through this experiment, which we will refer to as **Experiment 1**, we show the potential benefits of the SASS approach compared to FA given basic, classic reward shaping taken from previous works. Next, we evaluate all three approaches (FA, RS and similarity) using the Pursuit and Mario tasks, to examine the effect of reward shaping. Through this experiment, which we will refer to as **Experiment 2**, we show that reward shaping is the most natural approach of the three for most non-expert developers. However, the combination of RS and SASS (as was also shown in Experiment 1) suggests significant potential benefits for the developers. Last, in Experiment 3, we evaluate all three tasks using three expert developers. The results support our findings in Experiments 1 and 2, demonstrating high effectiveness for the combination of RS and SASS.

Throughout this section we will use the following notation: a basic Q-learning agent is denoted Q, a QS-learning agent is denoted QS, a Q-learning agent that uses state-space abstraction is denoted QA, a Q-learning agent that uses reward shaping is denoted QR and an agent which combines reward shaping and similarities is denoted QRS.

We first discuss the three domains we tested in this study followed by the three experiments.

4.1 Evaluated Domains

Studies have shown that games can serve as a motivational tool for computer science students **?** **MOSHE: missing reference. did you mean to "EAAI11-Taylor"?**. As a result, we chose three game setting as discussed below.

4.1.1 Simple Robotic Soccer

Proposed in Littman (1994), the task is performed on an 8×8 grid world, defining the state-space S. Two simulated robotic players occupy distinct cells on the grid and can either move in the four cardinal directions or stay in place (5 actions each). The simulated robots are designed to play a simplified version of soccer: At the beginning of each game, players are positioned according to Figure 1 and possession of the ball is assigned to one of the players (either the learning agent or the fixed, handcoded-policy

opponent[5]). During each turn, both players select their actions simultaneously and the actions are executed in random order. When the attacking player (the player with the ball) executes an action that would take it to a square occupied by the other player, possession of the ball goes to the defender player (the player without the ball) and the move does not take place. A goal is scored when the player with the ball enters the other player's goal region. Once a goal is scored the game is won; the agent who scored receives 1 point and the other agent receives -1 point and the game is reset. The discount factor was set to 0.9 as in the original paper.

We used a basic state-space representation as done in Martins and Bianchi (2013), which is, to our knowledge, the most recent investigation of the game. A state s is represented as a 5-tuple $\langle x_A, y_A, x_B, y_B, b\rangle$ where x_i and y_i indicate player i's position on the grid and $b \in \{A, B\}$ indicates which player has the ball. The action-space is defined as a set of 5 actions as specified above. Overall, the state-action space consists of approximately 41,000 state-action pairs.

4.1.2 Pursuit

The Pursuit task (also known as Chase or Predator/Prey task) was proposed by Benda et al. Benda (1985). For our evaluation we use the recently evaluated instantiation of Pursuit implemented in Brys et al. (2014). According to the authors' implementation, there are two predators and one prey, each of which can move in the four cardinal directions as well as stay in place (5 actions each) on a 20×20 grid world. The prey is caught when a predator moves onto the same grid cell as the prey. In that case, a reward of 1 is given to the predators, 0 otherwise. We refer the reader to the original paper for the complete description of the underlining MDP and parameters.

4.1.3 Mario AI

Super Mario Bros is a popular 2-D side-scrolling video game developed by the Nintendo Corporation. We use the popular Mario AI game, often used for the evaluation of RL techniques Karakovskiy and Togelius (2012). We use the recently evaluated formulation of the Mario AI task proposed by Suay et al. Suay et al. (2016). The authors use a 27-dimensional discrete state-variables representation of the state-space and model 12 actions that Mario can take. We refer the reader to the original paper for the complete description of the underlining MDP and parameters. Given the authors' abstraction of the state-space, the size of the state-action space is over 100 billion, though some of the states are never encountered in reality. For example, it is impossible to have Mario trapped by enemies from all directions at the same time. Due to the huge state-action space, and unlike the Simple Robotic Soccer and Pursuit, Q-learning without the authors' abstraction will not be evaluated.

In the Pursuit and Mario AI tasks, we use $Q(\lambda)$-learning and $QS(\lambda)$-learning, which are slight variations of the Q-learning and QS-learning algorithms that use eligibility traces Sutton and Barto (1998). The addition of eligibility traces to the evaluation was carried out as done by the authors of the recent papers from which the implementations have been taken. This allows us to compare the different approaches with recently provided baseline solutions without altering their implementation.

4.2 Experiment 1: Initial Non-Expert Developers Study

In this experiment we seek to investigate the potential benefits of the SASS approach. We focus on technically-able non-experts with some background in programming and RL. To that end, we recruited 16 Computer Science grad-students (4 PhD students and 12 Masters students, ranging in age from 23 to 43, 10 male and 6 female) majoring in AI to participate in the experiment and act as designers for two RL agents. All participants have prior knowledge of RL from advanced AI courses. The students are majoring in Machine Learning (7), Robotics (4) and other computational AI sub-fields (5).

We chose to start with the Simple Robotic Soccer domain, which is the simplest domain of the three evaluation domains of this study. Prior to the experiment, all participants participated in an hour-long

[5]The opponent was given a handcoded policy, similar to that used in the original paper, which instructs it to avoid colliding with the other player while it has the ball and attempts to score a goal. While defending, the agent chases its opponent and tries to steal the ball.

tutorial reminding them of the basics of Q-learning and explaining the Simple Robotic Soccer task's specification. The tutorial was given by the first author of this article, an experienced lecturer and tutor. Participants were given two python codes: First, an implemented QA agent in which participants had to design and implement a state-space abstraction. Specifically, the participants were requested to implement a single function that translates the naïve representation to their own state-space representation. Second, participants were given a QS agent in which they had to implement a similarity function. Both codes already implemented all the needed mechanisms of the game and the learning agents, and they are available at `http://www.biu-ai.com/RL`. Under both conditions, a basic reward shaping was implemented as suggested in the original simple robotic soccer paper. The suggested reward shaping is of a Potential Based Reward Shaping (PBRS) structure Ng et al. (1999). Note that PBRS allows one to modify the reward function without altering the desired theoretical properties of Q-learning and QS-learning algorithms.

We used a within-subjects experimental design where each participant was asked to participate in the task twice, a week apart. In both sessions, the participants' task was to design a learning agent that would outperform a basic Q-learning agent in terms of asymptotic performance and/or average performance (one would suffice to consider the task successful) by using either abstraction or similarities, in no more than 45 minutes of work. To that end, participants were provided with a graph describing the learning performance the basic Q-Learning agent. Ideally, we want participants to take as much time as they need. However, given that each participant had to dedicate about 3 hours for the experiment (1 hour tutorial + 1.5 hours of programming and half an hour of logistics) we could not ask participants for more than 45 minutes per condition.

In order to ensure the scientific integrity of the submitted agents, participants were requested to perform the task in our lab, in a quite room, using a special Linux machine which we prepared for them. The machine was disconnected from any Internet service. Furthermore, while programming, a lab assistant was present to assist in any technical issue. No significant technical difficulties were encountered that might jeopardize the results. Participants were counter-balanced as to which function they had to implement first. We then tested the participant's submitted agents against the same handcoded opponent against whom they trained. After each session, subjects were asked to answer a NASA Task Load Index (TLX) questionnaire Hart and Staveland (1988). During each session, participants could test the quality of their designed agent at any time by using by running the testing procedure which worked as follows: The designed agent was trained for 1,000 games such that after each batch of 50 games, the learning was halted and 10,000 test games were played during which no learning occurred. The winning ratio for these 10,000 test games was presented to the designer after each batch. Given a 'reasonable' number of updates per step, the procedure does not take more than a few seconds on a standard PC. In order to allow designers to compare their agents' success to a basic Q-learning agent (the benchmark agent they were requested to outperform), each designer was given a report on a basic Q-learning agent that was trained and tested prior to the experiment using the same procedure described above.

After all agents were submitted, each agent was tested and received two scores: one for its average performance during its learning period and one for the asymptotic performance of the agent, i.e., its performance after the training is completed.

Results

Under the QS condition, participants defined similarity functions. A similarity function is beneficial only if it helps the QS agent outperform the basic Q agent. Otherwise, we say that the similarity function is "flawed" in that it hinders learning.

When analyzing the average performance of the submitted agents, we see that out of the 16 submitted QS agents, 12 (75%) successfully utilized a beneficial similarity function. On the other hand, only 3 (19%) of the 16 QA agents outperformed Q-learning. The average winning ratio recorded for the QS agents along their training was 68.2% compared to the 42.7% averaged by the QA agent and 60.8% averaged by the benchmark Q agent.

Asymptotically, 13 out of the 16 QS agents (81%) outperformed or matched the basic Q-learning performance. *None* of the QA agents asymptotically outperformed Q-learning. On average, under the QS-learning condition, participants designed agents that asymptotically achieved an average winning ratio of 74.5%. The QA-learning condition achieved only 47.7% and the Q agent recorded 72.5%.

Interestingly, *all 16 participants* submitted QS-learning agents which perform better than their submitted QA-learning agents both in terms of average learning performance and asymptotic performance. Namely, the QS agents' advantage over the QA agents is most apparent when examining each designer separately. Furthermore, for all participants, the QS agent outperforms the QA agent from the 3^{rd} test (the 150^{th} game) onwards. For 9 of the 16 participants (56%), the QS-learning agent outperformed the QA-learning agent from the first test onwards.

We further analyzed the types of similarities that participants defined under the QS-learning condition. This phase was done manually by the authors, examining the participants' codes and trying to reverse-engineer their intentions. Fortunately, due to the task's simple representation and dynamics, distinguishing between the different similarity notions was possible. It turns out that representational and symmetry similarity notions were the most prevalent among the submitted agents. In 8 out of the 16 QS agents, representational similarities were instantiated, mainly by moving one or both of the virtual players across the grid. Symmetry similarities were used by 7 out of the 16 participants. All 7 of these agents used the idea of mirroring, where the state and action were mirrored across an imaginary horizontal line dividing the grid in half. Some of them also defined mirroring across an imaginary vertical line dividing the grid in half, with an additional change of switching ball position between the players. While we were able to show that each of these ideas is empirically beneficial on its own, we did not find evidence that combining them brings about a significant change. Transitional similarities were only defined by 2 designers. Both designers tried to consider a more strategical approach. For example, moving towards the opponent while on defense is considered similar regardless of the initial position. It turns out that neither of the provided transitional similarities were beneficial on their own as they were defined.

Only 4 out of the 16 participants (25%) used more than a single similarity notion while defining the similarity function. Interestingly, the two best performing QS agents combined 2 notions in their similarity function. Therefore, we speculate that combining more than a single similarity notion can be useful for some designers, yet in the interest of keeping with the task's tight time frame, participants refrained from exploring "too many different directions" and focused on the ones they initially believed to be the most promising.

Recall that 4 participants (25%) submitted flawed similarity functions. Although these participants were unable to find a beneficial similarity function, the submitted agents were not considerably worse than the basic Q-learning. The average performance for these 4 agents was 56.9% compared to 60.8% for the basic Q-learning and their average asymptotic score was 61.5% compared to 72.5% for the basic Q-learning. Unlike the significant difference between the QA and QS conditions in terms of agents' performance, a much larger number of participants is needed to achieve significant results in terms of TLX scores. TLX scores are available at `http://www.biu-ai.com/RL`.

The results are summarized in Table 1.

Criteria	QS	QA	Q
Avg. Winning Ratio (during training)	68.2%	42.7%	60.8%
Avg. Winning Ratio (asymptotically)	74.5%	47.7%	72.5%
Better agent than benchmark (during training)	75%	19%	-
Better agent than benchmark (asymptotically)	81%	0%	-
Best Agent (during training)	75%	0%	25%
Best Agent (asymptotically)	81%	0%	19%

Table 1: Summary of main results from the Experiment 1 non-expert study.

4.3 Experiment 2: Non-Expert Developers Study

In Experiment 2, we seek to investigate three speedup methods: FA, RS and SASS. We again focus on technically-able non-experts with some background in programming and RL. Similar to Experiment 1, we required 32 human participants, all of which senior Bachelor or graduate students who major in AI and have participated in an advanced AI course. Participants are ranging in age from 20 to 50, 23 male and 9 female. The students are majoring in Machine Learning (22), Robotics (7) and other computational AI sub-fields (3). To do: Moshe: Add demographics **MOSHE: done, add mean age**.

Unlike Experiment 1, in this experiment we seek to evaluate the two more complex domains: Pursuit and Mario AI. First, we divided the 32 participants into two groups of 16 randomly. Each group was assigned a different domain. Participants were given three Java codes: First, an implemented QA agent in which participants had to design and implement a state-space abstraction. Specifically, the participants were requested to implement a single function that translates the basic representation to their own state-space representation. Second, participants were given a QS agent in which they had to implement a similarity function. Last, participants were given a QR agent in which they had to implement the reward shaping function $F(s, a, s')$. To do: Moshe - was this a function? How did it work? Namely the participant got a representation of the source state, the last performed action and a representation of the target state, and required to return a decimal value, which represents the reward shaping value. **MOSHE: done**. All codes already implemented all the needed mechanisms of the game and the learning agents, and they are available at `http://www.biu-ai.com/RL`. To do: Moshe - need to update the site. Take the HTML from Oleg and fix this please. *Unlike Experiment 1, no prior reward shaping was implemented.*

We again use a within-subjects experimental design where each participant was asked to participate in the task thrice, a week apart between every two consecutive conditions. Due to the increased complexity of the two domains tested in this experiment compared to Experiment 1, and to allow easy reproducibility of the experiment, participants were given an interactive PowerPoint presentation that introduced the problem domain as well as reminded them of the fundamentals of the tested speedup method. The PowerPoint presentations are available on our website `http://www.biu-ai.com/RL`. To do: Moshe - do this please. As before, in all sessions, the participants' task was to design a learning agent that would outperform a basic Q-learning agent in terms of asymptotic performance and/or average performance (one would suffice to consider the task successful) by using either abstraction, similarities or reward shaping, in no more than 45 minutes of work.

Similar to Experiment 1, participants had to use a specific Windows machine, at our lab, and were assisted by a lab assistant in case they faced any technical difficulties.

During each session, participants could test the quality of their agent by running the code and examine the presented learning process graph, which presented the average training score for each batch of 100 games and updated on-line as the learning proceeded. For the Pursuit task, given a 'reasonable' code, the procedure does not take more than a few seconds on a standard PC. For the Mario AI task, which is more complex and it's code runs a bit slower, the process took up to a half minute. **MOSHE: add total episode count for each experiment**

the following testing procedure: For the Pursuit task – the agent is trained for ? games such that after each batch of ? games, the learning was halted and ? test games were played during which no learning occurred. For the Mario AI task – the agent is trained for ? games such that after each batch of ? games, the learning was halted and ? test games were played during which no learning occurred. The winning ratio for these ? test games was presented to the designer after each batch. For both conditions, given a 'reasonable' number of updates per step, the procedure does not take more than a few seconds on a standard PC. To do: Moshe - is this still true? fill in the numbers above **MOSHE: done. I used a slightly different method, see the previous paragraph I added.**

In order to allow designers to compare their agents' success to a basic Q-learning agent (the benchmark agent they were requested to outperform), each designer was given a report on a basic Q-learning agent that was trained and tested prior to the experiment using the same procedure described above.

Following each programing session, the participants were asked to answer a NASA Task Load Index (TLX) questionnaire Hart and Staveland (1988). In addition, in order to acquire better understanding on

participants' subjective experience, an additional short questionnaire was administered. The questionnaire consisted of 9 statements to which participants had to rate the degree to which each statement reflects their subjective feeling on a 10-point Likert scale. For example, "To what extent the speedup method you used was appropriate for the task you required to complete?". The complete questionnaire is available at Appendix A. To do: Moshe - Talk to me about this. I need to understand what exactly you did... And add an appendix with the questionnaire.

Similar to Experiment 1, following the submission of all agents, each agent was trained and tested. For the Pursuit agent, each agent was evaluated by running 200,000 training episodes. After each 10,000 episodes batch the learning was halted and a 10,000 evaluation session performed during which no learning occurred. This process repeated 50 times for each agent. For the evaluation of Mario AI agents we run 20,000 training episodes, and after each 1,000 episodes batch the learning was halted and a 1,000 evaluation session performed during which no learning occurred. This process repeated 50 times for each participant code. For this evaluation we used the same machine used by the study participants, a PC machine with 12 GB of RAM and a CPU with eight cores, each operating at 3 GHz. To do: Use the same format we used in experiment 1 **MOSHE: done**

In addition to the evaluation of the three methods that the study participants developed during this experiment, we evaluated an additional condition. We manually combined each of the developers' QR agent with her QS agent, resulting in a new agent QRS. Note that each of the resulting QRS agent uses both the reward shaping and similarity implementations of a specific participant. It is important to mention that participants developed each agent independently and were not informed about this future combination. In total, 128 agents were evaluate for the two domains combined (32 participants, 4 agents each).

In the Pursuit and Mario AI tasks, we use $Q(\lambda)$-learning and $QS(\lambda)$-learning, which are slight variations of the Q-learning and QS-learning algorithms that use eligibility traces Sutton and Barto (1998). The addition of eligibility traces to the evaluation was carried out as done by the authors of the recent papers from which the implementations have been taken. This allows us to compare the different approaches with recently provided baseline solutions without altering their implementation.

Results

We report the results for each group separately.

Pursuit: Recall that a submitted agent is considered successful if it outperforms the basic Q-leaning agent in at least one of the two criteria of interest: average performance or asymptotic performance.

When analyzing the average performance of the submitted agents, we see that out of the 16 submitted QS agents, 14 (87.5%), successfully utilized a beneficial similarity function. Similar to the results of Experiment 1, only very few of the submitted QA agents (4 out of 16, 25.0%) where able to outperform the basic Q-learning agent. When examining the submitted QR agents, we see an astonishing 93.75% successful agents (15 out of 16). As for the QRS agents, 14 out of the 16 agents (87.5%) were successful, similar to the QS condition.

Asymptotically, To do: Moshe - Need to rewrite this in the style below... the results were very similar- 12 out of the 16 QS agents (75.0%) outperformed or matched the basic Q-learning performance; 8 of the 16 QA agents (50.0%); and almost all of the submitted RS agent, with 93.75% successful agents (15 out of 16). As to previous analysis, the QRS condition showed a slight improve over the QS condition, with 13 successful agents out of 16 (81.25%).

When considering a valid agent as one that outperformed the Q agent in at least one of the above aspects, we get that the average performance criteria is the more generalized, as all the agents that managed to outperform the average training performance of the baseline agent managed to get score lower asymptoticly as well. Interestingly, that holds for all agent types except of the QA condition, in which such relation not found. For such definition of valid agent, under the QS condition, participants designed agents that achieved an average training score (the lower the better) of 110.07, compared to an average of 131.7 with the QA condition, and to the amazing results of the QR condition which scored 44.66 on average. Consistent with previous results, the QRS condition scored 37.28 on average, slightly

improving the QR condition results. The basic Q agent average training score is 143.97, which makes the agents designed with the various speedup methods superior. We further analyzed the final scores of the valid agents. The average final score of valid agent designed with the QS condition is 33.03, compared to an average of 54.93 with the QA condition, and to 25.47 with the QR condition. Surprisingly, the QRS condition haven't managed to improve the QR condition results this time, scoring a bit higher (which means- worse) at 28.59. Such results indicates that the combination of QS and QR mainly improves the convergence rate of the resulted agent to outperform the QR agent, but not so much with the asymptotic results, as it only managed to get close the the QR agent result and not further. The asymptotic result of the Q agent is 36.08, outperforming the average result of the QA condition agents only.

All of the above results are summarized in Table 2.

Criteria	QS	QA	QR	QRS	Q
# valid agents with avg. training performance higher than baseline avg. training performance	14 (87.5%)	4 (25.0%)	15 (93.75%)	14 (87.5%)	-
# valid agents with final score higher than baseline final score	12 (75.0%)	8 (50.0%)	15 (93.75%)	13 (81.25%)	-
Total # valid agents (in at least one way)	14 (87.5%)	11 (68.75%)	15 (93.75%)	14 (87.5%)	-
Average training performance of valid agents (the lower the better)	110.07	131.70	44.66	37.28	143.97
Average asymptotic score of valid agents (the lower the better)	33.03	54.93	25.47	28.59	36.08
Average training performance of all agents (the lower the better)	115.51	245.03	321.75	404.91	143.97
Average asymptotic score of all agents (the lower the better)	38.21	192.67	314.15	383.21	36.08

Table 2: Summary of main results from the Experiment 2 Pursuit non-expert study.

As for the flawed agents, only 3 of them (9 in total) presented performance that are close to the baseline performance (up to 10% worse), while all others were at least 4 worse, up to 120 times, with no significant distinction between the speedup methods.

Considering the average training performance as agent comparison criteria, we notice that 7 participants developed an QR agent that bits all of other agents they submitted, and for 7 participants the same is true for their QSR agent. Only 2 participants developed an QS agent better than their other agents, and $none$ with the QA condition. This results reflects the fact that the performance with the QSR condition are very similar to the QR condition, and in many cases even surpasses it. Deeper head-to-head analysis reveals similar trend- 12 participants developed a QS agent better than their QA agent; 13 developed the QR and QSR agents better than the QS agent and 14 developed them better than their QA agent. For 8 participants the combination of the QS and QR agents resulted an agent that outperform the QR agent performance.

Finally, we analyzed the types of the developed agents and the ideas behind them to shed light on the experiment results from another angle. This part is done manually by the authors, mainly by reverse-engineering the submitted agents. In total, 5 of the participants considered the symmetrical properties of the board while developing their QS agents, while 4 of them integrated rotational calculations (means that state-action pairs considered similar in case one is the rotated other) and 3 used mirroring. All of the participants that used the mirroring method implemented rotational similarities as well. Total of 9 participants developed their QS agents that calculates similar state-action pairs based on the target state- such pairs considered similar in case both lead to the same state or to close ones. As for the QR agents,

most of participants (14 out of 16) developed agents based on the idea of motivating the predator to move towards the prey and discouraging it to move in other directions. This idea is very natural, in a sense, and can shed light on the presented results, as it very reasonable, effective and easy to implement. Further more, that idea alone was enough to produce outstanding results. For instance, only 2 participants referred to ótherṕredator location while calculating the shaped reward, even though this idea is natural as well, making it not essential for a good performing agent. In addition, turns out that the relative distance to the prey has only a small influence on the performance, as only 4 participants depended the shaped reward value based on it.

As mentioned earlier, after each development section the participant required to fill a NASA TLX questionnaire and another custom one. We run an ANOVA one-way test on the TLX questionnaire results in order to examine the task load differences. The ANOVA test results turns out significant ($F = 6.79348$) so we further analyzed the speedup methods results in pairs using T-Test in to reveal the difference origin. From the T-Test results we conclude that the TLX results of both Abstraction and Reward Shaping speedup methods distributes the same, while the TLX results of the Similarities speedup method has a different distribution. An analysis of the TLX questionnaire components reveals that the Mental, Temporal and Frustration components has a different distributions while the Physical, Performance and Effort components distributes the same across all the tested speedup methods. In addition to providing a reasonable explanation to the QS agent results', such results indicates that the Similarities method demands a higher level of mental effort in order to be implemented appropriately in the Pursuit domain, which makes it a bit less efficient comparing to the Abstraction and Reward Shaping methods. The full TLX results and tests results can be found at the project website- To do: add website address here.

In addition to the TLX questionnaire we set a custom questionnaire the aims at extracting the participants subjective experience during the experiment. For instance, we asked the participants to evaluate their agent performance quality in compare to the basic Q-learning agent performance. The results show that in some cases the participant reported about significant improvement over the baseline, while the true results head-to-head comparison reveals only a slight difference, and vice versa. Most of the questions in the introduced a fixed 10-point Likert scale, while some were free text answers. The original form phrased in Hebrew due to local preferences, but its translation to English can be found at Appendix A.

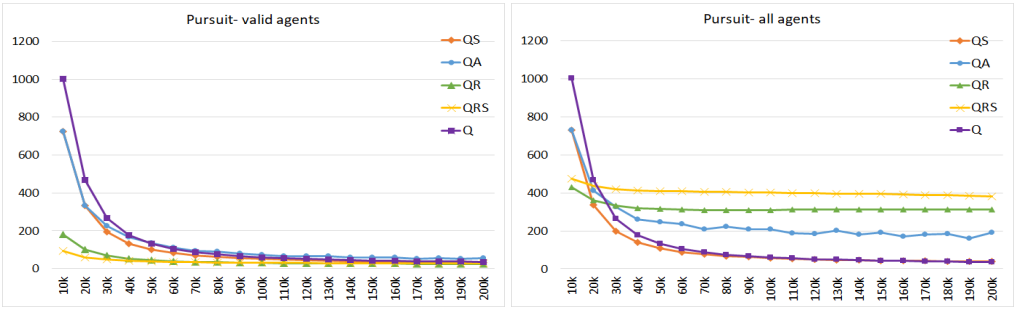
From the results we conclude that the participants understood their task requirements and purpose well, regardless of the speedup method that been used. In consistent with the submitted agents observed results, the participants reported that they managed to implement their knowledge in the agent using the QR and QA speedup methods much better that with the QS method. In addition, the participants rated the Q speedup method as the least appropriate for the assigned task (i.e. outperforming the baseline agent results), and that the time they invested using it was justified to a small extent.

The full questionnaire results can be found at the project website- To do: add website address here.

–¿ Asymptotically, 13 out of the 16 QS agents (81%) outperformed or matched the basic Q-learning performance. *None* of the QA agents asymptotically outperformed Q-learning. On average, under the QS-learning condition, participants designed agents that asymptotically achieved an average winning ratio of 74.5%. The QA-learning condition achieved only 47.7% and the Q agent recorded 72.5%.

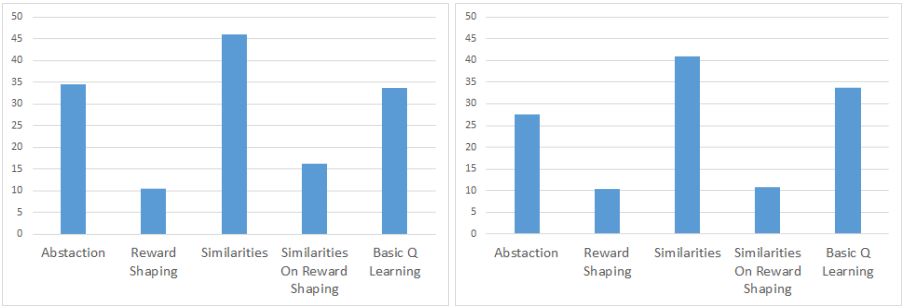
To do: NEED to add much more analysis here... In the spirit of Experiment 1... Moshe - Talk to me about this... We need the following: 1) comparison for ave. performance (not only flawness); 2) comparison for asymp. performance(not only flawness); 3) comparisons head-to-head (For how many participants the BEST agent was QS for example) 4) ANALYSIS OF THE QRS AGENTS; 5) TYPES OF SIMILARITIES AND REWARD SHAPING USED; 6) Discuss the flawed agents. How bad were they? See a similar discussion in Experiment 1; 7) Graphs and tables with correct names; 8) analysis of TLX; 9) analysis of post questionnaire

To do: We need to fix the graphs... Maybe have one with all agents and one with only valid... **MOSHE: done**



(a) Valid agents average learning curve (b) All agents average learning curve

Figure 2: Pursuit domain agent learning curves. The horizontal axis is the total amount of train episodes passed, the vertical axis is the average agent score (lower is better).



(a) Average score factor- average train session duration (b) Final score factor- average train session duration

Figure 3: Pursuit domain agents' train session average duration. The vertical axis is in seconds.

To do: IMPORTANT - Moshe, can we test MANUALLY the abstraction and RS combination? It would be very useful. TALK TO ME ABOUT THIS! **MOSHE: we talked about it, probably not possible. ignoring for now**

Mario AI: To do: Moshe - write the results that you have for MARIO – WITHOUT DISCUSSION. I'll write why it didn't work AI needs some good failures (smoking letters)....

Similar to Pursuit domain, a submitted agent considered successful if it outperforms the basic Q-learning agent in at least one of the two criteria defined earlier- average performance and asymptotic performance.

Considering the average performance criteria, under the QS learning condition only 3 of 16 (18.75%) submitted agents are successful, similar to the QRS-learning condition results. The results with both other speedup methods aren't significant higher- only 4 (25.0%) successful agents under the QR condition and 5 (31.25%) under the QA condition. Under the asymptotic performance criteria the results are even worth- 5 of 16 (31.25%) submitted QA agents are successful, only 2 (12.5%) of the QR agents and $None$ of the QS and QRS agents. The results are summarized in Table 3. See Figure 4a and Figure 4b for graphical representation.

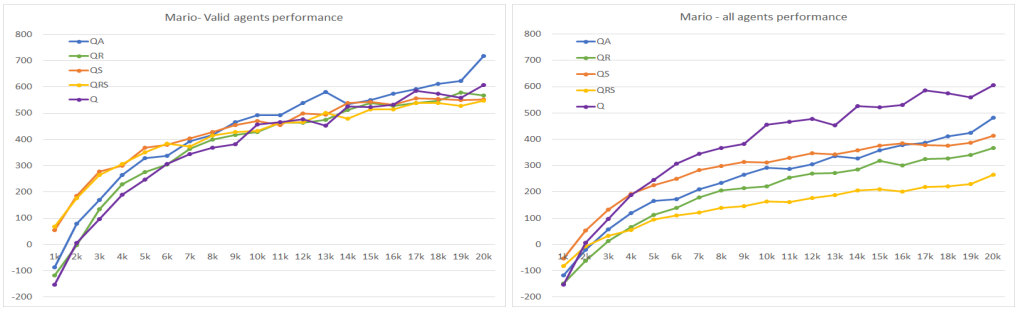
To do: discuss why it didn't work

4.4 Experiment 3: Expert Developers Study

While Experiments 1 and 2 focus on non-expert technically-able human designers, in Experiment 3 we seek to investigate RL experts. To that end, we recruited 3 highly experienced, expert programmers with

Criteria	QS	QA	QR	QRS	Q
# valid agents with avg. training performance higher than baseline avg. training performance	3 (18.75%)	5 (31.25%)	4 (25.0%)	3 (18.75%)	-
# valid agents with final score higher than baseline final score	0 (0.0%)	5 (31.25%)	2 (12.5%)	0 (0.0%)	-
Total # valid agents (in at least one way)	3 (18.75%)	6 (37.5%)	5 (31.25%)	3 (18.75%)	-
Average training performance of valid agents (the higher the better)	429.67	433.79	382.38	414.56	377.10
Average asymptotic score of valid agents (the higher the better)	551.22	717.57	567.71	547.13	606.40
Average training performance of all agents (the higher the better)	284.49	253.76	199.59	142.70	377.10
Average asymptotic score of all agents (the higher the better)	413.67	482.50	366.08	265.09	606.40

Table 3: Summary of main results from the Experiment 2 Mario non-expert study.



(a) Valid agents average learning curve

(b) All agents average learning curve

Figure 4: Mario domain agent learning curves. The horizontal axis is the total amount of train episodes passed, the vertical axis is the average agent score (the higher the better).

Masters degrees in Computer Science and proven experience in RL (two of whom are 26 years old and the third is 27 years old). None of the experts co-author this paper. Each expert was asked to implement 6 RL agents: a basic Q-learning agent; a QS-learning agent; a QA-learning agent, a QR-learning agent; a QRS-learning agent; and a Dyna agent (denoted $Dyna$). Each expert was given one RL task: Simple Robotic Soccer, Pursuit and the Mario AI game as discussed in Section 4.1.

Each expert was instructed to take as much time as he needs to implement the agents yet keep track over the invested time to implement each condition. After all agents were submitted, the experts where asked to fill a questionnaire, available at Appendix **??** through which they were debriefed for their subjective experience. To do: Moshe - See comment below about this.

Unfortunately, we were unable to get all three experts to come to our lab. As a result, each expert used his personal computer for the programming of the different agents. The reported running times of the agents were evaluated post-hoc on a personal Linux computer with 16 GB RAM and a CPU with 4 cores, each operating at 4 GHz. All technical parameters used by the three experts in this study (learning rates, exploration type, eligibility trace parameter, etc.) are fully specified in the code and are available at

`http://www.biu-ai.com/RL`. To do: Moshe - the website needs a complete makeover. Please take care of it.

4.4.1 Simple Robotic Soccer

For this task, our expert is a 26 years old programmer who works as a scientific programmer at one of the Israeli Universities. He completed a Master degree majoring in AI and have worked on RL in the past.

We discuss the non-trivial submitted agents:

QA used a simple distance-based approach, which represented each state according to the learning agent's distance to its opponent and goal.

QS used two major similarity notions: First, *representational similarities* – the agent artificially moves *both players* together across the grid, keeping their original relative distance (see Figure 1). As the players are moved further and further away from their original positions, the similarity estimation gets exponentially lower. Second, *symmetry similarities* – experiences in the upper half of the field are mirrored in the bottom part by mirroring states and actions with respect to the Y-axis and vice-versa. *Transition similarities* were not defined by the expert for this task.

QR used a shaping reward similar to the one proposed in Bianchi et al. (2014). The expert defined that moving towards the goal while on offense and towards the opponent while on defense receives a PBRS. Therefore, whenever an action is intended to change the proximity (using Manhattan distance) to the attacker or the goal (depending on the situation). To do: Moshe - make sure this is correct... **MOSHE: verified, this is correct**

Results: Each agent was trained for 2,000 games. After each batch of 50 games, the learning was halted and 10,000 test games were played during which no learning occurred. The process was repeated 350 times. The results show that the QR condition agent outperforms all other agents from the first batch up to the 19th batch, which than is passed by the QS agent. The combination of both agents results the QRS agent that outperforms all agents from the first batch onwards. See Figure 5a for a graphical representation of the learning process.

The run-time for QS was significantly worse than the other speedup methods, which slows down the QRS agent for a great extent. With 2,000 games each, the QS and QRS agents averaged 0.0278 and 0.019 seconds for a game, respectively, significantly longer even from the baseline, which averaged 0.0022 seconds. The QA was a bit slower than the Q agent, with average of 0.0037 seconds for a game. The only agent that outperformed the baseline agent was the QR agent, which completed each game in 0.0016 seconds in average. To do: Moshe - need to update with the QR and QRS **MOSHE: done**

Reflections

To do: Moshe - we need to interview the experts and summarize their reflections on the task.... Similar to what you did for the non-experts I think... SET A QESTIONEER AND INTERVIEW THE EXPERTS ASAP! things like How many ideas they tested and others... What was hard or easy... ADD the questionnaire as Appendix B **MOSHE: we talked about it, I interview each expert "unformally" and write down their conclusions and experience**

The expert developed the QS agent based on ideas and thoughts he had while the development of the basic Q-learner agent. The implementation of those ideas was complicated, so the expert had to depend on 'trial-and-error' most of the time. Comparing to the QA learning method, the expert rate the later as the more natural way to translate his knowledge into code that assist agent's learning. He explains that the Abstraction method is more similar to the way peoples are evaluating their surrounding before they decide which action to take. For example, on the road junction, the driver ignores most of the available information around him and focuses solely on the traffic lights color in order to decide whether or not to drive forward or keep still. As for the QR method, the expert believes that this is the most natural and convenient way to accelerate learning, since it enables to integrate his idea in the learning process in a very simple and straight forward way almost without translations. Finally, the expert believe that he would choose the QR method for future similar tasks.

4.4.2 *Pursuit*

For this task, our expert is a 27 years old programmer who works as a senior programmer at the Israeli Defense Force (IDF) **ARIEL: Should we say that?**. He completed a Master degree majoring in AI and have worked on RL in the past.

We discuss the non-trivial submitted agents:

QA was already defined by Brys et al. Brys et al. (2014) who implemented tile-code approximation. The expert did not see a reason to change the QA implementation.

QS was defined based on linear differences and angular rotations. Each state is represented as $\langle \Delta_{x_1}, \Delta_{y_1}, \Delta_{x_2}, \Delta_{y_2} \rangle$ where Δ_{x_i} (Δ_{y_i}) is the difference between predator i's x-index (y-index) and the prey's x-index (y-index), thereby a similarity of 1 was set for all states in which the relative positioning of the prey and predators is the same. Symmetry similarities were defined using 90°, 180° and 270° transpositions of the state around its center (along with the direction of the action). Furthermore, experiences in the upper (left) half of the field are mirrored in the bottom (right) part by mirroring states and actions. Transition similarities were defined for all state-action pairs that are expected to result in the same state.

QR To do: Moshe - add here. **MOSHE: done** was designed based on a simple logic that encourage predator steps towards the prey and punish on steps any other direction. The shaping applied very gently, namely the shaping value that chosen by the expert was $\pm 10^{-22}$, which is very low compared to the final states reward signal, which is set to ± 1.

Results: Each agent was trained for 10,000 games. After each batch of 100 games, the learning was halted and 10,000 test games were played during which no learning occurred. The process was repeated 50 times. The results show that the QR agent is the most efficient one and that it learns significantly faster than other agents. In addition, both the QS and QA agents outperforms the baseline agent and show big improvement in convergence rate. $Unexpectedly$, the combination of QR and QS agents resulted an agent that is $worse$ than each of them, and even from the baseline agent. Initially, the agent shows performance better than the QR agent, but surprisingly, after the first four batches it stops learning and begin regression which leads to divergence. See Figure 5b for a graphical representation of the learning process. To do: Moshe - need to update with the QR and QRS **MOSHE: done**

On average, QS updated 12 entries per iteration. However, while Q and QA-learning agents complete their training (10,000 games each) in 8.5 seconds on average (with no significant difference between the two), QS completes the same training in 17.5 seconds on average. $Dyna$ performed significantly worse, averaging 87 seconds for its training. Unlike other conditions, $Dyna$ also introduced extreme memory requirements due to its model-based approach.

Reflections: To do: Moshe - same as before

4.4.3 *Mario AI*

For this task, our expert is a 27 years old programmer who works as a software development team leader at a large international hi-tech company. He completed a Bachelor degree in Computer Science and has more than 10 years programming experience, including work with RL.

We discuss the non-trivial submitted agents:

QA was implicitly defined by the original authors as they already abstracted the state space. The expert did not change the given abstractions.

QS was defined on top of the authors' abstraction. It turns out that each state representation indicates whether Mario can jump or shoot using 2 Boolean variables. Given a state-action pair in which Mario does not jump or shoot, all respective states (with the four variations of these two Boolean variables) were defined as similar to the original pair. Namely, if Mario walks right, then regardless of Mario's ability to shoot or jump, the state-action pair is considered similar to the original one. Symmetry similarities are defined using the mirroring of the state-actions across an imaginary horizontal line that divides the environment in half, with Mario in the middle. As illustrated in Figure 1, regardless of specific state, performing action a (e.g., move right) is assumed similar to using action a+"run" (e.g., run right).

QR was ? To do: Moshe - I asked him to implement it, but it will take some time (he is abroad). Remind me of this!!!

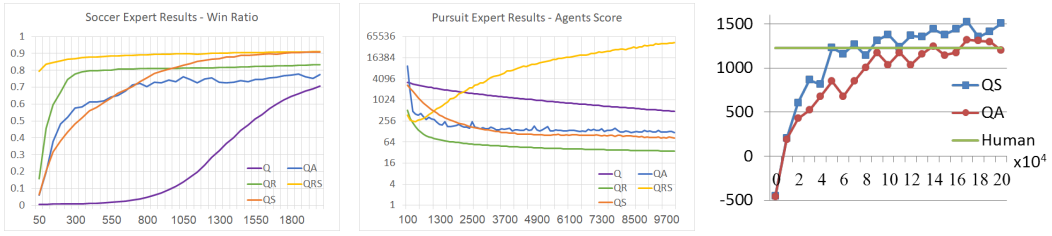
Results: Each agent was trained for 20,000 games. After each batch of 1,000 games, the learning was halted and 1,000 test games were played during which no learning occurred. The process was repeated 100 times. The two agents are also compared to human performance level as evaluated in Suay et al. (2016). The results show that the QS-learning agent surpasses the human performance level significantly faster than the QA-learning agent.

Due to extreme memory requirements in run-time, the expert was unable to evaluate the $Dyna$ condition properly.

See Figure 5c for a graphical representation of the learning process.

To do: Moshe - update the results with QR and QRS

On average, the QS updated 33 entries per iteration. However, the QA requires 63.4% of QS's runtime to complete its training (20,000 games each, 33 vs. 55 seconds).



(a) Simple Robotic Soccer (b) Pursuit (c) Mario AI

Figure 5: The QS-learning agent outperforms QA-learning, Q-learning and $Dyna$ agents in all three domains. The x-axis marks the number of training games. The Y-axis marks the average game score; in Simple Robotic Soccer and Mario AI, the higher the better, and in Pursuit, the lower the better. Notice that the Y-axis at the Pursuit graph is in logarithmic scale

5 Conclusions

ARIEL: I'll rewrite this once we have all the results! We conducted a first-of-its-kind human study that explores human aspects in speeding up RL agents. We focused on the task of injecting human knowledge into an RL learner using the notion of similarity and generalization. We present a new approach, SASS, which calls for the integration of simple handcoded state-action similarities into RL algorithms in order to bring about a more natural, incremental and simple means for a designer to leverage his prior knowledge. SASS is shown to be both effective and efficient, for expert and non-expert technically-able designers. Namely, we show that by using SASS, human designers were better at leveraging their knowledge for speeding up tabular RL compared to a classic FA approach. We hope this work will inspire other researchers to investigate their approach in human studies with actual programmers.

In future work, we will tackle the challenge of on-line identification and refinement of similarities with and without a designer's input. Moreover, we plan to extend the proposed approach to other RL algorithms (e.g., linear function approximation and DeepRL) and techniques (e.g., learning from demonstrations, reward shaping, etc.). We further plan to include non-technical users.

Acknowledgment

This article extends our previous reports from AAMAS 2017 Rosenfeld et al. (2017b) and IJCAI 2017 Rosenfeld et al. (2017a) in several major aspects: First, in Rosenfeld et al. (2017b), the SASS approach was presented and tested with three experts as described in Section **??**. Then, in Rosenfeld et al. (2017a), the study was extended to include additional 16 non-expert designers who implemented the QS and

QA learning conditions as discussed in Section **??**. In this article we almost triple our participant pool by recruiting additional 32 participants in an additional experiment (Experiment 2, Section **??**). Furthermore, we investigate the reward shaping condition which was not investigated in previous reports thus extending our expert study accordingly. Trough this additional experiment, which includes two domains not investigated with non-experts, an additional speedup method and almost tripling the subject pool compared to previous reports, we were able to enhance the credibility and validity of our previously reported results and demonstrate new insights such as the benefits of our proposed methodology combined with reward shaping.

The extended version of Rosenfeld et al. (2017b), entitled "Speeding up Tabular Reinforcement Learning Using State-Action Similarities" have received the *Best Paper Award* at the Fifteenth Adaptive Learning Agents workshop at AAMAS 2017.

This research was funded in part by MAFAT. It has also taken place at the Intelligent Robot Learning (IRL) Lab, which is supported in part by NASA NNX16CD07C, NSF IIS-1149917, NSF IIS-1643614, and USDA 2014-67021-22174.

References

Albus, J. S. (1981). *Brains, Behavior and Robotics*. McGraw-Hill, Inc., New York, NY, USA.

Azaria, A., Rosenfeld, A., Kraus, S., Goldman, C. V., and Tsimhoni, O. (2015). Advice provision for energy saving in automobile climate-control system. *AI Magazine*, 36(3):61–72.

Benda, M. (1985). On optimal cooperation of knowledge sources. *Technical Report BCS-G2010-28*.

Bianchi, R. A., Martins, M. F., Ribeiro, C. H., and Costa, A. H. (2014). Heuristically-accelerated multiagent reinforcement learning. *IEEE transactions on Cybernetics*, 44(2):252–265.

Bruner, J. S. (1957). Going beyond the information given. *Contemporary approaches to cognition*, 1(1):119–160.

Brys, T., Harutyunyan, A., Suay, H. B., Chernova, S., Taylor, M. E., and Nowé, A. (2015). Reinforcement learning from demonstration through shaping. In *IJCAI*, pages 3352–3358.

Brys, T., Nowé, A., Kudenko, D., and Taylor, M. E. (2014). Combining multiple correlated reward and shaping signals by measuring confidence. In *AAAI*, pages 1687–1693.

Busoniu, L., Babuska, R., De Schutter, B., and Ernst, D. (2010). *Reinforcement learning and dynamic programming using function approximators*, volume 39. CRC press.

Girgin, S., Polat, F., and Alhajj, R. (2007). Positive impact of state similarity on reinforcement learning performance. *IEEE Transactions on Cybernetics*, 37(5):1256–1270.

Hart, S. G. and Staveland, L. E. (1988). Development of nasa-tlx (task load index): Results of empirical and theoretical research. *Advances in psychology*, 52:139–183.

Hester, T. and Stone, P. (2013). Texplore: real-time sample-efficient reinforcement learning for robots. *Machine learning*, 90(3):385–429.

Jong, N. K. and Stone, P. (2007). Model-based function approximation in reinforcement learning. In *AAMAS*, page 95. ACM.

Karakovskiy, S. and Togelius, J. (2012). The Mario AI benchmark and competitions. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):55–67.

Kelly, G. (1955). *Personal construct psychology*. New York: Norton.

Knox, W. B. and Stone, P. (2010). Combining manual feedback with subsequent MDP reward signals for reinforcement learning. In *Proc. of AAMAS*.

Leffler, B. R., Littman, M. L., and Edmunds, T. (2007). Efficient reinforcement learning with relocatable action models. In *AAAI*, volume 7, pages 572–577.

Levy, P. and Sarne, D. (2016). Intelligent advice provisioning for repeated interaction. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *ICML*, volume 157, pages 157–163.

Martins, M. F. and Bianchi, R. A. (2013). Heuristically-accelerated reinforcement learning: A comparative analysis of performance. In *Conference Towards Autonomous Robotic Systems*, pages 15–27. Springer.

Mataric, M. J. (1994). Reward functions for accelerated learning. In *Machine Learning: Proceedings of the Eleventh international conference*, pages 181–189.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, Georg Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Narayanamurthy, S. M. and Ravindran, B. (2008). On the hardness of finding symmetries in markov decision processes. In *ICML*, pages 688–695.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287.

Peng, B., MacGlashan, J., Loftin, R., Littman, M. L., Roberts, D. L., and Taylor, M. E. (2016). A need for speed: Adapting agent action speed to improve task learning from non-expert humans. In *AAMAS*, pages 957–965.

Ribeiro, C. and Szepesvári, C. (1996). Q-learning combined with spreading: Convergence and results. In *Procs. of the ISRF-IEE International Conf. on Intelligent and Cognitive Systems (Neural Networks Symposium)*, pages 32–36.

Ribeiro, C. H. (1995). Attentional mechanisms as a strategy for generalisation in the q-learning algorithm. In *Proceedings of ICANN*, volume 95, pages 455–460.

Rosenfeld, A., Agmon, N., Maksimov, O., Azaria, A., and Kraus, S. (2015a). Intelligent agent supporting human-multi-robot team collaboration. In *Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI, Buenos Aires, Argentina, July 25-31, 2015*, pages 1902–1908. AAAI Press.

Rosenfeld, A., Azaria, A., Kraus, S., Goldman, C. V., and Tsimhoni, O. (2015b). Adaptive advice in automobile climate control systems. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, pages 543–551.

Rosenfeld, A., Keshet, J., Goldman, C. V., and Kraus, S. (2016). Online prediction of exponential decay time series with human-agent application. In *ECAI*, pages 595–603.

Rosenfeld, A. and Kraus, S. (2016). Providing arguments in discussions on the basis of the prediction of human argumentative behavior. *TiiS*, 6(4):30:1–30:33.

Rosenfeld, A., Taylor, M. E., and Kraus, S. (2017a). Leveraging human knowledge in tabular reinforcement learning: A human subjects study. Under review.

Rosenfeld, A., Taylor, M. E., and Kraus, S. (2017b). Speeding up tabular reinforcement learning using state-action similarities. In *AAMAS*, pages 1722–1724.

Sequeira, P., Melo, F. S., and Paiva, A. (2013). An associative state-space metric for learning in factored mdps. In *Portuguese Conference on Artificial Intelligence*, pages 163–174. Springer.

Stone, P., Kuhlmann, G., Taylor, M. E., and Liu, Y. (2006). Keepaway soccer: From machine learning testbed to benchmark. In Noda, I., Jacoff, A., Bredenfeld, A., and Takahashi, Y., editors, *RoboCup-2005: Robot Soccer World Cup IX*, volume 4020, pages 93–105. Springer Verlag, Berlin.

Suay, H. B., Brys, T., Taylor, M. E., and Chernova, S. (2016). Learning from demonstration for shaping through inverse reinforcement learning. In *AAMAS*, pages 429–437.

Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT press.

Szepesvári, C. and Littman, M. L. (1999). A unified analysis of value-function-based reinforcement-learning algorithms. *Neural computation*, 11(8):2017–2060.

Tamassia, M., Zambetta, F., Raffe, W., Mueller, F., and Li, X. (2016). Dynamic choice of state abstraction in q-learning. In *ECAI*.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, University of Cambridge England.

Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. (2016). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

Zinkevich, M. and Balch, T. (2001). Symmetry in markov decision processes and its implications for single agent and multi agent learning. In *ICML*.

Appendix A Appendix - Post experiment subjective evaluation questionnaire

1. Your id number: _____
2. How clear the task purpose was?

Not clear at all	1	2	3	4	5	6	7	8	9	10	Very clear

3. How familiar are you with the AI field?

Not familiar at all	1	2	3	4	5	6	7	8	9	10	Expert

4. How familiar are you with Reinforcement Learning?

Not familiar at all	1	2	3	4	5	6	7	8	9	10	Expert

5. What was the idea you tried to implement? [free text answer]
6. How successfully have you implemented your idea?

Not managed to implement my idea	1	2	3	4	5	6	7	8	9	10	Very successfully

7. In case you got 15 minutes more to work on your idea, approximate how better did you do with implementing your idea?

The same	1	2	3	4	5	6	7	8	9	10	A lot better

Not appropriate at all	1	2	3	4	5	6	7	8	9	10	Very appropriate

8. How many different ideas have you tried / thought about before getting to the idea you implemented? [numeric answer]
9. To what extent the speedup method you used was appropriate for the task you required to complete?
10. To what extent the performance of the idea you implemented were better than the Basic Q Learning agent performance, in terms of learning rate?

Less good	1	2	3	4	5	6	7	8	9	10	Much better

11. To what extent you believe the time you invest in developing your idea was worthwhile?

Not worthwhile at all	1	2	3	4	5	6	7	8	9	10	Very worthwhile

12. To what extent you managed to implement your knowledge about how to solve the task by using the speedup method you used?

Not managed at all	1	2	3	4	5	6	7	8	9	10	Very successfully

13. What was the most complicated part on the task? [free text answer]