WE NEED A SUMMARY!

1 Experimental Testbed and Platform

Our experimental testbed, while still under development, already includes several two-agent grid games. These games are designed to vary the level of coordination required, while at the same time allowing agents to defend against uncooperative partners.

A grid game is a game played by two agents on a grid, in which each agent has a goal. See, for example, Figure 1a, which is a 3x5 grid in which the two agents' initial positions are one another's goals: Orange begins in position (1,2), Blue's goal; and Blue begins in position (5,2), Orange's goal. We refer to grid positions using x-y coordinates, with $(1, 1)$ as the bottom left position.

One grid game match proceeds in rounds, and each round consists of multiple turns. On each turn, the agents choose one of five actions (north, south, east, west, or wait), which are then executed simultaneously. In the most basic setup, agents transition deterministically, and there is no tie-breaking when two agents collide.[1] Instead, if their chosen actions would result in a collision with one another, neither agent moves. A round ends when either (or both) players move into their goal, or when a maximum number of turns has been taken.

As mentioned above, our grid games are specifically designed to prevent the agents from reaching their goals without coordinating their behavior. Consequently, one approach is for an agent to cooperate blindly with its opponent by simply moving out of the opponent's way, and hoping the opponent then waits for the agent to catch up. However, such strategies can be exploited by uncooperative ones that proceed directly to the goal as soon as their path is unobstructed.

To distinguish "unsafe" from "safe" cooperation, we devised a new categorization for strategies in our grid games. Specifically, we call strategies that allow for cooperation, while at the same time maintain a defensive position in the event that the other agent is uncooperative, **cooperative defensive** strategies. More formally, an agent's strategy is **cooperative** (C) if it is one that allows both it and its opponent to reach their goals, while an agent's strategy is **defensive** (D) if its opponent does not have a strategy that allows it to reach its goal strictly first. A cooperative defensive (CD) strategy is both cooperative and defensive.

We now proceed to describe a sample set of grid games, and equilibria comprised of CD strategies (when they exist), to illustrate the kinds of interactions we plan to study. Our first example, Hallway, is depicted in Figure 1a. This game is one in which the agents can choose to coordinate, for example, if both agents agree upon a joint strategy where one agent moves along the top row and the other along the bottom, without interfering with one another. But, an agent could choose to "defect" from this joint strategy, by proceeding straight to its goal. There are CD strategies, however, which defend against such non-cooperative behavior.

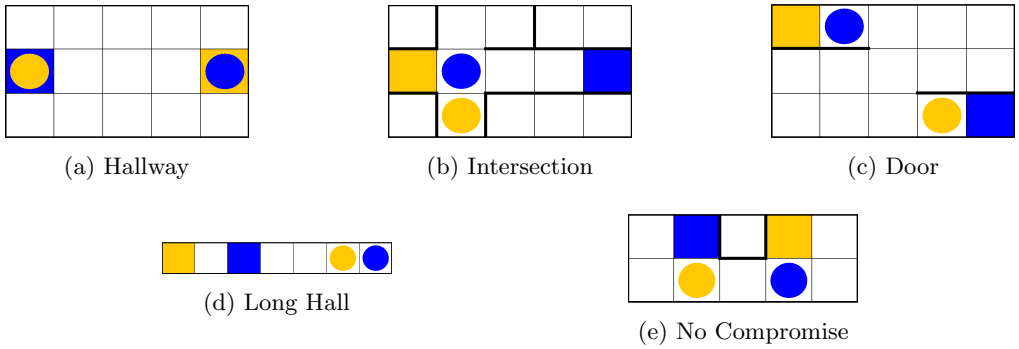
For example, if Orange moves south initially to $(1, 3)$ and Blue moves west to $(4, 2)$, Orange might choose to return and remain on its goal until Blue retreats to $(4, 3)$ or $(4, 1)$, at which point the players are equidistant from their goals, and both can reach them safely. This joint strategy is an equilibrium comprised of CD strategies, since Orange and Blue both remain in positions where they have the ability to block their opponents until they both have unobstructed equidistant paths to their respective goals.

The grid in Figure 1b (Intersection) requires Blue to defend against the possibility of Orange behaving uncooperatively, which it can achieve by squatting on the orange goal. Orange can then move to $(3, 1)$ where both agents are equidistant from their goals. Therefore, this game also has an equilibrium comprised of CD strategies for both players.

This equilibrium is not the shortest path, however. Purely cooperative agents in this game could adopt a joint strategy in which Blue moves east, while Orange waits a single step, before both agents proceed into their goals. This strategy profile is not defensive for Blue though, because it does not have the opportunity to observe if Orange will cooperate (wait) or defect (go north), and therefore cannot defend itself if Orange decides to head straight toward its goal.

Figure 1c (Door) is a grid that requires coordination to navigate through the narrow center space at $(3, 2)$. Any equilibrium comprised of CD strategies for this grid must be asymmetric, because it requires one agent to cede to the other agent the center cell. For example, if Orange chooses to cede that cell, it should step west into $(2, 3)$ while Blue steps south into $(3, 2)$. Then, Orange needs to step east back into $(3, 3)$ to prevent Blue from marching straight into its goal. Only when Blue agrees to step aside to $(2, 2)$ will they

[1] It is a simple matter to vary these rules within our infrastructure, as future experimental design might dictate.



(a) Hallway (b) Intersection (c) Door

(d) Long Hall (e) No Compromise

Figure 1: Five example grid games.

both be equidistant from their respective goals and in position to cooperate. This intricate pattern of first giving way to the opponent, and then forcing them to step around later represents an equilibrium comprised of CD strategies, since both agents are able to prohibit their opponent from reaching the goal first, but still leaves open the possibility for them both to reach their goals, cooperatively.

In the grid in Figure 1d (Long hall), Blue begins one step closer to its goal than Orange does. However, Orange can squat on the blue goal until Blue chooses to cooperate by taking one step back. If Orange can predict when Blue steps back, then Orange can take one step closer to its goal while Blue steps further away, in which case only Orange would reach its goal. The strategy that minimizes the risk to either agent requires that Blue wait one turn initially, while Orange moves toward its goal. These two strategies comprise a CD equilibrium.

Our last grid, shown in Figure 1e (No compromise), requires not only cooperation, but both agents must also exhibit trust for one another, or both agents cannot arrive at their goals at the same time. For example, Orange may sit on Blue's goal so that Blue can move to $(1, 2)$. Then, Blue must wait two turns before both agents are equidistant from the goals. If Blue defects and moves south into $(1, 1)$ while Orange moves south into $(2, 2)$, Orange still has the opportunity to go back up north to block Blue from reaching its goal. However, if Blue moves south into $(1, 1)$ when Orange steps east into $(3, 2)$, Blue will arrive at its goal sooner. Therefore, a trust spanning multiple rounds is required for the agents to effectively cooperate in this game.

No equilibrium in CD strategies exists for No Compromise. The game is like Door in that only one player can go move through the middle cell at a time. Unlike Door, however, it is not possible for the agents to simultaneously maintain a defensive position and to signal cooperation, because any cooperative move leads to an asymmetric situation in which the agents are no longer equidistant from their goals. As a result, after one agent cooperates, there is always an incentive for the other agent to defect, and there is nothing the cooperative agent can do to defend itself. Note however, that if Orange sits on the blue goal while Blue walks to $(1, 1)$ and then Blue cooperates by waiting for Orange to walk to $(1, 3)$, Blue's policy is CD. Still, Orange cannot respond in kind with a CD strategy; the aforementioned strategy is C.

Taking these five grid games as an initial testbed, we performed two pilot studies: the first involved simulations of artificial agents playing against one another; the second pitted humans against other humans on Mechanical Turk. We report on the results of these preliminary studies in the next two sections. The ultimate goal of this project, of course, is to design artificial agents that play well with humans. We propose to achieve this goal by building agents that infer peoples' utility functions, and then learn to collaborate appropriately based on these inferences.

Simulation Experiments We carried out a set of simulation experiments with Q-learning in the grid games presented. For each grid game, we conducted 50 independent runs in which two Q-learners faced off. The agents' value functions were optimistically initialized with a value of 40 for all states and they used Boltzmann exploration with a temperature of 0.5. The discount factor was set to 0.9, and the learning rate to 0.01. To ensure that the state space was adequately explored, any state that is reachable from the initial state had some probability of being selected as the starting position for a round. Once either agent reached its goal (or 100 moves were taken), the round was terminated. There were no step costs beyond discounting,

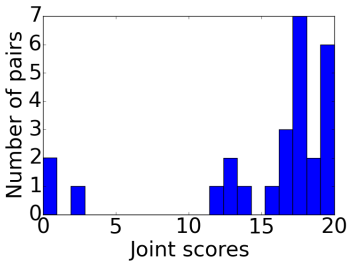


Figure 2: Histograms of joint scores achieved by pairs with team treatment

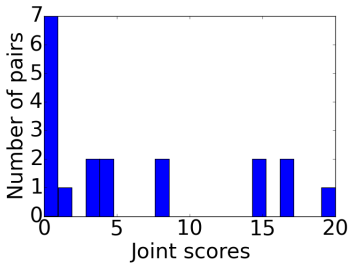


Figure 3: Histograms of joint scores achieved by pairs with individual treatment

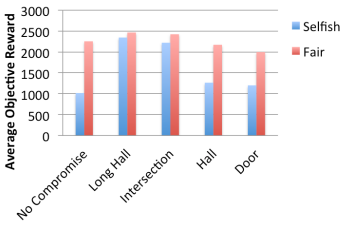


Figure 4: Average score of Q-learners in self play (30,000 rounds)

and rewards were 50 for reaching a goal.

We denote the outcome of a round using a pair of letters, where **G** means the agent reached the goal and **N** means the agent did not reach the goal. The first letter in the pair represents the agent's own outcome and the second represents its opponent's outcome. For example, **GN** is used to denote that the agent reaches its goal while its opponent does not.

As two Q-learning algorithms are not guaranteed to converge in self-play, we arbitrarily stopped the learning after 30,000 rounds, and checked the strategies learned. In spite of Q-learning not explicitly seeking outcomes with high social welfare, it very reliably identified cooperative strategies leading to **GG** outcomes.

Only the No Compromise game posed a challenge to the Q-learners. There, they tend to thrash about, finding a pair of strategies that work well together only to eventually discover that one of the agents has an opportunity to defect. The defection is destabilizing, so a new search begins for strategies that work well together. This result is not altogether unsurprising, because No Compromise is the only game in our testbed that does not possess a pair of CD strategies that constitute a Nash equilibrium.

In a second set of Q-learning experiments, we examined the impact of endowing agents with **social preferences**. That is, their rewards no longer depend solely on their own successes and failures—they depend on those of other agents in the environment as well.

To make this idea more precise, we separate **objective** and **subjective** reward. Objective reward is the usual reward signal provided to the agent from the environment, and by which behavior is judged. Standard reinforcement-learning agents, such as a Q-learning agent, seek to optimize their own objective reward signal—we call this preference the "selfish" preference because these agents are only concerned with their own outcomes.

In some environments, however, it useful to distinguish this objective reward from subjective reward—the quantity the agent *believes* it should be optimizing. Previous work has shown that optimizing subjective reward can sometimes lead agents to be more effective in their acquisition of objective reward [**?**]. (Indeed, we reach this same conclusion in our experiments.)

Considering the four different outcomes in these games—**GG**, **GN**, **NG**, **NN**—there are 75 different possible preference orderings (allowing for ties). The selfish ordering that ignores the opponent's outcome is one of these: $\mathbf{GG} \sim \mathbf{GN} \succ \mathbf{NG} \sim \mathbf{NN}$. Nine of the 75 orderings are consistent with the selfish ordering, strictly preferring **Gx** to **Nx** for all **x**.

Of particular interest is the **fair** preference, which we define as the objective reward of the agent minus 25% of the difference between its own and the opponent's objective rewards: $r_s = r_a - 0.25\,|r_a - r_o|$, where r_s is the agent's subjective reward, r_a is the agent's objective reward, and r_o is the opponent's objective reward.[2] By incorporating this fairness term into the agent's subjective reward, it strictly prefers the following ordering: $\mathbf{GG} \succ \mathbf{GN} \succ \mathbf{NN} \succ \mathbf{NG}$. That is, the agent prefers making it to its goal as opposed to not, but it additionally prefers that the opponent only get to its goal if the agent itself does as well. To say it another way, a fair agent wants its opponent to win with it or lose with it.

Figure 4 shows the result of the selfish and fair agents playing against others of the same type in each of our test grid games. Of the nine orderings, only three (all variations of the fair preference ordering) achieve

[2]Other percentages would achieve the same result.

consistent cooperation in self play across all five grid games. Consequently, fair agents obtain higher total (objective) rewards than others. We also ran all nine preference orderings against one another. The average scores (across both players) in games involving a fair agent tend to be higher than the average scores in games not involving a fair agent.

We also analyzed the types of strategies learned by fair and selfish Q-learners after multiple simulations of various configurations. Interestingly, we found that Q-learners with fair preferences tend to find CD strategies more often, especially when paired with selfish agents.

In summary, Q-learners naturally learn to cooperate in the grid games studied, discovering equilibria comprised of CD strategies when they exist. Cooperation can be induced in other games by manipulating the Q-learners' subjective rewards to incorporate the success of others.

In the next section, we describe analogous experiments conducted with humans playing grid games. In those experiments as well, we were able to manipulate the rewards to favor fairness, and doing so induced more cooperation than otherwise.

Turk Experiments We ran two studies in which human subjects played grid games. We recruited participants on Mechanical Turk to play the Hallway game against another Turker.

Each participant began with an instruction phase that used a series of practice grids to teach them the rules of the game: arrow keys to move north, south, east, or west in the grid; spacebar to wait; both agents move simultaneously; when two agents try to enter the same square, their moves fail; and the round ends when either agent reaches a goal. Example grids demonstrated outcomes in which both agents reached a goal, and outcomes in which one did and the other did not. All transitions (including transitions that did not involve changing location) were animated so that participants could see that their actions registered.

After the instruction phase, the participants were paired up. Each pair played a match consisting of 20 rounds, which ended when either or both agents reached a goal, or when they had taken 30 actions without either reaching a goal.

We designed two different treatments, one (the "individual" treatment) was a control, and the other (the "team" treatment) was intended to inspire team reasoning. In the individual treatment, a participant received a bonus when they reached their goal, regardless of whether the other participant also reached their goal. In the team treatment, a participant received a bonus only if they reached their goal at the same time as the other member of their pair.

We recruited 40 participants to form 20 pairs in the individual treatment; in the team treatment, we recruited 50 participants. All participants received \$2.00 as a base payment, and \$0.10 bonuses for goals scored according to the rules of the treatment.

Figure 2 and figure 3 are two histograms illustrating the distribution of joint scores achieved by the pairs in the two treatments. The team treatment, which in the Hallway game requires team reasoning (because the players only succeed when they simultaneously reach their goals), produced significantly more successful teams. In particular, 19 teams (76%) achieved a score at or above 15 points in the team treatment, but only 3 achieved a similar score in the individual treatment.

Table 1 compares the outcomes of matches in the two different treatments under an intuitive classification scheme, which we based on the behavior of the participants in the final two rounds after 18 prior rounds of learning. We are planning much more extensive experimentation with human participants on grid games, after which we hope to interpret some of these classifications (e.g., trust, CD, etc.) as types of norms.

Compete means, in the final round, only one player reaches their goal, and in the last two rounds, the players collide at least once. *Trust* means both players score in the last round, but only when one player did not take advantage of an opportunity to defect. *Surrender* means that in the final two

	Individual Reasoning	Team Reasoning
Compete	32%	0%
Trust	21%	76%
Surrender	21%	8%
CD	11%	16%
Alternate	11%	0%
Stalemate	5%	0%

Table 1: A comparison between the strategies learned by pairs in the two treatments. These percentages represent 19 games in the individual reasoning experiment, and 25 games in the team reasoning experiment.

rounds, only one player reaches the goal and does
so without any collisions. *CD* (*Cooperatively Defend*) means that both players score in the final round, but each does so in a way that prevents the other player from reaching the goal alone. In *Alternate* matches, both players reach their goal alone exactly once in the last two rounds, without any collisions. Finally, in *Stalemate* matches, neither player reaches their goal in the final round.

The treatment impacts the types of behavior we observe. The incidence of competitive behavior drops dramatically, while trust increases threefold. Since trust increases, surrendering becomes less necessary, and consequently much rarer. Also of interest, the incidence of CD behavior doubles. Overall, 92% of participants exhibit cooperative behavior (either trust or CD) in the team treatment, compared to only 33% in the individual treatment.

The results involving human subjects playing grid games are consistent with the results obtained in our simulations of Q-learners playing grid games. In both sets of experiments, more cooperation was achieved when the treatment incorporated social preferences. While the subjective utility functions of artificial agents are within our control, so that we can perhaps lead artificial agents towards collaborative behavior, the subjective utility functions of humans are not. Nonetheless, behavioral economists often infer approximations of utility functions from experimental data by assuming some underlying structural form, and then estimating the relevant parameters [**?**, **?**]. Likewise, one of the primary uses of our Turk experimental framework is to generate trace data of humans playing grid games and learning norms (such as trust, CD, etc.), so that we can then proceed to infer utility functions and relate them back to the norms they produce.

Next we present our proposed norm representation. We then present an algorithm for learning norms from trace data, human- or machine-generated. We also present an interactive algorithm, in which agents can learn while at the same time establishing a norm. We then proceed to demonstrate the viability of our algorithms on human-human (Turk) and agent-agent (simulation) data. A key step in our plan for future work is to integrate our learning algorithms with the Turk platform, so that we can study interactive human and agent learning.

2 Norm Representation

Our norm representation and learning algorithms are best explained using the formalisms of Markov decision processes and stochastic games. We start by providing the necessary background.

A **Markov Decision Process** (MDP) is a model of a single-agent decision making problem, defined by the tuple (S, A, T, R), where S is the set of states in the world; A is the set of actions that the agent can take; $T(s' \mid s, a)$ defines the transition dynamics: the probability the environment transitions to state $s' \in S$ after the agent takes action $a \in A$ in state $s \in S$; and $R(s, a, s')$ is the reward function, which returns the reward the agent receives when environment transitions to state s' after the agent takes action a in state s.

The goal of planning or learning in an MDP is to find a policy $\pi : S \to A$ (a mapping from states to actions) that maximizes the expected future discounted reward under that policy: $E^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})\right]$, where $\gamma \in [0, 1]$ is a discount factor specifying how much immediate rewards are favored over distant rewards.

To find an optimal policy, many algorithms compute the optimal state ($V^*(s)$) and state-action ($Q^*(s, a)$) value functions that specify the expected future discount reward under the optimal policy from each state, and from taking an action in a state and then following the optimal policy respectively. Given these functions, the optimal policy is derived by taking an action in each state with the maximum Q-value: $\pi(s) \in \arg\max_{a \in A} Q(s, a)$ [**?**].

Inverse reinforcement learning (IRL) is a class of a *learning from demonstration* (LfD) problem. An LfD problem is a problem in which an agent is presented demonstrations of behaviors and must learn a policy of behavior that closely reproduces the observed behavior. In IRL, an agent learns this policy indirectly by learning a reward function for the MDP that describes an environment that would motivate an agent to behave in a way that is consistent with the observed behavior. A strength of the IRL formulation is that simple reward functions can often capture complex behavior. As a result, a learned reward function can often produce behavior that accurately generalizes the observed behavior to many states, including states not observed in the input demonstrations.

Although there are multiple IRL formalizations and approaches, they all take as input an MDP together with a **family** of a reward functions R_Θ that is defined by some parameter space Θ, and a dataset D

of trajectories (where a trajectory is a finite sequence of state-action pairs: $\langle (s_1, a_1), ..., (s_n, a_n) \rangle$). The algorithms then output a specific parameterized reward function $R_\theta \in R_\Theta$, which induces a policy that is consistent with the input trajectories. Different IRL algorithms frame the objective function for policy consistency differently. One common approach is to treat the search problem as a probabilistic inference problem [**?**, **?**, **?**, **?**].

The stochastic games formalism can be viewed as an extension of MDPs to the multi-agent case [**?**]. A **stochastic game** is defined by the tuple (I, S, A^I, T, R^I), where I is an index set of agents in the environment; S is the set of states of the environment; A^I is set of actions for each of the agents with A^i denoting the action set for agent $i \in I$; $T(s' \mid s, j)$ is the transition dynamics specifying the probability of the environment transitioning to state $s' \in S$ when the *joint action* $j \in \times_i A^i$ of all agents is taken in state $s \in S$; and R^I is a set of reward functions for each agent with $R^i(s, j, s')$ denoting the the reward received by agent $i \in I$ when the environment transitions to state $s' \in S$ after the agents take joint action $j \in \times_i A^i$ in state $s \in S$.

The goal in a stochastic game is to find a joint strategy that satisfies some solution concept. Different solution concepts for stochastic games have been explored in the past including minimax, Nash, correlated, and CoCo equilibria [**?**, **?**, **?**, **?**]. There are problems with these approaches, however. First, the resulting planners must solve for game-theoretic equilibria in an inner loop, a problem which in the case of Nash equilibrium, for example, is notorious for its computational intractability [**?**]. Second, in the general case of non-constant sum games, none of the planners that make reasonable assumptions about agent behavior yield unique joint plans, and furthermore, none has solved the ensuing equilibrium selection problem suitably.

Immediately generalizing from the case of MDPs, the goal of inverse reinforcement learning in stochastic games is to learn a set of reward functions for the stochastic game that describe an environment that would motivate the agents to behave in a way that is consistent with the observed behavior under some solution concept such as a Nash equilibrium [**?**]. This problem, however, is exceedingly difficult, because the planning that is necessary in the inner loop of an IRL algorithm is subject to the challenges identified by the equilibrium planners mentioned above.

In this project, we plan to develop IRL algorithms that can explain norm-adhering behavior in a stochastic game. In our preliminary studies, we assume players' rewards are known, and our goal is to recover a **social reward function** that combines these known rewards in such a way as to capture a social norm expressed in the joint play of the agents (when one exists).

Like the players' reward functions, a social reward function operates on states, joint actions, and next states: $R^S : S \times_i A^i \times S$. In our initial model, we assume that this social reward function can be written as linear combination of a **team function**, which represents team goals, and a family of what we call **bias functions** (B_Θ) defined by some parameter space Θ. The team function takes as input a multi-agent reward function (R^I), and returns a single numeric value for any state-joint action-next state triple that represents a team goal. One example of such a function is total welfare (i.e., the sum of all agent rewards): $\mathcal{T}(R^I, s, j, s') = \sum_i R^i(s, j, s')$. The bias function family is similar in nature to a reward function family that would be input to a classic IRL algorithm, but operates on state-joint action-next state triples, thereby encoding a bias that motivates norm-adhering behavior in games. In addition to the parameters of the bias function, the social reward function may also include a parameter to trade-off the team rewards against the biases.

Recall our working definition of a social norm as a behavioral instruction that members of the community expect one another to follow. This definition suggests that norms could potentially be learned in a supervised fashion simply by learning a function that maps states to joint actions directly, rather than capturing norms in a social reward function that motivates joint actions. However, as an environment and a target policy become more complex, directly learning the policy becomes more challenging, and generalizing from it likely less successful. As with standard IRL, the advantage of learning a social reward function instead is that simple reward functions can often induce complex behaviors *across states*. For example, consider a seemingly straightforward norm that two agents approaching one another each stay to their right. If the action space is over low-level controls (e.g., rotations and small movements) and the policy is context dependent (e.g., navigating obstacles to get to the right side), implementing all the joint action rules necessary to make the agents pass on the right could be difficult to specify (and hence, learn). However, a social reward function can simply define a bonus for when the agents move past each other on the right and the planning algorithm

Algorithm 1 Batch_Learning$((I, S, A^I, T, R^I), \gamma, D, \mathcal{T}, B_\Theta)$

Require: stochastic game (I, S, A^I, T, R^I); discount factor γ; multi-agent norm-following demonstrations D; team function $\mathcal{T}$; and parameterized bias function family B_Θ.

$A^M := \times_i A^I$ ▷ Joint action set

$R^M(s, a, s') := \mathcal{T}(R^I, s, a, s')$ ▷ Team reward function

$R^S_\Theta(s, a, s') := R^M(s, a, s') + B_\Theta(s, a, s')$ ▷ Social reward function family

$R^S_\theta := \text{IRL}((S, A^M, T, R^S_\Theta), \gamma, D)$

return R^S_θ ▷ Learned social reward function

does the rest.

To facilitate social reward functions that generalize, the bias function family must operate on a set of useful features. As part of this project, we plan to investigate whether existing state-of-the-art feature selection methods for RL [**?**, **?**, **?**, **?**] work well in our setting, and then to work towards developing new methods to the extent necessary.

3 Norm Learning

Given our representation of norms by a social reward function that is parameterized by a family of bias functions, our approach to learning norms is simply to optimize those parameters to best match observed behavior. We consider two forms of norm learning. The first is **batch** learning, in which an agent learns offline from batches of demonstrations of other agents conforming to a norm. The second is **interactive** learning in which an agent learns about the behavior of a set of unknown agents, while at the same time interacting with them and establishing norms.

Batch Learning The batch learning setting is similar to the standard IRL problem in that the algorithm is given a batch of demonstrations and a family of reward functions over which to search. In our case, however, the batch data consists of trajectories of states and joint actions in a stochastic game, and the reward function family is a social reward function family. To learn the parameterization of the social reward function family that best captures the behavior exhibited in the demonstrations, we convert the stochastic game into an MDP, and then use IRL to learn a parameterization of the reward function family in this MDP.

Pseudocode for the batch learning algorithm is shown in Algorithm 1. The algorithm takes as input a stochastic game; a discount factor; a set of multi-agent demonstrations (which perhaps shows adherence to some norm); a team function; and a family of bias functions defined by some parameter space Θ. Since the demonstrations are a set of trajectories of state-joint action pairs, specifying the behavior of all the agents in the environment, the first step of the batch learning algorithm is to transform the stochastic game into an MDP. For the most part, this transformation is straightforward: the states are the same, the MDP action set is the space of joint actions, and the MDP transition dynamics still operate on joint actions (which is the action set in the MDP). A family of social reward functions is then defined as a linear combination of the input team function (which depends on the stochastic game's reward functions), and the family of bias functions.[3] After learning a social reward function, any single agent can then behave by selecting a joint action from the policy that results from it, and then selecting their individual action from the joint. That is, agent i takes action $j^i \in A^i$, where $j \in \arg\max_j Q_{R^S_\theta}(s, j)$.

Although any off-the-shelf IRL algorithm would be applicable here, in practice we use **receding horizon IRL** (RHIRL) [**?**]. IRL algorithms are typically computationally demanding because they require planning in their inner loops. RHIRL addresses this limitation by replacing the usual infinite-horizon policy with a receding horizon controller (RHC) that only plans out to some finite horizon from any given state, thereby bounding computation time. An RHC tends to steer an IRL algorithm towards a reward function with shaping values: short-term rewards that guide the agent in the right direction without having to plan too far ahead. Since these are precisely the kinds of values we want our bias functions to capture, RHIRL allows

[3] A parameter controlling the linear combination of the team and bias function may be optimized as well.

Algorithm 2 Interactive_Learning$((I, S, A^I, T, R^I), k, \gamma, \mathcal{T}, B_\Theta)$

Require: stochastic game (I, S, A^I, T, R^I); agent index k; discount factor γ; team function $\mathcal{T}$; and parameterized bias function family B_Θ.
initialize policy π^k arbitrarily
$A^M := \times_i A^I$ ▷ Joint action set
$D := \{\}$
for each round of play **do**
play round following π^k
append new trajectory of play to D
$R^S_\theta :=$ Batch_Learning$((I, S, A^I, T, R^I), \gamma, D, \mathcal{T}, B_\Theta)$
$\forall s \in S$, $\pi^k(s) := a^k \in A^k$ where $a \in \arg\max_a Q_{R^S_\theta}(s, a)$
end for

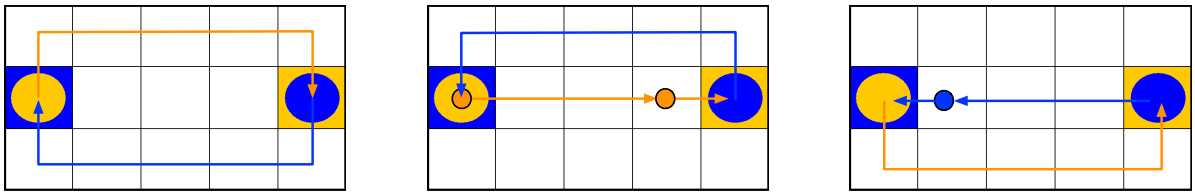
us to plan with a short horizon, thereby saving on computation time, which enables interactive learning, described next.

Interactive Learning In the interactive norm-learning setting, an agent plays with a set of unknown agents, with whom it can potentially establish new norms. Pseudocode for our interactive norm-learning algorithm is shown in Algorithm 2. The algorithm takes as input the same arguments as the batch learning algorithm, except it includes an index specifying which agent in the stochastic game the learner is, and it does not include the batch of demonstrations. The agent begins by initializing its policy arbitrarily.[4] The first round of interaction produces a trajectory of joint behavior that the learner can now learn from. It does so by running our batch learning algorithm on the single trajectory of experience acquired thus far. It then proceeds to follow the learned policy in the next round. After the second round completes, the agent now has two trajectories on which it can run batch learning to update its social reward function and its behavior. This process then repeats for all rounds of interaction. Note that updating the policy after each round requires computing the Q-values under the newly learned social reward function. However, in practice, this computation can be performed lazily for each state the agent observes in the next interaction.

Norm Learning Results We now present preliminary results demonstrating that both our batch and interactive norm learning algorithms can produce reasonable results in grid games. For batch learning, we hard code examples of agents following different norms, and show that our algorithm is able to recover the appropriate norm when it is trained on demonstrations of behavior following that norm. We also show that the learned norm can generalize to sensible behavior in other situations by testing the agents in similar grid games after learning. For interactive learning, we show that two agents playing together can very quickly establish a norm, and that restarting learning from scratch can result in the agents establishing different norms.

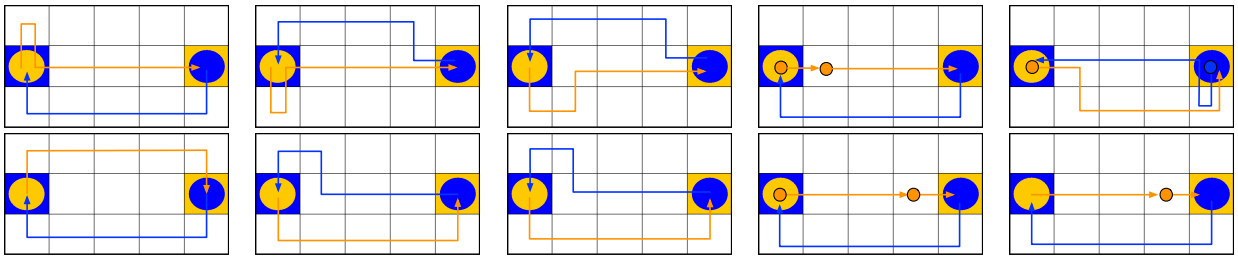
For both sets of experiments, we examine behavior in Hallway. The social reward function consists of a total welfare team reward function and a linear family of bias functions. The bias functions operate on 50 state-joint action binary features. The first 25 features are an indicator variable for selecting one of the 25 joint actions when the agents are in *conflict* and the latter 25 features are indicator functions for selecting one of the 25 joint actions for when the agents are not a conflict. We define the agents to be in conflict when they are in the same row of the grid and each is closer to their opponent's goal than they are to their own goal. Additionally, when one of the agents is one step away from their goal, none of the features activate, thereby preventing norm preferences from favoring premature termination of the joint task. Since these features directly encode preferences for joint actions rather than higher-level goals, when training we use RHIRL with a horizon of 1. We do not expect this set of features to be complete or capable of capturing all the possible norms that might be established in a grid game. So an important area for future work is to identify expressive features that can also generalize well. Nevertheless, the features used here provide a proof of concept for our approach.

[4] In practice, the policy can be initialized to the agent's role in the joint policy that follows from the team function alone.



(a) (b) (c)

Figure 5: The three different norms the batch learning algorithm trained on in Hallway. Colored arrows indicate the path of the player of the same color. A small circle indicates a wait action.



(a) (b) (c) (d) (e)

Figure 6: The learning results from each of the 5 interactive matches (a-e). The top image for a match is the first round of interaction of the agents. The bottom image for a match is the learned behavior. For matches b-e, the last move of an agent that reaches their goal sooner may be to wait for two steps, or to move north and/or south and back again; only one example is illustrated.

We tested our batch learning algorithm on datasets in which agents followed three different norms. The first norm (Figure 5a) shows the orange player going north and then along the top row of the grid while the blue player goes south and then along the bottom row. The second norm (Figure 5b) shows the blue player going north and along the top row while the orange player waits, then goes east, and then waits again for blue to catch up so they can enter their goals together. The third norm (Figure 5c) has the orange player go south and along the bottow row while the blue player immediately goes west toward its goal and then waits two steps.

For each of these three norms we generated five demonstrations, two of which described the norm exactly, and the other three of which contained slight deviations from the norm (for example, moving back to the center row before reaching the end). We then separately trained our batch algorithm on each set of demonstrations. In all cases, the learned social reward function motivated behavior that consistently replicated the observed norm on which it was trained. Additionally, we planned using the learned social reward function in a larger 7x3 hallway grid game, and obtained the same norm behavior.

In our interactive norm learning experiment, we played two interactive learning agents against one another. The agents played five matches, each one consisting of five rounds. Each match began fresh, without any knowledge of previous matches, to see whether different norms could emerge. The social reward function was initialized to the total welfare joint policy. In each match, the agents very quickly (typically after the first round) converged on a norm that persisted for the rest of the match. However, in each match, a different norm was learned, showing that agents employing our algorithm quickly adapt to the particular shared experience they have with other agents.

Figure 6 shows the results of learning in each of the five matches (with the top image showing the behavior in the first round and the bottom image showing the learned behavior). While different normative behavior was learned in different matches, once learned, the learned behavior was consistently employed in all subsequent rounds. Of particular interest are the results in the fifth match (Figure 6e). In the first round of this match, both players wait (a suboptimal joint action), and Blue then moves in an odd looping motion. Nevertheless, a norm based on the actions that ultimately worked is learned from the experience, and the

	Human vs Human	Batch Norm IRL
Trust	76%	72%
CD	16%	4%
Stalemate	0%	24%
Surrender	8%	0%

(a) The percentage classifications of game outcomes for the Amazon Turk experiments and the batch norm learning algorithm trained on the Turk experimental data. The total number of games was 25.

	Trust	CD	Stalemate	Surrender
Trust	60%	0%	16%	0%
CD	8%	4%	4%	0%
Stalemate	0%	0%	0%	0%
Surrender	4%	0%	4%	0%

(b) The confusion matrix showing how games were classified in the human experiments (row) vs. the batch norm learning experiments (column).

agents do better after that.

Batch Norm Learning on Human Data Finally, we trained our batch norm learning algorithm using data collected in our Amazon Turk team treatment experiments. The final 10 trajectories of each pair of human players was used to train a batch learner that made decisions for both agents jointly. Table 2a shows the classifications of the strategies learned in the 25 games, while Table 2b illustrates the learner's ability to create joint strategies that are classified in the same way as the human strategies. Like the humans, we can see that the algorithm learned to be trusting in many cases, and to cooperate defensively in a few. But the learner was sometimes unsuccessful at learning a strategy that led either agent to the goal (even when the humans had managed to reach the goals), resulting in a fair number of games ending in Stalemate. Overall, 64% of the policies learned exactly matched the classification of the human data that was used in training. Undoubtedly, feature selection plays a major role here, and will be the subject of future research.

4 Plan

The three year timeline for our proposed project is as follows:

5 Deliverables

6 Evaluation

7 Broader Impacts

****Amy: we were criticized on our failure to really relate to IoT; i.e., this isn't convincing ****

The many devices in our world that collect and share data among the ever-growing "Internet of Things" cannot yet intelligently coordinate their actions to carry out complex tasks without extensive human programming and configuration. As this collection of devices grows and begins to include more traditional robotic actors and more sophisticated software agents, the need for these entities to not just communicate but work collaboratively is only going to increase. We will want and need them to work together with and without us to accomplish everyday tasks and help us make better decisions.

The domains that would be positively impacted by machines that can collaborate effectively with humans is endless, such as robotic sous chefs and gerontechnological support for aging in place. In particular, the results of this project synergize with the main efforts of Brown University's Humanity Centered Robotics Initiative (HCRI), of which co-PI Littman is co-director. Specifically, within HCRI there are ongoing efforts to design robotic systems that support independent living tasks. For an elderly person to trust and collaborate on tasks with a machine effectively, the machine must act in a manner that the elderly person expects. This project lies the foundation for these important applications.

In an effort to increase diversity in computer science, we have pre-selected two graduate students to participate in this project—one woman, and the other Hispanic. All students (both graduate and undergraduate) who join our team will benefit from collaborating with the cognitive psychologists on our team.

Specifically, they will strengthen their understanding of a number of fields, all of which are critical to the development of artificial agents that collaborate effectively with humans: e.g., behavioral economics, cognitive psychology, reinforcement learning, and software engineering.

"One of the things that [President Obama] really strongly believe[s] in is that we need to have more girls interested in math, science, and engineering." To that end, we also plan to integrate our work on this project into Artemis, a free summer program run at Brown and directed by PI Greenwald that introduces rising 9th grade girls to computational thinking. For example, we might have the Artemis girls teach a robot to collaborate with them on various tasks, such as finding and navigating to common household objects.

Our deliverables include an open-source publicly accessible toolkit for machine norm learning via reinforcement learning. Further, we will build a database of machine-machine, human-machine, and human-human experimental results on norm-learning problems, which can serve as a benchmark for future researchers to build artifical agents that increasingly achieve human-like behavior. Finally, we expect to publish the results of the proposed research in top-tier archival, conference proceedings and journals with high impact factors, and present our work at both computer science and psychology conferences.

8 Results from Prior NSF Support

Intellectual Merit Greenwald is currently funded by NSF (RI: Small–1217761, 2012–2016, $450k) to build artificial agents that assist humans with decision making in information-rich and time-critical environments like online markets. This work is ongoing, but some preliminary publications include [**?**, **?**, **?**, **?**]. Previously, she was funded to develop "Methods of Empirical Mechanism Design (EMD)" (CCF: Medium–0905234, 2009–2012, $850k, with Mike Wellman). This project expanded the scope of mechanism design beyond the small-scale, stylized, or idealized domains most previous work was limited to. Publications include [**?**, **?**, **?**]. Before that, she received two prior NSF awards, a PECASE CAREER grant, "Computational Social Choice Theory" (IIS: Career–0133689, 2002–2007, $375k), and "Efficient Link Analysis" (IIS: Small–0534586, 2005–2008, $363k), which focused on ranking web pages and other social networks with hierarchical structure.

Littman is a co-PI on "RI: Medium: Collaborative Research: Teaching Computers to Follow Verbal Instructions" (No. 1065195, 9/11–8/14, $704K) and "RI: Small: Collaborative Research: Speeding Up Learning through Modeling the Pragmatics of Training" (No. 1319305, 10/13–9/15, $440k). These proposals developed autonomous agents that extract preferences from people via verbal interaction and reward feedback [**?**, **?**, **?**].

Broader Impact Broader impacts of our past work include undergraduate and graduate student training, design and evaluation of learning modules for outreach programs Learning Exchange and Artemis, and organizing two major computer science conferences (Littman) [**?**]. We also published novel benchmarks for evaluating grounded language learning programs (Littman), and developed a simulation platform for empirical game-theoretic analysis (Greenwald).

Greenwald also had an additional NSF award (EAGER: 1059570, 2010–2012, $90k) which supported the expansion and evaluation of the Artemis Project, a free, summer program in which Brown computer science women teach computer science to rising 9th grade girls. This program has the dual effect of empowering Brown women, while at the same time, exposing girls to computer science. With this money, she successfully expanded Artemis to BU, and ran pilot programs at Columbia and UMBC.

Greenwald also has $5k in funding from the Tides Foundation, through the IgniteCS progam, for a project entitled Bringing Computer Science Education to Providence Public Schools.