Cooperating with Machines: Supplementary Information

Jacob W. Crandall*[1], Mayada Oudah[1], Tennom[1], Fatimah Ishowo-Oloko[1], Sherief Abdallah[2,3], Jean-François Bonnefon[4], Manuel Cebrian[5], Azim Shariff[6], Michael A. Goodrich[7], and Iyad Rahwan*[8]

[1]*Masdar Institute of Science and Technology, Abu Dhabi, UAE*
[2]*British University in Dubai, Dubai, UAE*
[3]*School of Informatics, University of Edinburgh, Edinburgh EH8 9AB, UK*
[4]*Centre National de la Recherche Scientifique, Toulouse, France*
[5]*NICTA, Melbourne, Australia*
[6]*Department of Psychology, University of Oregon, Eugene, OR 97403, USA*
[7]*Computer Science Department, Brigham Young University, Provo, UT 84602, USA*
[8]*The Media Lab, Massachusetts Institute of Technology, Cambridge, MA 02139, USA*

August 24, 2015

Overview of the Supplementary Information

This work reports on years of effort to develop and analyze a machine-learning algorithm that learns to effectively interact with a human partner in repeated interactions in which the human and machine have conflicting, but not fully opposed, interests. Specifically, we seek an algorithm that can learn to establish and maintain mutually cooperative relationships with people as well as or better than people in arbitrary repeated games.

We hasten to acknowledge that the goal of such an algorithm is not cooperation in and of itself. A self-regarding machine (which we assume in this work) should cooperate only if doing so maximizes its payoffs. However, when players repeatedly interact with each other over long periods of time, have conflicting interests, and both desire to maximize their own individual payoffs, cooperation is very often the best policy (or at least close to it). Thus, an effective algorithm must be able to derive cooperative behavior when interacting with other intelligent entities, including humans. Likewise, it should be able to exploit weak partners (when possible) and avoid being exploited by non-cooperative partners.

The algorithm we eventually produced to achieve these feats is called S# (pronounced 'S sharp'). S# interweaves S++ [1, 2] with new algorithmic mechanisms designed to give it the ability to engage in cheap talk (the sending and receiving of costless and non-binding communication signals) with its partner. The algorithm combines and builds upon decades of work by researchers in computer science, economics, and the diverse array of behavioral and social sciences. The algorithm packages these rather diverse and complex findings and algorithmic structures in a way that hides much of the complexity of the problem. S++ is a simple expert algorithm that uses an adapted version of aspiration learning to select among a finite set of

*Correspondence should be addressed to jcrandall@masdar.ac.ae and irahwan@mit.edu

experts, each of which itself uses known mathematics to produce (or learn) a strategy over the entire space of the game. This hierarchical learning structure allows the algorithm to produce rich and complex strategies within short periods of time. Importantly, the hierarchical nature of the algorithm also allows it to express and interpret speech acts at a level conducive to human interaction. Our results show that this is a critical linchpin for a machine to elicit cooperative behavior from a human partner.

Our journey in understanding, deriving, and analyzing S++ and S# is explained in detail in this supplementary information, the contents of which follow the storyline of the main article. The supplementary material is divided into six sections. The first three sections provide technical algorithmic details and analysis, including the problem description, descriptions S++ and S#, and analysis of the performance of these algorithms when paired with other computer algorithms. The last three sections describe user studies in which we paired S++, S#, and several existing machine-learning algorithms with people.

We first provide a summary of each section (or appendix) of the supplementary information. The appendices then follow.

(Appendix A) Problem Formulation and Motivation

The problem domain of learning in two-player repeated stochastic games (scenarios that include, among others, both repeated normal-form and repeated extensive-form games) is extremely challenging, especially when paired with a human partner. This problem domain has many technical challenges that prohibit the straight-forward application of existing machine-learning algorithms. In this section, we formally define and discuss repeated stochastic games, discuss many of the challenges related to learning effective strategies in these games, and provide examples of repeated stochastic games.

(Appendix B) A Comparison of AI Algorithms in Repeated Games

Many algorithms for playing repeated games have been produced over the last several decades. In this section, we compare the performance of a representative set of 19 of these algorithms in Axelrod-style round-robin tournaments and evolutionary studies played across a set of 390 repeated normal-form games. These comparisons demonstrate that S++ [1, 2] is among the top performing AI algorithms for repeated games. It quickly learns effective strategies when paired with a wide range of associates across a wide variety of games. Our analysis also shows that S++ has strong evolutionary properties. These attributes suggest that S++ could possibly be successful in establish cooperative relationships with people in arbitrary scenarios.

(Appendix C) Description of Algorithms

In this section, we review the algorithmic details of S++. We also describe an extension to the algorithm, called S#, that allows it to engage in cheap talk with its partner. This extension defines how cheap talk is generated by the S++ learning algorithm, and how S# uses communication signals from its partner to help it select experts in a more effective manner.

(Appendix D) User Study: Human vs. Machine 1

In this 58-participant user study, which was conducted in 2012-2013, we paired S++, model-based reinforcement learning (MBRL-1), and humans with each other in four repeated normal-form games (Prisoner's Dilemma, Chicken, Chaos, and Shapley's Game), each lasting approximately 50 rounds. The primary purpose of the user study was to determine the ability of S++ to consistently elicit cooperative behavior from

people in repeated interactions. The results of the study showed that S++ was more effective in self play than both people and MBRL-1. The results also showed that S++ was as effective at establishing cooperative relationships with people as people were. However, both S++ and people typically failed to cooperate with human partners in these games. This failure caused us to conduct additional user studies to try to determine what algorithmic characteristics would allow a machine to more effectively establish cooperative relationships with people.

(Appendix E) User Study: Human vs. Machine 2

Given the failure of S++ to consistently establish cooperative relationships with people, we extended the algorithm with the ability to generate and voice cheap talk. We dub the resulting algorithm S#–, as it is an early (but incomplete) version of S#. We then conducted a second (96-participant) study, carried out in two parts, in September 2014. In Part 1 of the study, we essentially repeated user study 1, except that, in this user study, we paired S++, CFR (the core technology behind world-class computer-poker algorithms [3]), and people together in two different repeated stochastic games rather than repeated normal-form games. The results of this study were essentially identical to the results of the first user study. In Part 2 of this user study, we allowed players to communicate with each other. We observed that the ability to talk to each other throughout the game vastly improved mutual cooperation among people. The extended version of S++ (S#–) also was able to raise its level of cooperation when paired with people, though cooperation emerged at a much slower rate than in human-human relationships.

(Appendix F) User Study: Human vs. Machine 3

We attributed the fact that S#– learned to cooperate with people much slower than people learned to cooperate with each other to the inability of S#– to listen to communication signals expressed by people. Thus, we developed S#, which not only generates and voices cheap talk, but also reacts to communication signals produced by its partner to help it select experts more effectively. We then conducted a final (66-participant) user study in April 2015 in which we paired S# and humans in three different repeated normal-form games (Prisoner's Dilemma, Chicken, and Alternator). The results of this study showed several important outcomes. First, S# paired with S# produced, by far, the highest levels of mutual cooperation under conditions in which players could and could not cooperate. Second, people did not consistently cooperate with each other in the absence of communication (mutual cooperation was approximately 30%). However, the frequency of mutual cooperation between people doubled when people could communicate (mutual cooperation was approximately 60%). Human-S# pairings followed identical trends. In both conditions, S# was able to learn to cooperate with people as well as people cooperated with each other. This illustrates that S# is able to successfully generate and respond to signals from a human partner so as to better elicit cooperative behaviors from people.

A Problem Formulation

This paper deals with decision-making in scenarios in which two entities (i.e., which could be people or machines) repeatedly interact with each other for an unknown amount of time. Life consists of many of these interactions, including relationships between family members, neighbors, co-workers, business partners, and nations, as well as interactions between a person and their laptop, handheld device, automated vehicle, and other embedded systems. These scenarios vary widely in many dimensions, though common attributes emerge in each problem, perhaps the most salient of which is that there are conflicting interests between the players. While individuals can benefit from cooperating with each other, each player has incentives to not cooperate. However, because the interactions are repeated (and often for long periods of time), non-cooperative behavior is often met with reciprocating behavior that can lead to dysfunctional relationships if the individuals cannot eventually find common ground.

In this paper, we assume that these interactions between players can be modeled as repeated stochastic games (RSGs). While such games do not capture every nuance of all interactions, the set of scenarios that can be modeled by repeated stochastic games is quite rich. RSGs are general enough to include, among other scenarios, repeated normal-form games and repeated extensive-form games.

In this section, we provide background notation used throughout the supplementary information, formally define RSGs, and state various assumptions made throughout the work. We then discuss the primary technical challenges associated with developing an algorithm that can effectively learn to interact with people in RSGs. Finally, we present and discuss example RSGs, focusing primarily on RSGs that are used in studies discussed in this paper.

A.1 Repeated Stochastic Games

We consider two-player RSGs played by players i and $-i$. An RSG consists of a set of *stage games* S. In each stage $s \in S$, both players choose an action from a finite set. Let $A(s) = A_i(s) \times A_{-i}(s)$ be the set of *joint actions* available in stage s, where $A_i(s)$ and $A_{-i}(s)$ are the action sets of players i and $-i$, respectively. In each stage, each player simultaneously selects an action from its set of actions. Once joint action $\mathbf{a} = (a_i, a_{-i})$ is played in stage s, each player receives a finite reward, denoted $r_i(s, \mathbf{a})$ and $r_{-i}(s, \mathbf{a})$, respectively. The world also transitions to some new stage s' with probability defined by $P_M(s, \mathbf{a}, s')$.

Each *round* of an RSG begins in the start stage $\hat{s} \in S$ and terminates when some goal stage $s_g \in G \subseteq S$ is reached. A new round then begins in stage $\hat{s}$. The game repeats for an unknown number of rounds.

Player i's strategy, denoted π_i, defines how it will act in each world *state*. In general-sum RSGs, it is often useful (and necessary) to define state not only in terms of stages, but also in terms of the history of the players' actions. Let H denote the set of possible joint-action histories. Then, the set of states is given by $\Sigma = S \times H$. Let $\pi_i(\sigma)$ denote the policy of player i in state $\sigma = (s, h) \in \Sigma$. That is, $\pi_i(\sigma)$ is a probability distribution over the action set $A_i(s)$. When $\pi_i(\sigma)$ places all probability on a single action $a_i \in A(s)$, the policy is said to be a *pure strategy*. Otherwise, the policy is called a *mixed strategy*.

A.2 Metrics of Success

The success of a player in an RSG is measured by the payoffs it receives. An ideal algorithm will maximize its payoffs (against any associate) over all rounds played. However, it is not possible to guarantee optimal behavior against arbitrary associates, even in simple games [4]. Thus, due to the difficulty in achieving, defining, and measuring optimal behavior in RSGs, much work on algorithmic development has focused on

developing algorithms that meet certain criteria, such as convergence to Nash equilibria [5, 6, 7, 8], Pareto optimality [9, 10], no-regret (also known as universal consistency) [11, 12, 13, 5], and security [5, 9][1].

However, despite the benefits that such properties might provide, achieving any one (or multiple) of these criteria does not necessarily guarantee high payoffs in repeated general-sum games. For example, while minimizing regret seems like a good idea, regret minimization is not guaranteed to correlate with higher payoffs in repeated games – a player with lower regret often obtains lower payoffs than a player with higher regret [14, 1]. As another example, many repeated general-sum games have an infinite number of Nash equilibria [15], each of which can have different values to the players. Thus, a guarantee to converge to Nash equilibria (alone) [8, 16] is a wholly insufficient property for successful learning in repeated games.

Thus, in this paper, we use two metrics of success:

1. *Empirical Performance*: Ultimately, the success of an algorithm or person in an RSG is measured by the sum of the payoffs (s)he/it receives throughout the duration of the game.

2. *Proportion of Mutual Cooperation*: The ability to cooperate is a key attribute of evolutionarily stable algorithms in RSGs [4]. This is largely because of the high impact that cooperation has on a player's empirical performance. Thus, we consider the ability to cooperate with each other (i.e., mutual cooperation) to be a key attribute of performance. However, in stating thus, we do not consider mutual cooperation as a substitute for high empirical performance, but rather as a supporting factor.

   The term cooperation has specific meaning in well-known games such as the Prisoner's Dilemma. However, in other games, the term is much more nebulous. Furthermore, mutual cooperation can be achieved in degrees; it is not necessarily an all or nothing event. However, for simplicity in this work, we define mutual cooperation as the *Nash bargaining solution* of the game [17], or the solution that maximizes the product of the advantages to the players. For each of the games considered in this work, we explicitly state this solution to avoid confusion.

A.3 Assumptions

The algorithms developed and analyzed in this work are designed for arbitrary two-player repeated stochastic games. This problem domain is quite general, though we do make several commonly made assumptions:

- As is common with the vast majority of repeated game experiments in behavior economics and biology [18, 19, 20, 21, 22], we assume that the full game description is given to both players prior to the beginning of the game. Both players know (1) the set of stages S, the set of joint actions $A(s)$ for all $s \in S$, the transition function P_M, and the reward functions $r_i(s, \mathbf{a})$, and $r_{-i}(s, \mathbf{a})$ for all $(s, \mathbf{a})$.

- Likewise, throughout the course of the game, we assume that the current stage s and the payoffs received by both players at the end of each stage are fully known to both players.

- We assume that the game is repeated for an unknown number of rounds, long enough that the players have sufficient time to develop and profit from relationships. In our user studies, games are repeated for approximately 50 rounds, which is consistent with past work reporting experiments in which people play repeated games [18, 19].

[1] In the interest of brevity, we omit formal definitions of these terms, and simply refer the reader to the cited literature or standard texts on game theory.

- Unless stated otherwise, the players were not told the identity of their associate, who could be a computer following some algorithm or another person. Participants were simply told that they were interacting with another player. This helps remove biases in strategies caused by predispositions toward certain player types, thus allowing for a properly controlled experiment.

While these assumptions can be relaxed in many instances, we make these assumptions to properly scope the work.

In this paper, we deal with repeated general-sum games, which include *zero-sum games* (one player's gain is the other player's loss), *fully cooperative games* (both player's receive the same payoff in all cases), and *games of conflicting interest* (the players receive different payoffs for at least some outcomes, but their payoffs are not fully in competition with each other). However, given our focus on cooperation in this paper, our analysis primarily concerns scenarios that have conflicting interests.

A.4 Technical Challenges

Developing learning algorithms that are general enough to perform effectively in arbitrary RSGs played against unknown associates (and particularly with people) requires that the algorithm can overcome many technical challenges. We now outline and briefly discuss some of these technical challenges.

- *Wide variety of games.* An algorithm must not be domain-specific. It must have superior performance in a wide variety of repeated games, which include zero-sum games, fully cooperative games, and games of conflicting interests. An algorithm should not be tailored to a particular game (such as the Prisoner's Dilemma), as the ability to elicit cooperative behavior from associates in one game does not necessarily entail that the same algorithm can induce cooperative behavior in another game. Furthermore, algorithms that perform effectively in zero-sum games often perform very poorly in games of conflicting interest (and vice versa). As an example, CFR, which plays a simplified version of Poker as well as people [3], cannot learn to cooperate in RSG's of conflicting interests [2] (see also results in Appendix E).

- *Multiple (even infinite) Nash equilibria.* RSGs often have multiple Nash equilibria. In fact, the folk theorem for repeated games tells us that, if the game is continued with high enough probability after each round, then many games have an infinite number of Nash equilibria of the repeated game [15]. Furthermore, it is often not possible for players to agree on which of these Nash equilibria is *best*, even if we assume "rational" players. Thus, due to these characteristics, attempts to "solve" RSGs via rationality assumptions and equilibrium computation alone are wholly insufficient. To maximize its payoffs, an algorithm must be able to learn (in real time) from its experiences, so that it can adapt to the unknown behaviors of people and other (unknown) associates. This knowledge has given raise to a large number of machine-learning algorithms for RSGs (see Appendix B).

- *Adaptive, unknown partners.* Standard machine-learning algorithms rely on stationary environments. However, when one's partner also learns, the environment is non-stationary with respect to the state-space of any practical machine-learning algorithm. Thus, straightforward applications of standard machine-learning algorithms are not guaranteed to produce effective behavior in RSGs, and often do not in practice. For example, such algorithms often learn myopic behaviors that preclude cooperation. In fact, the non-stationary nature of these environments is such that efforts to play a best response to the current estimates of the associate's strategy often produce non-cooperative behavior.

- *Large strategy spaces.* Even simple RSGs have rather large strategy spaces. As mentioned previously, an algorithm's strategy must define a policy in each state $\sigma \in S \times H$. Even if $|A_i(s)| = 2$ for all $s \in S$ and we only consider pure strategies, the size of player i's strategy space is on the order of $2^{|S||H|}$. As such, many machine-learning algorithms require thousands of rounds of experience to converge, even in extremely simple scenarios [8, 10]. Furthermore, large strategy species make it difficult for algorithms to reason at a high enough level to represent cooperation, reciprocity, etc.

- *Limited experience (data).* Unfortunately, in order to effectively interact with people, an algorithm most be able to derive effective behaviors within only a few rounds of experience. If a machine fails to produce realistic behaviors within short timescales, people are unlikely to interact with it in the future, thus eliminating the possibility of cooperation.

- *Human attributes.* The previous technical challenges are sufficient in and of themselves. However, human-machine interactions rely on more than just algorithmic achievements. To form effective relationships with people, machines must also address human attributes, including issues related to trust, friendship, and human "ways of thinking." Addressing such challenges is not easy. For example, human cooperation appears to not require sheer computational power, but rather relies on intuition [23], cultural norms [24], and pre-evolved dispositions toward cooperation [25], common-sense mechanisms that are difficult to encode in machines. On the other hand, online-learning algorithms rely on random exploration [26] to investigate the benefits of various courses of action. However, random exploration is likely to breed distrust in a human partner, thus leading to dysfunctional, non-cooperative, relationships.

These technical challenges make developing algorithms for RSGs interesting and extremely challenging. Failures to meet these challenges often cause AI algorithms to defect rather than to cooperate when cooperation and self-interest appear to be in conflict.

A.5 Example Games

RSGs include many common forms of repeated games, including repeated normal-form games and repeated, extensive-form games. We now describe several RSGs that are used in the user studies conducted as part of this research.

A.5.1 Example Repeated Normal-Form Games

Repeated normal-form games are RSGs that only have a single stage (i.e., $|S| = 1$). A two-player normal-form game can be represented by a payoff matrix, which shows the payoffs obtained by each player for each joint action. Table 1 shows five different normal-form games: Prisoner's Dilemma, Chicken, Shapley's Game, Chaos, and Alternator Game. At left is the payoff matrix of each game. The *row player's* actions correspond to the rows of the matrix, while the column player's actions correspond to the columns of the matrix. For example, in the the game Chicken, the row player's action set is $\{a, b\}$, while the column player's action set is $\{c, d\}$. The cells of a payoff matrix give the payoffs that are received by the two players for each joint action. The payoff (higher values are better then lower values) to the row player is listed first, followed by the payoff to the column player. For example, in the game Chicken, if the row player selected action a while the column player played d, then the row player would receive a payoff of 33 and the column player would receive a payoff of 100. Once both players have played an action and been informed of their payoffs, a new round begins.

Game	Attributes
Prisoner's Dilemma **c** | **d** **C** 60, 60 | 0, 100 **D** 100, 0 | 20, 20	**Mutual Cooperation**: `Cc` → $r_{\text{row}} = r_{\text{col}} = 60$ **Pure one-shot NE**: `Dd` → $r_{\text{row}} = r_{\text{col}} = 20$ **maximin strategy**: Row plays `D` → $r_{\text{row}} \geq 20$ Col plays `d` → $r_{\text{col}} \geq 20$
Chicken **c** | **d** **a** 84, 84 | 33, 100 **b** 100, 33 | 0, 0	**Mutual Cooperation**: `ac` → $r_{\text{row}} = r_{\text{col}} = 84$ **Pure one-shot NEs**: `bc` → $r_{\text{row}} = 100, r_{\text{col}} = 33$ `ad` → $r_{\text{row}} = 33, r_{\text{col}} = 100$ **maximin strategy**: Row plays `a` → $r_{\text{row}} \geq 33$ Col plays `a` → $r_{\text{col}} \geq 33$
Shapley's Game **d** | **e** | **f** **a** 0, 0 | 100, 0 | 0, 100 **b** 0, 100 | 0, 0 | 100, 0 **c** 100, 0 | 0, 100 | 0, 0	**Mutual Cooperation**: `ae-bd` → $E[r_{\text{row}}] = E[r_{\text{col}}] = 50$, and other equivalent solutions in which the players take turns **One-shot NE**: Both play randomly → $r_{\text{row}} = r_{\text{col}} = 33\frac{1}{3}$ **maximin strategy**: Row plays randomly → $r_{\text{row}} \geq 33\frac{1}{3}$ Col plays randomly → $r_{\text{col}} \geq 33\frac{1}{3}$
Chaos **d** | **e** | **f** **a** 46, 67 | 24, 6 | 100, 0 **b** 0, 37 | 37, 7 | 1, 53 **c** 14, 69 | 20, 100 | 71, 90	**Mutual Cooperation**: `cf` → $r_{\text{row}} = 71, r_{\text{col}} = 90$ **One-shot NE**: `ad` → $r_{\text{row}} = 46, r_{\text{col}} = 67$ **maximin strategy**: Row plays mixed strategy → $r_{\text{row}} \geq 28.85$ Col plays mixed strategy → $r_{\text{row}} \geq 42.79$
Alternator Game **d** | **e** | **f** **a** 0, 0 | 35, 70 | 100, 40 **b** 70, 35 | 10, 10 | 45, 30 **c** 40, 100 | 30, 45 | 40, 40	**Mutual Cooperation**: `cd-af` → $E[r_{\text{row}}] = E[r_{\text{col}}] = 70$ **Pure one-shot NEs**: `bd` → $r_{\text{row}} = 70, r_{\text{col}} = 35$ `ae` → $r_{\text{row}} = 35, r_{\text{col}} = 70$ **maximin strategy**: Row plays mixed strategy → $r_{\text{row}} \geq 31.11$ Col plays mixed strategy → $r_{\text{col}} \geq 31.11$

Table 1: Five normal-form games in which cooperation and self-interest are in conflict. In addition to giving the payoff matrix of each game, the table also defines various attributes about these games, namely the mutually cooperative solution (i.e., the Nash bargaining solution), one-shot Nash equilibria, and maximin strategies. For each attribute, the strategy/solution is listed, followed by the payoffs received by the row (r_{row}) and column (r_{col}) players when the strategy/solution is played.

Each of the games summarized in Table 1 are games of conflicting interest. Thus, in these games, each player requires the help of the other to receive a high payoff. However, both players could also potentially exploit or bully the other player. As such, to some extent, cooperation and self-interest appear to be in conflict in these games, though common-ground can be established to give both players high payoffs.

Mutual cooperation is obtained by different behaviors in the various games. In the Prisoner's Dilemma, Chicken, and Chaos, mutual cooperation is defined by a single joint action. On the other hand, Shapley's Game and Alternator Game require players to alternate between joint actions to produce fair and efficient payoffs. Such coordination, in which players are tempted to deviate, can be difficult to establish.

In each of these five games, mutual cooperation is distinct from the one-shot NEs and the maximin strategy. Thus, myopic behavior does not tend to produce mutual cooperation in these games. In each round, a choice to cooperate exposes a player to risks. However, when the game is repeated, a player can punish deviations from the cooperative solution in hopes of eliciting future cooperation from their partner.

Normal-form games have the advantage of presenting choices between cooperation and defection in simple terms. However, in many real-world scenarios, cooperation and defection are established by sequences of moves, rather than in a single step. Extensive-form and other multi-stage stochastic games model such scenarios. These games represent additional challenges for algorithm designers since they require algorithms to compute and reason about cooperation, defection, reciprocity, etc. over sequences of moves rather than in a single step. We present two such scenarios, which are used in the second user study in this paper, which is described in Appendix E.

A.5.2 A Repeated Extensive-Form Game

Extensive-form games have a tree-like structure in which player's take turns making moves. When played repeatedly, extensive-form games are RSGs in which the leaf node is the start stage and the leaf nodes are goal stages. In stages in which it is player i's turn to move, player $-i$'s action set has cardinality of 1 (i.e., $|A_{-i}(s)| = 1$); it has no ability to impact the outcome of that stage.

An example repeated extensive-form game is the Block Game depicted in Figure 1. The Block Game is a turn-taking game in which the two players share a set of nine blocks. In each round, the players take turns selecting a block until they each have three blocks, with one player (typically the older sibling) going first in each round. If a player's three blocks form a valid set (i.e., she has all blocks of the same color, all blocks of the same shape, or none of her blocks have the same color or shape), then her payoff in the round is the sum of the numbers on her blocks. If she fails to collect a valid set, she loses the sum of her blocks divided by 4.

Though rather simple, this game is strategically complex. Each player would like to collect all of the squares (40 points) or all of the red blocks (23 points). However, to reach either of these outcomes, the other player would have to accept either getting all of the triangles (10 points) or all of the blue blocks (17 points), respectively. Since the other player can get 18 points by taking blocks that all differ in shape and color, (s)he/it is unlikely to repeatedly accept either of those two outcomes. Thus, a player has to decide whether to try to bully the other player (possibly by punishing the other player, which might require the acceptance of outcomes producing negative payoffs in early rounds in order to get better outcomes in later rounds) or to collaborate with them in some way. This decision depends on the characteristics of the other player.

There are a number of different solutions the players can converge to in this game, which we list in descending collaborative order:

1. *(Mutual Cooperation) Alternate between □'s and △'s* – The players alternate between selecting all of the squares and all of the triangles. Thus, both players average 25 points per round $((40 + 10)/2)$.

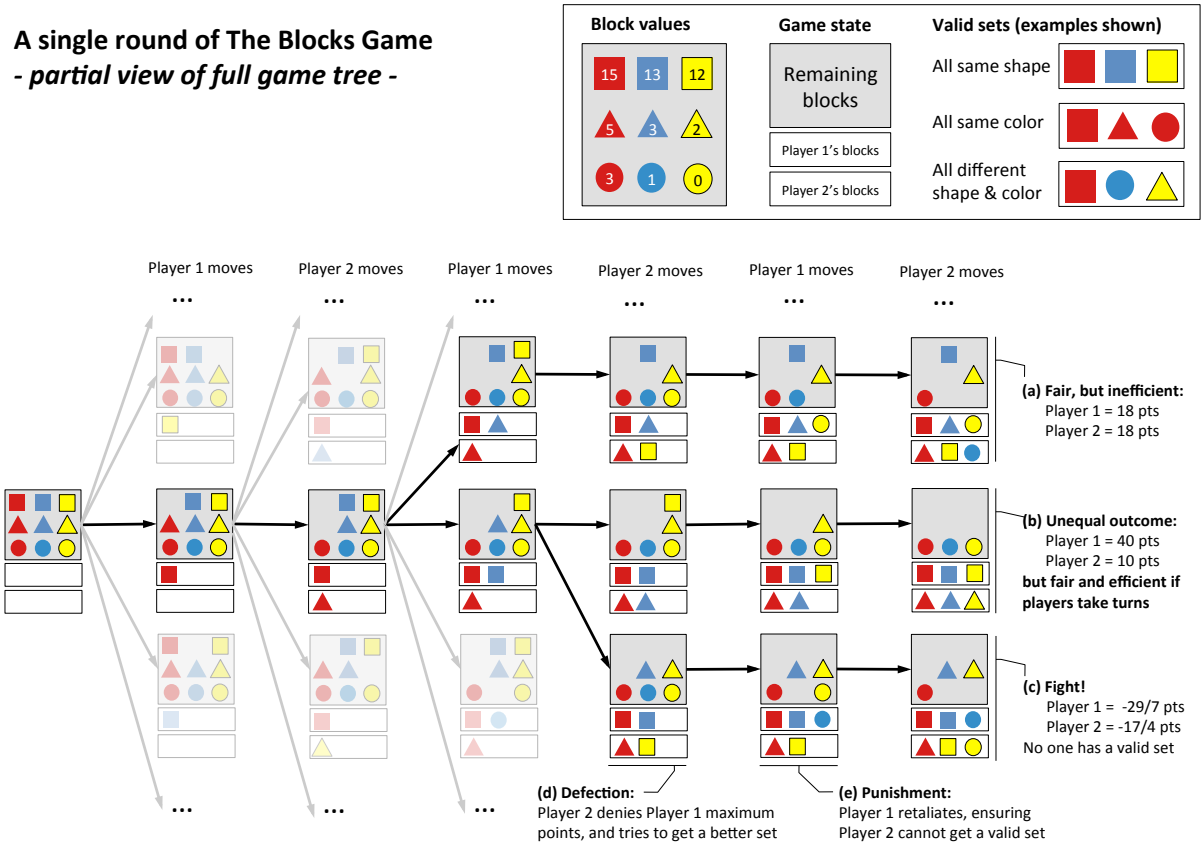


Figure 1: An extensive-form game in which two players share a set of blocks. The two players take turns selecting blocks from a nine-piece blocks set until they each have three blocks. The goal of each player is to get a valid set of blocks with the highest value as possible, where the value of a set is determined by the sum of the numbers on the blocks. Invalid sets produce negative points, defined by the sum of the values on the blocks divided by four. (a) A fair, but inefficient outcome in which both players receive 18 points. (b) An unequal outcome in which one player receives 40 points, while the other player receives just 10 points. However, when the players take turns getting the higher payoff (selecting all the squares), this is the Nash bargaining solution of the game, producing an average payoff of 25 to both players. (c) An outcome in which neither player obtains a valid set, and hence both players lose points. (d) This particular negative outcome is brought about when player 2 defects against player 1 by taking the block that player 1 needs to complete its (most-valuable) set. (e) Player 1 then retaliates to ensure that player 2 does not get a valid set either.

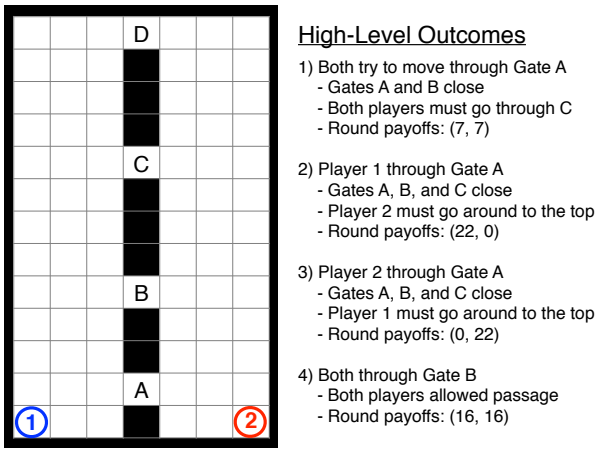


Figure 2: The SGPD, a maze version of the prisoners' dilemma.

2. *Alternate between red and blue* – The players take turns selecting the red and blue blocks (respectively) in alternating rounds, Both players average 20 points per round $((23 + 17)/2)$.

3. *All different* – The players both select blocks with no matching attributes. This always gives both players 18 points. This solution is fair, but not efficient, since alternating between □'s and △'s gives both players higher payoffs.

4. *Pure red-blue* – One of the players always takes all the red blocks (23 points), while the other player always takes the blue blocks (17 points).

5. *Pure* □-△ – One player always takes the squares (40 points), while the other player always gets the triangles (10 points).

A.5.3 A Stochastic Game Prisoner's Dilemma

RSG's also exist that are neither repeated normal-form games nor repeated extensive-form games. One such game is the stochastic game prisoner's dilemma (SGPD) [27], a maze game in which the high-level payoffs of the game equate to a standard prisoner's dilemma. At the start of each round of the game, the players are placed in opposite corners of the maze as shown in Figure 2. The players move (simultaneously) to adjacent cells (up, down, left, or right) with the goal of reaching the other player's start position in as few moves as possible. Each move costs a player one point, but a player receives 30 points when it reaches its goal. Once both players have arrived at their respective goals, a new round begins from the original start stage.

To reach their goals, the players must pass through one of four gates (Gates A-D). While the least-cost path to the goal is through Gate A, only one player can pass through Gate A in a round. When a player passes through Gate A, Gates A, B, and C close for the other player, and it must pass through Gate D. If both players attempt to pass through Gate A at the same time, neither is allowed passage and Gates A and B close. On the other hand, both players can pass through Gates B, C, and D separately or at the same time, though Gate A closes when either player passes through Gate B.

We expect players to eventually converge to one of four solutions, which we list in descending collaborative order:

1. *(Mutual Cooperation) Both B* – Both players use Gate B, resulting in each receiving 16 points per round.

2. *Alternate between Gates A and B* – The players take turns going through Gate A, which results in an average payoff of $11 = \frac{22+0}{2}$ points per round to each player.

3. *Both go attempt to go through Gate A* – Both players attempt to go through Gate A, thus requiring each player to pass through Gate C. This gives both players 7 points.

4. *Exploitation* – One of the players always goes through Gate A, leaving the other to go through Gate D. This gives the players 22 and 0 points, respectively.

A.5.4 Summary

These example RSGs differ in many different ways. However, in each case, a player depends on the cooperation of its partner to receive high payoffs. Bullying and other forms of defection are also possible in each game. When profitable, algorithms must be able to establish effective, cooperative relationships when associating with unknown partners, including people. When one's partner does cooperate, one is also faced with the choice of reciprocating cooperation, or seeking more profitable payoffs by defecting. Whether or not defection is worthwhile depends on the algorithms/strategies employed by one's partner.

B A Comparison of AI Algorithms in Repeated Games

Algorithms for learning to play repeated games were first studied in the context of equilibrium attainment and computation. Example algorithms included Fictitious Play [28] and other forms of so-called "rational" learning [29, 5]. In the early 1980s, Axelrod conducted the first computer tournament, in which he compared algorithms in the iterated prisoner's dilemma [4]. But it was not until the pioneering work of Littman, who combined reinforcement learning with concepts from game theory [30], that research in the area of learning in games was performed in mass among members of the AI community (e.g., [31, 32, 33, 34, 6, 35, 36, 8, 37, 38, 39, 40, 7, 41, 16, 42, 43, 44, 12, 45, 9, 46, 47, 48, 10, 49, 50, 51, 52, 14, 53, 1, 54, 55, 3]). As a result of this research emphasis, many general-purpose learning algorithms have been developed, many of which are quite sophisticated and effective.

In this paper, we are interested in developing algorithms that can effectively learn to interact with people in arbitrary repeated games. Despite the large amount of research devoted to algorithmic development in the area of repeated games, very little work has addressed how well these algorithms are capable of playing repeated general-sum games with people, though game-specific research has been done in one-shot games [56]. However, with the growing presence of autonomous machines in society that interact repeatedly with us (i.e., repeated games), developing algorithms that can establish cooperative relationships with people, and do so in arbitrary scenarios, is critical.

To be capable of establishing effective relationships with people, an algorithm must have several properties:

1. *Game independence.* The algorithm must have superior performance in general, and not in only specific, targeted games. For example, the algorithm should perform effectively across many scenarios rather than being limited to a small set of payoff matrices.

2. *Partner independence.* The algorithm should be capable of establishing effective relationships with unknown associates, whose behavial dynamics may or may not change over time (i.e., one's partner might be learning too!). There has been a tendency to focus on analyzing the performance of algorithms against static (non-adaptive) partners. While this focus lends itself to elegant theoretical results, such results say little about whether algorithms will be able to effectively interact with people (who's behavior is not static).

3. *Fast.* The algorithm should be able to learn effective behavior within short timescales – i.e., within a relatively small number of rounds of experience. People are unlikely to continue to utilize and interact with machines that do not quickly produce effective behavior.

4. *Cooperative (when beneficial).* When beneficial, the algorithm should have the foresight to forego immediate, short-term payoffs in order to establish effective cooperative relationships that have long-term value. However, an algorithm should not cooperate for the sake of cooperating. We assume a self-regarding machine, so cooperation should emerge only if cooperation improves the machine's payoffs. If the algorithm can obtain higher payoffs by exploiting its partner, it should do so.

5. *Evolutionary robustness.* The algorithm must avoid being invaded and, when possible, be capable of invading other algorithms. Algorithms that are not evolutionarily stable are likely to be exploited by humans or replaced by other algorithms.

We anticipate that these properties, though difficult to achieve for arbitrary games, are necessary, but insufficient, characteristics of an algorithm that can establish and maintain effective relationships with people.

Thus, in this section, we evaluate a representative set of existing AI algorithms with respect to these properties.

B.1 Algorithms

We compared a representative set of 19 existing algorithms, plus RANDOM, across 390 different two-player, normal-form games using Axelrod-style tournaments [4]. These algorithms, along with citations, parameter values, and other implementation details are list in Table 2. While certainly not all-inclusive, these algorithms represent many different genres of algorithms. Many of these algorithms have also been shown to have high performance characteristics in previous empirical evaluations [9, 50, 10, 1].

The selected algorithms represent a variety of algorithmic types. For example, FICTITIOUS PLAY and STOCHASTIC FICTITIOUS PLAYER represent belief-based (rational) learners, while we included Q-LEARNING, WOLF-PHC, MBRL-1, MBRL-2, JCAL, and M-QUBED to represent reinforcement-learning algorithms. Expert algorithms are also represented in the set, including the no-regret algorithms GIGA-WOLF and EXP3, as well as WMA, S++, EEE, and S. The set also includes a satisficing algorithm (S-LEARNING), two leader algorithms [39] (BULLY and GODFATHER; also known as zero-determinant strategies [57]), and two hybrid leader-follower algorithms named MANIPULATOR and MANIPULATOR-GODFATHER.

In selecting parameter values for these algorithms, we sought to remain consistent with values used in the published literature, while also seeking to optimize the performance of the algorithms. However, it was not possible for us to test all combinations of parameter values for all algorithms. In any event, this comparison is not intended to identify the *best* algorithm for repeated games, but rather to find a robust and effective algorithm that meets the previously stated criteria.

B.2 Games

We paired the algorithms together in 390 different two-player, normal-form games. There are 78 structurally distinct two-player, two-action normal-form games in which each player strictly orders its preferences over the four outcomes [58]. However, different distances between the four payoffs can create quite different games. Thus, for each of the 78 structurally-distinct games, we considered five different sequences of numbers, translated and scaled to the range $[0, 1]$ such that the lowest payoff is 0, and the highest is 1:

1. Standard: 1, 2, 3, 4 → 0.00000, 0.333333, 0.666667, 1.000000
2. Power2: 1^2, 2^2, 3^2, 4^2 → 0.00000, 0.200000, 0.533333, 1.000000
3. Power3: 1^3, 2^3, 3^3, 4^3 → 0.00000, 0.111111, 0.412698, 1.000000
4. Separated: $\frac{\mathbf{1}-1}{2}$, $\frac{\mathbf{2}-1}{2}$, $(\mathbf{3}-1)\cdot 2$, $(\mathbf{4}-1)\cdot 2$ → 0.00000, 0.083333, 0.666667, 1.000000
5. SquareRoot: $\sqrt{1}$, $\sqrt{2}$, $\sqrt{3}$, $\sqrt{4}$ → 0.00000, 0.414214, 0.732051, 1.000000

In short, for each of the 78-structurally distinct games, we form five different games using the five different sequences of numbers supplied above. Thus, for the structural game associated with prisoner's dilemmas, we form the five different games shown in Table 3. Each game is a variation on the same game that can provide a unique challenge. For example, in the game shown in Table 3c, the players are both better off by alternating between the solutions `bc` and `ad` rather than always playing the solution `ac`, though defection (the second action) remains the dominant action.

Table 2: Representative algorithms and implementation details.

Algorithm	Implementation Details
FICTITIOUS PLAY (FP)	Originally proposed by Brown [28]. See also Fudenberg and Levine [5]. Fictitious play is a special case of "rational" learning with a Dirichlet-distributed prior [5]. We used $\kappa^0(a) = 0$ for all a. Typically studied for the computation of Nash equilibria, it is still interesting as an online-learning algorithm.
STOCHASTIC FICTITIOUS PLAY (SFP)	See Fudenberg and Levine [5]. A variant of Fictitious Play that can learn to play mixed strategies. We used $\kappa^0(a) = 0$ for all a, along with Boltzmann exploration with temperature parameter set to $\tau = \frac{100}{t}$.
Q-LEARNING (QL)	A model-free reinforcement learning algorithm proposed by Watkins [59]. Our implementation encodes state as the previous joint action and uses ε-greedy exploration. Q-values were initialized randomly in the interval $[0, \frac{1}{1-\gamma}]$. $\varepsilon = \frac{1}{10+t/10}$, $\gamma = 0.95$, $\alpha = \frac{1}{10+t/100}$.
WOLF-PHC	By Bowling and Veloso [8]. A reinforcement learning algorithm that uses policy-hill climbing and a variable learning rate. We used $\alpha = \frac{1}{10+t/100}$, $\varepsilon = \frac{1}{10+t/100}$, $\delta_l = \frac{2}{100+t/100}$, $\delta_w = \frac{1}{100+t/100}$.
CJAL	By Banerjee and Sen [60]. A conditional joint-action learner. $N = 400$ and $\epsilon = 0$.
MBRL-1	A model-based reinforcement learning algorithm that encodes its state as the previous joint action. The algorithm estimates the associate's behavior using the fictitious-play assessment conditioned on the current state, and then computes a best response using value iteration (discount factor $\gamma = 0.95$). It uses ε-greedy exploration, $\varepsilon = \frac{1}{10+t/10}$, and $\kappa^0(a) = 1$ for all a.
MBRL-2	Same as MBRL-1, except that it encodes state as the previous two joint actions.
M-QUBED	By Crandall and Goodrich [10]. A model-free reinforcement learning algorithm. Parameters set as specified in Table 7 of the cited paper.
WMA	Weighted Majority Algorithm, by Freund and Shapire [61]. We used $\beta_i = 0.999$.
EXP3	By Auer et al. [31]. A no-regret algorithm. We used $\eta_i = \frac{\lambda}{\lvert A_i \rvert}$, where $\lvert A_i \rvert$ is the number of actions for player i, and $g_i = 50000$.
GIGA-WOLF	By Bowling [12]. A no-regret algorithm. We used $\eta_i = \frac{1}{\sqrt{t}}$.
S++	By Crandall [1]. Our implementation was identical to that of the published work.
EEE	By de Farias and Meggido [43]. An ε-greedy expert algorithm. In our implementation, the algorithm selects from the same set of experts as S++. We used $\omega = \lvert A_i \rvert \lvert A_{-i} \rvert$ and $\varepsilon = \frac{1}{10+t/10}$.
S	Derived from Karandikar et al. [34]. Our implementation of S [1] is identical to that of S++ except we do not prune the set of selectable experts or implement best-response and security overrides (as is done with S++). We used $\lambda = 0.99$.
SATISFICING (S-LEARN)	By Stimpson and Goodrich [41], derived from [34]. Uses aspiration learning [62] and a satisficing decision rule to select an action in each round. We used $\lambda = 0.99$.

Continued on the next page

Table 2 (continued) Representative algorithms and implementation details.

Algorithm	Implementation Details
BULLY	By Littman and Stone [39, 40]. The algorithm is similar to a zero-determinant strategy [57] (but discovered a decade earlier). The version of the algorithm we use enforces (by punishment) the target solution that can produce the highest payoff to the player when the associate maximizes its own payoffs given that it (1) remembers only the last joint action played and (2) plays only pure strategies.
GODFATHER (GF)	By Littman and Stone [39, 40]. Equivalent to BULLY with a different target solution. GODFATHER seeks to enforce the target solution that maximizes the product of the players' advantages. This algorithm is similar (but not identical) to a generalized tit-for-tat strategy.
MANIPULATOR	By Powers and Shoham [9]. This algorithm serially follows three different algorithms. In our implementation, it follows BULLY for the first 150 rounds. Thereafter, if its payoffs become lower than expected, it switches to MBRL-1. After 300 rounds, if its average payoff ever drops substantially below its maximin value (i.e., $v_i^{\mathrm{mm}} - \varepsilon$), it plays its maximin strategy thereafter.
MANIPULATOR-GODFATHER	By Powers and Shoham [9]. Identical to MANIPULATOR, except that it follows GODFATHER instead of BULLY over the first 150 rounds.
RANDOM	Randomly chooses an action from a uniform distribution in each round.

Human relationships, human-machine relationships, and machine-machine relationships can consist of various lengths of interactions, from just a handful of encounters to hundreds and even thousands of encounters (such as relationships with co-workers, family members, and smart phones). Thus, we consider three different game lengths: 100-round games, 1000-round games, and 50,000-round games. A satisfactory algorithm must perform effectively over all game lengths.

B.3 Round-Robin Tournaments

We first evaluated the algorithms in round-robin tournaments, in which each algorithm was paired with each of the other algorithms and itself in each of the 390 games, both as a row and as a column player. The winner of the round-robin tournament is the algorithm that has the highest payoff, averaged over all rounds played.

The full results of the round-robin tournament for all three game lengths are provided in Tables 6–8. We make several observations about these results:

- S++ finished first in the round-robin tournaments for each game length, though there was not a lot of separation among the top few algorithms at each game length.
- BULLY, a zero-determinant algorithm [57], finished second in the 1000-round and 50,000 round tournaments, and fourth in the 100-round tournament.
- Interestingly, one of the oldest (and simplest) algorithms, FICTITIOUS PLAY, finished second place in 100-round games. FICTITIOUS PLAY has a very simple representation, which often produces

(a) Standard

	c	d
a	0.666667, 0.666667	0.000000, 1.000000
b	1.000000, 0.000000	0.333333, 0.333333

(b) Power2

	c	d
a	0.533333, 0.533333	0.000000, 1.000000
b	1.000000, 0.000000	0.200000, 0.200000

(c) Power3

	c	d
a	0.412698, 0..412698	0.000000, 1.000000
b	1.000000, 0.000000	0.111111, 0.111111

(d) Separated

	c	d
a	0.666667, 0.666667	0.000000, 1.000000
b	1.000000, 0.000000	0.083333, 0.083333

(e) SquareRoot

	c	d
a	0.732051, 0.732051	0.000000, 1.000000
b	1.000000, 0.000000	0.414214, 0.414214

Table 3: Five different games produced for one of the 78 structurally distinct payoff orderings.

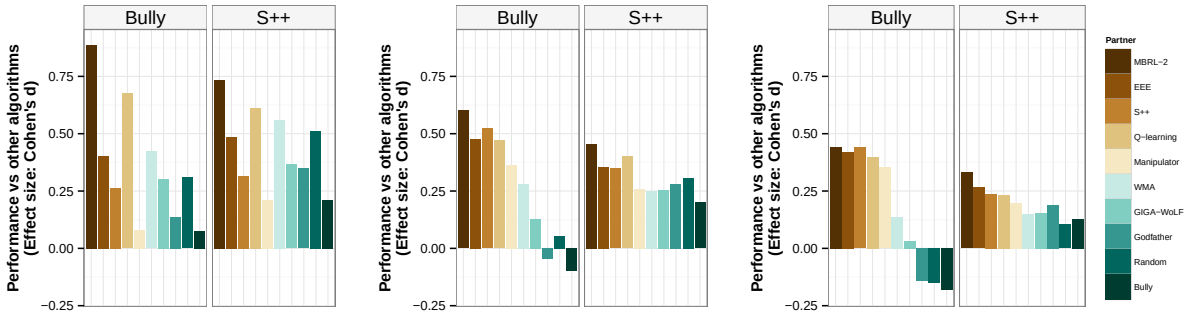
reasonable, yet myopic, strategies quickly. Thus, this algorithm is effective in relationships of short durations, but is not as effective in long-term relationships, as it often fails to cooperate.

- Another algorithm that performed consistently well was MANIPULATOR, which finished in third for all three game lengths. MANIPULATOR is identical to BULLY for the first 150 rounds, and continues to play BULLY if it succeeds in bullying its associate. Otherwise, it switches to MBRL-1.

- GODFATHER is similar (though not identical) to a generalized tit-for-tat, which has been shown to be a robust strategy in the iterated prisoner's dilemma [4]. Nevertheless, it was not a highly effective strategy in these round-robin tournaments, finishing no higher than seventh (100-round games) and as low as fourth to last (50,000-round games). Its low performance shows the relative strength of AI algorithms, many of which have the ability to adapt to varied payoff structures and the (initially) unknown tendencies of their partners.

Despite the fact that S++ and BULLY had very similar average payoffs across the round-robin tournaments, they had very different performance profiles (Figure 3). BULLY tended to perform very well against some algorithms (algorithms that adapt to its strategy), but it also tended to perform quite poorly when paired with other algorithms (such as itself, and other algorithms that refuse to adapt to it). On the other hand, S++ substantially outperformed the average when paired with each algorithm. These results show that S++ is more adept than BULLY at interacting with a wide range of partners.

The consistency of S++ across many different payoff matrices, game lengths, and partners can be traced to several of its properties. First, S++ learns to cooperate with copies of itself, consistently receiving among the higher average payoffs in self play (see Tables 6–8). In 50,000-round games, its average per-round payoff in self play is equivalent to what is obtained in the Nash bargaining solutions [17] (averaged over all games). Additionally, S++ learns cooperative behaviors within relatively short timescales. For example, in the prisoner's dilemma (Table 3a), it learns to cooperate within relatively few rounds of experience in self play, whereas many other learning algorithms either fail to learn to cooperate (and thus obtain substantially lower payoffs than they otherwise might have), or only learn to cooperate after hundreds or thousands of rounds (Figure 4).



(a) 100-round games (b) 1000-round games (c) 50,000-round games

Figure 3: The payoffs of BULLY and S++ compared to the average payoffs obtained by other algorithms (in terms of effect size: Cohen's d) against a variety of algorithms in (a) 100-round games, (b) 1000-round games, and (c) 50,000-round games. While BULLY is very effective when paired with some algorithms algorithms, it performs poorly when paired with others. On the other hand, S++ substantially outperforms the average in all pairings at all game lengths.

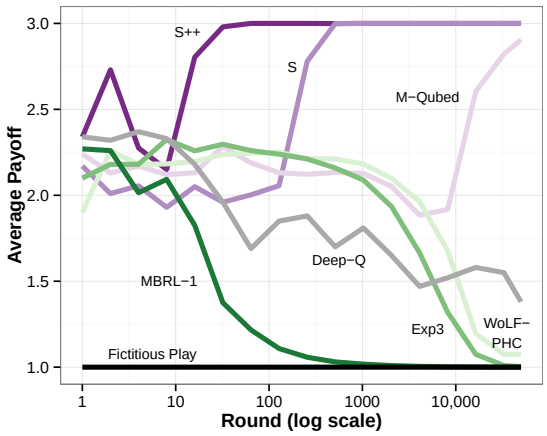


Figure 4: Average payoffs over time of eight different machine-learning algorithms in self play in a standard (0-1-3-5)-prisoner's dilemma (Table 4a). Results are an average of 50 trials each. Some of the algorithms learn to cooperate (the purple-shaded curves), while others learn to defect (green/grey-shaded curves). Only S++ learns to cooperate within timescales that support interactions with people. This figure is a duplication of Figure 1c from the main paper.

(a) (0, 1, 3, 5)-Prisoner's Dilemma

	cooperate	**defect**
cooperate	3, 3	0, 5
defect	5, 0	1, 1

(b) Chicken

	Swerve	**Straight**
Swerve	84, 84	33, 100
Straight	100, 33	0, 0

Table 4: Two different normal-form games in which cooperation and self-interest are in conflict.

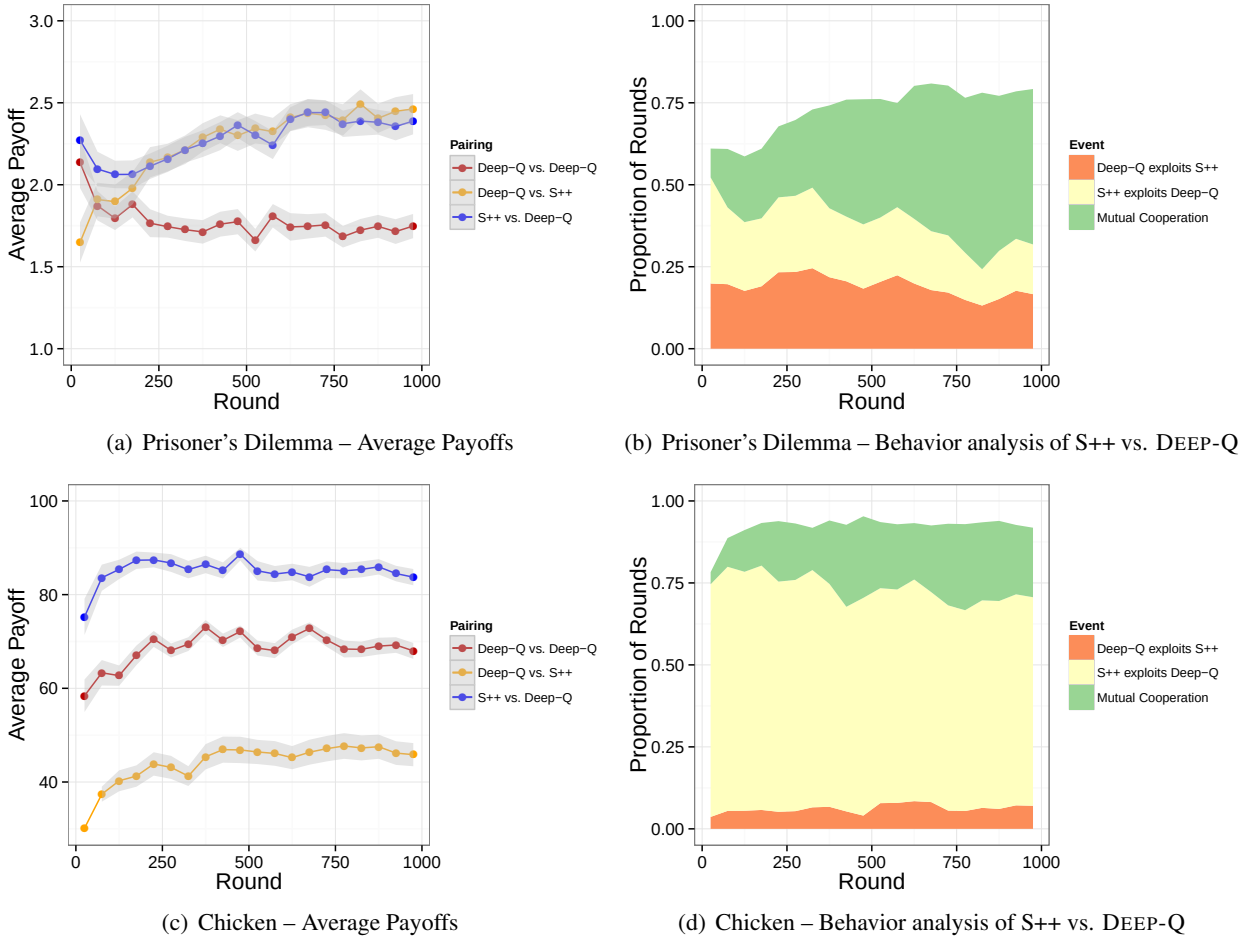
Algorithm	**Implementation Details**
DEEP-Q	An implementation of the Q-learning algorithm that used an artificial neural network to approximate the underlying Q-values. We used the Encog Java implementation (http://www.heatonresearch.com/encog) with resilient propagation (RPROP) applied as a learning rule [63]. To speedup the learning process, we followed the steps of the Deep-Q-Network [64], and used experience replay [65]. The state was encoded as the previous joint action, one node for each player. Output nodes represented the Q-values, one for each action. For two-player, two-action normal form games, we used a feedforward network with a 2-4-4-2 architecture (two input nodes, two output nodes, and two hidden layers of 4 nodes each). ε-greedy exploration was used with $\varepsilon = 0.1$ and $\gamma = 0.9$.

Table 5: The DEEP-Q algorithm. We did not include DEEP-Q in the set of 19 representative algorithms since (1) we had other representative reinforcement-learning algorithms and (2) it has a slow runtime, making the gathering of large amounts of data difficult. Still, given the recent success of the Deep-Q-Network in Atari games [64], it is a very interesting algorithm for comparison in repeated games.

Second, S++ also learns effective, though not always cooperative, strategies when associating with other kinds of algorithms. As a case study, we consider S++'s behavior when it is paired with DEEP-Q in a (0, 1, 3, 5)-Prisoner's Dilemma and in Chicken (Table 4). DEEP-Q (Table 5) defects most of the time in self play in the Prisoner's Dilemma, with occasional short runs of mutual cooperation. This results in relatively low average payoffs that tend toward 1 over time (Figure 4). S++ obtains much higher payoffs against DEEP-Q than DEEP-Q does (Figure 5a). The reason for this success is that S++ is able to elicit more frequent cooperative behavior from DEEP-Q (giving both players higher payoffs) such that the algorithms eventually engage in mutually cooperation in about half of the rounds (Figure 5b). Thus, when paired with DEEP-Q in the Prisoner's Dilemma, S++ is preferable to DEEP-Q due to its ability to establish a cooperative relationship.

In Chicken, S++ is also preferable to DEEP-Q (when paired with DEEP-Q), but for a different reason. In self play, DEEP-Q often learns to cooperate (wherein both players `swerve`) in Chicken, and on average obtains a per-round payoff of approximately 70 (Figure 5c). However, when S++ is paired with DEEP-Q in Chicken, it often exploits DEEP-Q (Figure 5d) wherein S++ plays `straight` while DEEP-Q plays `swerve`. This results in increased payoffs to S++. Thus, the performance advantage of S++ over DEEP-Q in Chicken is due to its ability to learn to exploit DEEP-Q.

Additional examples of how S++ adapts to its partner's behavior are provided in Figure 6, which shows the kinds of strategies that S++ learns to play against various associates in the game Chicken. S++ is an expert algorithm (see Appendix C) that selects among a set of expert strategies. These strategies include another learning algorithm (MBRL-1), the maximin strategy (Maxmin), and many leader and follower strategies [39] that attempt to establish different kinds of joint solutions. Rather than enumerate all strategies



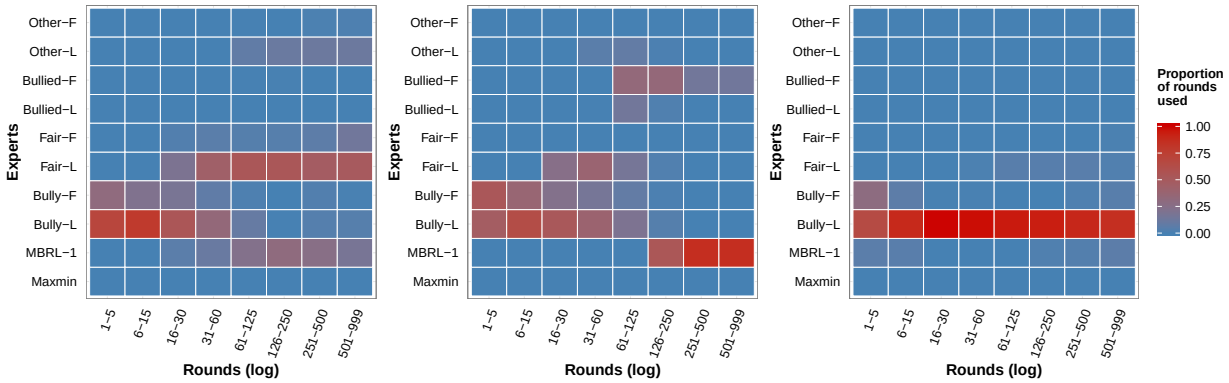
(a) Prisoner's Dilemma – Average Payoffs

(b) Prisoner's Dilemma – Behavior analysis of S++ vs. DEEP-Q

(c) Chicken – Average Payoffs

(d) Chicken – Behavior analysis of S++ vs. DEEP-Q

Figure 5: Analysis of the interaction between S++ and DEEP-Q in a Prisoner's Dilemma and in Chicken. All results are an average of 50 trials. (a) Average payoffs obtained by DEEP-Q in self play, and by DEEP-Q and S++ when they are paired with each other in the Prisoner's Dilemma. Error bands show the standard error on the mean. (b) The proportion of rounds that DEEP-Q and S++ cooperate together and exploit each other in the Prisoner's Dilemma. (c - d) Repeat of the evaluations presented in (a) and (b) in the game Chicken. These results demonstrate that S++ performs substantially better against DEEP-Q than DEEP-Q does against itself in both games. However, it does so for different reasons. In the Prisoner's Dilemma, S++'s superior performance is caused by its ability to elicit cooperation from DEEP-Q, whereas its superior performance in Chicken is brought about by its ability to exploit DEEP-Q. This illustrates the ability of S++ to adapt to its partner and the game in order to obtain high payoffs.



(a) S++ vs. S++ (b) S++ vs. Bully (c) S++ vs. MBRL-2

Figure 6: An illustration of the learning dynamics of S++ in Chicken. For ease of understanding, experts are categorized into groups (see text). (a) The proportion of time that S++ selects each group of experts over time when paired with a copy of itself. S++ initially seeks to bully its associate, but then switches to fair, cooperative experts when attempts to exploit are unsuccessful. (b) When paired with BULLY, S++ learns the best response, which is to be bullied, achieved by playing MBRL-1, Bully-L, or Bully-F. (c) S++ quickly learns to play experts that bully MBRL-2. These results are a duplication of Figure 3 from the main paper.

in the figure, we group them into eight categories. Bully-L and Bully-F consist of leader and follower experts, respectively, that advocate Pareto efficient solutions that give S++ a higher payoff than its partner. Fair-L and Fair-F similarly consist of leader and follower experts, respectively, that advocate Pareto efficient solutions that give S++ and it partner the same payoff. Bullied-L and Bullied-F consist of Pareto optimal strategies that give S++'s partner higher payoffs than S++, and Other-F and Other-L consist of the remaining strategies, which advocate Pareto-dominated outcomes.

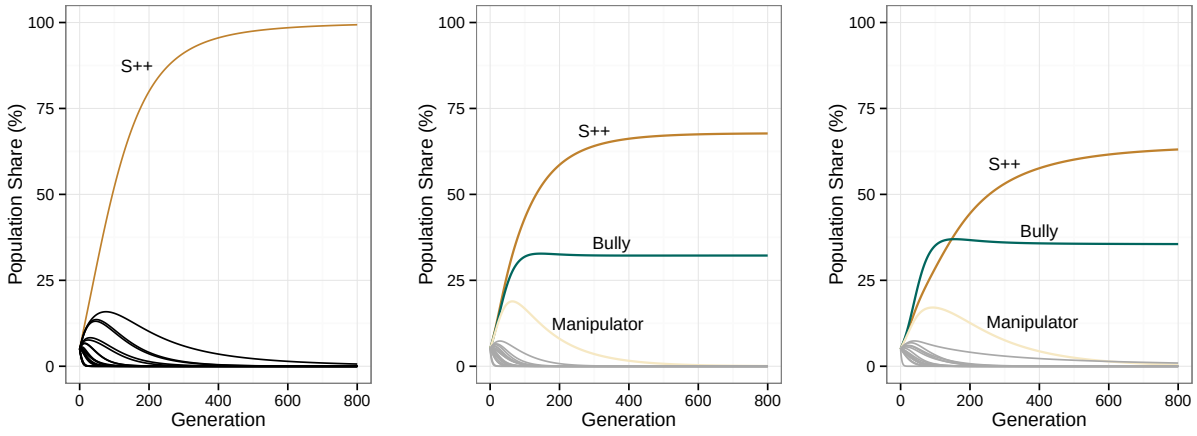
Figure 6 shows that S++ learns widely different strategies when associating with itself, BULLY, and MBRL-2 in the game Chicken. In self play, S++ learns to play fair strategies (Figure 6a). On the other hand, when paired with BULLY it learns to play a best response (Figure 6b) to BULLY's strategy. On the other hand, S++ learns to exploit MBRL-2 (Figure 6c).

In short, S++ generally meets the first four necessary properties outlined at the beginning of this section. Relative to existing learning algorithms, it has superior performance across many different games. It performs effectively against many different kinds of algorithms, including itself, other learning algorithms, and static (non-adaptive) strategies. It also learns relatively fast, and is able to learn to cooperate or exploit, depending on what will give it the highest payoffs.

We now turn to the last necessary property, that of evolutionary robustness.

B.4 Evolutionary Dynamics

We used the replicator dynamic [66] to examine the evolutionary dynamics of a population consisting of the 20 representative algorithms (Table 2). Formally, let x_i^t be the proportion of the population that uses algorithm i in generation t, such that $(x_1^t, ..., x_{20}^t)$ describes the distribution of algorithms used in the population, subject to the constraint that $\sum_{i=1}^{20} x_i^t = 1$. Also, let $R(i, j)$ denote the expected per-round payoff of algorithm i when it associates with algorithm j, which is determined by the results of the round-robin tournaments. These values are provided in Tables 6–8 for tournaments with 100, 1000-, and 50,000-round



(a) 100-round games (b) 1000-round games (c) 50,000-round games

Figure 7: Evolutionary dynamics in 100-, 1000-, and 50,000-round games.

games, respectively. Then, the fitness of algorithm i in generation t is given by

$$f_i^t = \sum_{j=1}^{20} x_j^t R(i,j). \tag{1}$$

Finally, the proportion of the population using algorithm i in generation $t+1$ is given by:

$$x_i^{t+1} = x_i^t + x_i^t \left(f_i^t - \phi_t \right), \tag{2}$$

where ϕ_t is the average fitness of the population in generation t, given by $\phi_t = \sum_{j=1}^{20} x_j^t f_j^t$.

Initially, we divided the initial population equally among the 20 algorithms, such that $x_i^0 = \frac{1}{20}$. The resulting evolutionary dynamics for 100-, 1000-, and 50,000-round games, respectively, are shown in Figure 7.

Due to its strong performance attributes described earlier in this appendix, S++ gained the highest population share at all game lengths. While BULLY was able to gain and then maintain a minority percentage of the population in games that repeated many rounds, the population evolved so that S++ completely dominated the population in shorter games.

The evolutionary fitness of S++ can be traced to its ability to invade and avoid being invaded. Tables 6–8 reveal two very interesting outcomes. First, with only two exceptions (GODFATHER in 100-round games and M-QUBED in 50,000-round games), S++ scores higher against each algorithm than that algorithm scores when paired with itself. Second, with two exceptions (BULLY and MANIPULATOR in 1000- and 50,000-round games), S++ obtains a higher payoff when paired with itself than the other algorithms obtain when paired with S++. Thus, the results shown in Figure 7 are robust to nearly every initial population. S++ can invade all of these algorithms with the exception of GODFATHER in 100-round games and M-QUBED in 50,000-round games. Likewise, S++ cannot be invaded by any of these algorithms, except that BULLY and MANIPULATOR can gain and maintain minority population shares in 1000- and 50,000-round games.

B.5 Conclusion

While there are many strong AI algorithms for repeated games, S++ [1] is one of the most effective among those we evaluated. It has all of the five desired properties outlined at the beginning of this appendix. It performs effectively across a wide range of games, has high performance against many different kinds of associates, and learns effective behaviors within short timescales. It also has the ability to cooperate with copies of itself and other strong algorithms, but also learns to exploit algorithms when doing so is beneficial. Finally, it has strong evolutionary characteristics at both shorter- and longer-term interaction lengths. Together, these results indicate that S++ could potentially be effective at learning to interact with people in repeated games.

	S++	FP	Manip-ulator	Bully	S	MBRL 1	GF	Manip-GF	EEE	GIGA-WoLF	MBRL 2	M-Qubed	WoLF-PHC	S-learn	QL	WMA	Exp3	sFP	Rand-om	CJAL	Ave
S++	0.685	0.752	0.654	0.655	0.728	0.772	0.744	0.744	0.731	0.706	0.746	0.695	0.686	0.702	0.709	0.626	0.614	0.626	0.612	0.612	**0.690**
FP	0.681	0.695	0.684	0.685	0.704	0.692	0.726	0.726	0.704	0.711	0.681	0.682	0.680	0.688	0.683	0.646	0.639	0.647	0.641	0.640	**0.682**
Manip-ulator	0.674	0.739	0.619	0.617	0.731	0.794	0.696	0.698	0.716	0.692	0.771	0.693	0.688	0.683	0.724	0.608	0.590	0.609	0.586	0.589	**0.676**
Bully	0.674	0.738	0.617	0.616	0.728	0.795	0.696	0.696	0.714	0.691	0.771	0.693	0.686	0.682	0.724	0.609	0.589	0.608	0.587	0.590	**0.675**
S	0.671	0.704	0.665	0.665	0.691	0.682	0.738	0.740	0.687	0.665	0.641	0.662	0.633	0.671	0.644	0.591	0.589	0.590	0.586	0.587	**0.655**
MBRL-1	0.663	0.716	0.676	0.677	0.663	0.662	0.743	0.744	0.676	0.669	0.621	0.628	0.628	0.671	0.626	0.600	0.598	0.601	0.600	0.598	**0.653**
GF	0.670	0.637	0.626	0.626	0.689	0.715	0.746	0.748	0.651	0.659	0.707	0.673	0.612	0.666	0.654	0.574	0.558	0.571	0.557	0.555	**0.645**
Manip.-GF	0.669	0.637	0.627	0.626	0.689	0.717	0.746	0.741	0.646	0.658	0.705	0.672	0.607	0.665	0.654	0.574	0.557	0.571	0.555	0.555	**0.644**
EEE	0.635	0.664	0.640	0.630	0.639	0.648	0.697	0.694	0.639	0.633	0.625	0.636	0.628	0.641	0.631	0.592	0.586	0.596	0.583	0.592	**0.631**
GIGA-WoLF	0.643	0.676	0.629	0.631	0.652	0.643	0.675	0.673	0.650	0.658	0.610	0.622	0.611	0.632	0.616	0.543	0.548	0.550	0.546	0.546	**0.618**
MBRL-2	0.645	0.694	0.660	0.660	0.638	0.592	0.723	0.723	0.651	0.632	0.557	0.589	0.560	0.605	0.567	0.521	0.522	0.522	0.522	0.522	**0.605**
M-Qubed	0.627	0.633	0.624	0.623	0.653	0.605	0.703	0.704	0.638	0.611	0.582	0.602	0.582	0.636	0.578	0.538	0.536	0.535	0.533	0.534	**0.604**
WoLF-PHC	0.608	0.616	0.608	0.608	0.609	0.601	0.658	0.655	0.605	0.619	0.585	0.590	0.595	0.591	0.587	0.573	0.570	0.569	0.569	0.569	**0.599**
S-learn	0.600	0.637	0.612	0.611	0.644	0.616	0.678	0.678	0.626	0.637	0.576	0.610	0.576	0.621	0.585	0.524	0.521	0.523	0.520	0.520	**0.596**
QL	0.623	0.653	0.633	0.635	0.622	0.582	0.689	0.690	0.628	0.607	0.561	0.567	0.566	0.601	0.558	0.526	0.525	0.526	0.523	0.526	**0.592**
WMA	0.495	0.504	0.495	0.496	0.503	0.489	0.555	0.555	0.507	0.482	0.487	0.486	0.492	0.488	0.487	0.484	0.484	0.485	0.484	0.485	**0.497**
sFP	0.493	0.506	0.496	0.496	0.502	0.494	0.555	0.556	0.505	0.479	0.486	0.486	0.490	0.490	0.486	0.482	0.482	0.484	0.482	0.482	**0.497**
Exp3	0.460	0.462	0.460	0.460	0.474	0.461	0.521	0.520	0.472	0.464	0.458	0.461	0.459	0.464	0.462	0.464	0.459	0.463	0.461	0.461	**0.468**
Random	0.452	0.460	0.456	0.453	0.468	0.458	0.515	0.516	0.472	0.462	0.457	0.457	0.455	0.460	0.459	0.460	0.455	0.460	0.455	0.458	**0.464**
CJAL	0.452	0.457	0.456	0.456	0.470	0.454	0.513	0.514	0.472	0.461	0.458	0.457	0.454	0.460	0.459	0.460	0.455	0.460	0.456	0.458	**0.464**

Table 6: The average per-round payoffs for each pairing across all 100-round games. The values show the average payoff obtained by the algorithm listed on the row when paired with the algorithm listed in the column row. As points of comparison, the average Nash bargaining solution over all games was 0.777, and the average payoff over all cells of all games to each player was 0.458.

	S++	Bully	Manip-ulator	S	MBRL 2	MBRL 1	FP	EEE	Manip-GF	WoLF-PHC	GF	CJAL	S-learn	GIGA WoLF	QL	M-Qubed	WMA	sFP	Exp3	Rand-om	Ave
S++	0.765	0.703	0.734	0.785	0.785	0.794	0.778	0.765	0.791	0.740	0.775	0.747	0.778	0.731	0.747	0.717	0.686	0.685	0.628	0.622	**0.738**
Bully	0.806	0.615	0.762	0.809	0.817	0.821	0.740	0.793	0.789	0.726	0.695	0.734	0.787	0.696	0.765	0.709	0.696	0.695	0.615	0.587	**0.733**
Manip-ulator	0.766	0.676	0.716	0.782	0.800	0.804	0.754	0.772	0.765	0.736	0.750	0.745	0.771	0.727	0.759	0.704	0.695	0.694	0.624	0.603	**0.732**
S	0.737	0.701	0.713	0.760	0.709	0.733	0.739	0.723	0.785	0.690	0.774	0.709	0.750	0.704	0.690	0.692	0.657	0.656	0.601	0.592	**0.706**
MBRL-2	0.711	0.699	0.700	0.760	0.699	0.711	0.749	0.714	0.772	0.673	0.769	0.698	0.747	0.687	0.712	0.718	0.667	0.666	0.627	0.622	**0.705**
MBRL-1	0.693	0.700	0.706	0.714	0.700	0.709	0.747	0.709	0.779	0.696	0.769	0.725	0.748	0.697	0.680	0.681	0.682	0.682	0.640	0.634	**0.705**
FP	0.693	0.685	0.708	0.723	0.702	0.701	0.695	0.703	0.754	0.713	0.725	0.729	0.721	0.714	0.705	0.688	0.693	0.692	0.649	0.643	**0.702**
EEE	0.699	0.688	0.699	0.719	0.710	0.717	0.718	0.702	0.765	0.688	0.758	0.708	0.728	0.682	0.698	0.692	0.669	0.668	0.629	0.622	**0.698**
Manip-GF	0.727	0.665	0.691	0.736	0.737	0.732	0.681	0.711	0.760	0.659	0.764	0.671	0.718	0.685	0.684	0.683	0.640	0.639	0.593	0.575	**0.688**
WoLF-PHC	0.682	0.667	0.690	0.696	0.682	0.686	0.685	0.688	0.739	0.692	0.712	0.698	0.681	0.681	0.668	0.665	0.672	0.673	0.629	0.624	**0.680**
GF	0.747	0.626	0.711	0.755	0.745	0.737	0.638	0.725	0.763	0.638	0.746	0.643	0.710	0.661	0.684	0.684	0.630	0.629	0.577	0.555	**0.680**
CJAL	0.682	0.666	0.687	0.708	0.686	0.672	0.671	0.678	0.744	0.668	0.707	0.678	0.693	0.668	0.672	0.672	0.664	0.662	0.628	0.624	**0.677**
S-learn	0.716	0.685	0.702	0.746	0.687	0.702	0.727	0.708	0.731	0.682	0.722	0.703	0.744	0.704	0.639	0.645	0.618	0.618	0.524	0.516	**0.676**
GIGA-WoLF	0.691	0.636	0.678	0.696	0.673	0.685	0.687	0.684	0.703	0.684	0.679	0.708	0.675	0.679	0.653	0.643	0.641	0.642	0.567	0.551	**0.663**
QL	0.686	0.679	0.687	0.716	0.635	0.639	0.711	0.672	0.745	0.638	0.731	0.638	0.720	0.660	0.637	0.637	0.604	0.605	0.548	0.542	**0.656**
M-Qubed	0.678	0.636	0.651	0.720	0.634	0.636	0.654	0.658	0.727	0.642	0.714	0.637	0.729	0.638	0.635	0.641	0.601	0.600	0.541	0.535	**0.645**
WMA	0.638	0.638	0.650	0.651	0.635	0.635	0.639	0.646	0.696	0.631	0.689	0.642	0.630	0.630	0.622	0.622	0.620	0.621	0.591	0.587	**0.636**
sFP	0.638	0.637	0.650	0.649	0.632	0.640	0.641	0.641	0.696	0.634	0.689	0.640	0.630	0.623	0.621	0.622	0.619	0.619	0.590	0.587	**0.635**
Exp3	0.507	0.508	0.521	0.514	0.510	0.512	0.518	0.518	0.563	0.508	0.568	0.513	0.494	0.511	0.496	0.495	0.501	0.502	0.490	0.489	**0.512**
Random	0.453	0.455	0.471	0.468	0.455	0.459	0.458	0.466	0.510	0.455	0.514	0.459	0.457	0.454	0.458	0.457	0.459	0.458	0.457	0.457	**0.464**

Table 7: The average per-round payoffs for each pairing across all 1000-round games. The values show the average payoff obtained by the algorithm listed on the row when paired with the algorithm listed in the column row. As points of comparison, the average Nash bargaining solution over all games was 0.777, and the average payoff over all cells of all games to each player was 0.458.

	S++	Bully	Manip-ulator	M-Qubed	MBRL 2	QL	EEE	S	MBRL 1	WoLF-PHC	CJAL	FP	S-learn	WMA	sFP	Manip-GF	GF	Exp3	GIGA-WoLF	Rand-om	Ave
S++	0.777	0.709	0.754	0.763	0.801	0.784	0.787	0.802	0.800	0.763	0.776	0.782	0.799	0.740	0.737	0.801	0.780	0.721	0.733	0.625	**0.762**
Bully	0.829	0.615	0.797	0.823	0.828	0.825	0.824	0.829	0.828	0.741	0.751	0.740	0.805	0.737	0.737	0.807	0.695	0.731	0.697	0.587	**0.761**
Manip-ulator	0.779	0.689	0.742	0.784	0.812	0.786	0.788	0.797	0.808	0.755	0.764	0.758	0.786	0.740	0.739	0.787	0.763	0.723	0.732	0.607	**0.757**
M-Qubed	0.776	0.706	0.722	0.768	0.730	0.769	0.744	0.817	0.717	0.754	0.744	0.742	0.815	0.715	0.715	0.792	0.778	0.704	0.721	0.637	**0.743**
MBRL-2	0.733	0.708	0.717	0.785	0.736	0.788	0.742	0.804	0.743	0.719	0.741	0.762	0.785	0.730	0.726	0.785	0.779	0.723	0.708	0.642	**0.743**
QL	0.751	0.708	0.720	0.761	0.718	0.764	0.738	0.798	0.716	0.747	0.738	0.738	0.812	0.711	0.713	0.793	0.777	0.704	0.713	0.638	**0.738**
EEE	0.732	0.709	0.723	0.768	0.758	0.765	0.730	0.769	0.740	0.721	0.749	0.732	0.783	0.721	0.723	0.788	0.779	0.717	0.699	0.640	**0.737**
S	0.751	0.708	0.725	0.728	0.732	0.740	0.743	0.778	0.751	0.725	0.734	0.745	0.772	0.715	0.716	0.793	0.780	0.705	0.713	0.594	**0.732**
MBRL-1	0.703	0.707	0.716	0.739	0.725	0.738	0.725	0.733	0.720	0.721	0.760	0.754	0.780	0.727	0.728	0.787	0.777	0.721	0.705	0.643	**0.730**
WoLF-PHC	0.705	0.687	0.719	0.718	0.718	0.721	0.718	0.734	0.712	0.731	0.741	0.707	0.723	0.734	0.738	0.769	0.732	0.727	0.697	0.642	**0.719**
CJAL	0.713	0.689	0.717	0.729	0.721	0.724	0.711	0.746	0.698	0.708	0.724	0.697	0.732	0.727	0.726	0.776	0.730	0.709	0.692	0.643	**0.716**
FP	0.696	0.685	0.713	0.714	0.706	0.718	0.708	0.734	0.703	0.734	0.757	0.695	0.732	0.727	0.726	0.761	0.725	0.720	0.715	0.643	**0.716**
S-learn	0.742	0.695	0.723	0.703	0.717	0.700	0.733	0.773	0.729	0.731	0.740	0.747	0.769	0.702	0.700	0.741	0.728	0.679	0.716	0.516	**0.714**
WMA	0.715	0.687	0.712	0.729	0.707	0.730	0.714	0.732	0.699	0.710	0.719	0.700	0.716	0.713	0.718	0.748	0.730	0.719	0.693	0.642	**0.712**
sFP	0.715	0.687	0.711	0.733	0.705	0.728	0.716	0.732	0.702	0.708	0.716	0.703	0.715	0.710	0.713	0.746	0.731	0.721	0.686	0.642	**0.711**
Manip-GF	0.738	0.674	0.709	0.730	0.744	0.726	0.730	0.749	0.734	0.677	0.697	0.690	0.728	0.667	0.665	0.768	0.770	0.659	0.692	0.580	**0.706**
GF	0.768	0.625	0.735	0.758	0.752	0.749	0.756	0.772	0.742	0.650	0.654	0.638	0.716	0.645	0.645	0.769	0.746	0.643	0.662	0.554	**0.699**
Exp3	0.703	0.679	0.698	0.717	0.687	0.716	0.698	0.716	0.687	0.696	0.701	0.690	0.701	0.689	0.691	0.729	0.724	0.696	0.679	0.633	**0.696**
GIGA-WoLF	0.703	0.637	0.690	0.701	0.702	0.707	0.708	0.708	0.696	0.719	0.734	0.687	0.686	0.701	0.706	0.710	0.679	0.699	0.682	0.553	**0.690**
Random	0.454	0.455	0.473	0.458	0.456	0.458	0.463	0.468	0.458	0.458	0.460	0.455	0.458	0.459	0.458	0.508	0.514	0.458	0.452	0.458	**0.464**

Table 8: The average per-round payoffs for each pairing across all 50,000-round games. The values show the average payoff obtained by the algorithm listed on the row when paired with the algorithm listed in the column row. As points of comparison, the average Nash bargaining solution over all games was 0.777, and the average payoff over all cells of all games to each player was 0.458.

C Algorithms: Descriptions of S++ and S#

The results of the empirical evaluation of algorithms in the previous section (Appendix B) demonstrate that S++ has performance properties in repeated games that give us reason to believe it could learn to effectively interact with people in repeated games. This led us to run three different user studies, in which we paired S++ and an extended version of the algorithm called S# (pronounced 'S sharp') with people in various repeated games. S# builds onto S++ by combining it with mechanisms to generate and to respond to communication signals. Both algorithms take as input a description of the RSG.

In this section, we first review S++ in detail. We then describe how S++ is extended to form S#.

C.1 S++

S++ was originally designed for repeated normal-form games [1], but has subsequently been generalized to repeated stochastic games (RSGs) [2][2]. S++ is an expert algorithm[3]. In each round t, an expert algorithm (employed by player i) selects an expert ϕ from a set of experts Φ_i. This expert then dictates the strategy executed by player i in round t. The set Φ_i essentially reduces the strategy space of the game to a handful of high-level strategies. Each expert defines a strategy over the entire state-space of the game. Thus, rather than learning a separate policy in each state $\sigma \in \Sigma$, the player must learn which high-level strategies or algorithms in Φ_i are the most profitable in the current scenario.

Thus, S++ is defined by two things: (1) its method for automatically generating the set of experts Φ_i from the description of the game, and (2) its mechanism for selecting which expert to execute in each round. We discuss both aspects of S++ in turn.

C.1.1 Computing the Set of Experts Φ_i

Littman and Stone [39] classified algorithms for repeated games into two categories: *leaders* and *followers*[4], each of which is successful in different circumstances. A good set of experts must contain both leader and follower experts. Such experts can be computed automatically for any RSG [2]. For simplicity, our review focuses only on normal-form games, borrowing heavily from the work of Crandall [1]. A method for computing a similar set of experts for multi-stage RSGs has also been defined by Crandall [2].

Leader Experts. Leader strategies are designed to encourage the associate to play its portion of a *target solution* [39], which is a sequence of joint actions that are played repetitively. Each solution χ produces an expected payoff profile $\mathbf{v}(\chi) = (v_i(\chi), v_{-i}(\chi))$. For example, in the Prisoner's Dilemma (Table 1), the solution $\chi = <(c, C), (d, C)>$ indicates that the column player always cooperates while the row player alternates between cooperating and defecting. This produces the expected payoff profile $\mathbf{v}(\chi) = (80, 30)$.

S++ derives a separate leader expert corresponding to each member of a set Ω of target solutions. Each leader expert is coupled with a trigger-mechanism that incentivizes the partner to play its portion of χ. When its partner has conformed with χ or has not profited by deviating from it, the leader expert plays its portion of χ. On the other hand, if the partner has increased its payoffs by deviating from χ, the leader plays its

[2]The generalization of S++ to RSGs was named MEGA-S++ [2]. MEGA-S++ is identical to S++ except that the set of experts defined for MEGA-S++ are designed to produce strategies in general (multi-stage) RSGs rather than just normal-form games. In this work, we simply refer to both versions as S++.

[3]We adopt the notation used in the published literature [1, 2]. This notation differs from the notation used in the main body of this paper, which was used in attempt to be more assessable to a general audience.

[4]Crandall [2] identified a third category called *preventative strategies*. Preventative strategies are sometimes successful when $|S| > 1$.

No.	Target solution (χ)	$v_1(\chi)$	Strategy
1	$<(c,C),(d,C)>$	80	If $g_2^t > 0$, play π_i^{attack}. Otherwise, play own portion of the current joint action in the sequence χ.
2	$<(c,C)>$	60	
3	$<(d,C),(c,D)>$	50	
4	$<(c,C),(d,D)>$	40	
5	$<(c,C),(c,D)>$	30	
6	$<(d,D)>$	20	

Table 9: Leader experts generated for player 1 in the Prisoner's Dilemma.

attack policy, given by

$$\pi_i^{\text{attack}} = \arg\min_{\pi_i} \max_{\pi_{-i}} u_{-i}(\pi_i, \pi_{-i}), \tag{3}$$

where $u_{-i}(\pi_i, \pi_{-i})$ is player $-i$'s expected payoff when the players use strategies π_i and π_{-i}, respectively.

Formally, each leader expert keeps track of a guilt parameter g_{-i}^t that defines how much its associate has benefited from deviating from χ. Initially, $g_{-i}^1 = 0$, and is subsequently updated as follows:

$$g_{-i}^{t+1} \leftarrow \begin{cases} 0 & \text{if } a_{-i}^t = b_{-i}^t \text{ and } g_{-i}^t = 0 \\ \max\left(0, g_{-i}^t + r_{-i}(\mathbf{a}^t) - v_{-i} + \delta_t\right) & \text{otherwise} \end{cases} \tag{4}$$

where b_{-i}^t is player $-i$'s current action defined by the target solution χ and δ_t is a small nonnegative value. We use $\delta_t = 0.1$ if $g_{-i}^t = 0$ and $\delta_t = 0.0$ otherwise. When $g_{-i}^t > 0$ (i.e., the partner is guilty of deviating), the leader expert plays its attack policy π_i^{attack}. When $g_{-i}^t = 0$, the leader expert plays its portion of χ.

Table 9 lists possible leader experts (for a particular set Ω) for the Prisoner's Dilemma. Depending on the implementation details (see Table 11), some of these experts are not always included in Φ_i.

Follower Experts. Followers play a best response to the strategy they ascribe to their associate. We include three kinds of follower experts in the set Φ_i, each of which estimates its associate's strategy differently. The first expert models its associate using the fictitious-play assessment, conditioned on the last executed joint action. Let $\kappa_{-i}^t(\mathbf{a}, a_{-i})$ be the number of times that player $-i$ has played a_{-i} given the previous joint action $\mathbf{a}$ up to round t. Then, the estimated probability that player $-i$ plays a_{-i} is

$$\gamma_{-i}^t(\mathbf{a}, a_{-i}) = \frac{\kappa_{-i}^t(\mathbf{a}, a_{-i})}{\sum_{b_{-i} \in A_{-i}} \kappa_{-i}^t(\mathbf{a}, b_{-i})}. \tag{5}$$

The expert then computes the automaton that maximizes the player's future discounted reward (we use discount factor $\gamma = 0.95$) given γ_{-i}^t. We denote this expert as ϕ_i^*.

A second follower expert, which we call ϕ_i^{mm}, assumes that its associate is seeking to exploit it. Under this assumption, its best response is to play its maximin strategy, given by

$$\pi_i^{\text{mm}} = \arg\max_{\pi_i} \min_{\pi_{-i}} u_i(\pi_i, \pi_{-i}). \tag{6}$$

Finally, the associate could also be implementing a leader strategy. Thus, we also include in Φ_i a follower expert for each target solution $\chi \in \Omega$. Such experts always play their portion of the target solution χ. If a joint action in the solution was not played in the previous round, these experts randomly select a joint action from the solution sequence and play the players's corresponding action.

Table 10 lists eight potential follower experts that are computed for the Prisoner's Dilemma.

No.	Target solution (χ)	$v_1(\chi)$	Strategy
1	(none)	$\mu_1(br_1(\gamma_2^t), \gamma_2^t)$	$br_1(\gamma_2^t)$
2	(none)	$v_1^{\text{mm}} = 20$	π_1^{mm}
3	$<(c,C),(d,C)>$	80	
4	$<(c,C)>$	60	If $\mathbf{a}^{t-1}$ is in χ, play own portion of the next
5	$<(d,C),(c,D)>$	50	joint action in χ. Otherwise, randomly select a
6	$<(c,C),(d,D)>$	40	joint action from χ and play own portion of that
7	$<(c,C),(c,D)>$	30	joint action.
8	$<(d,D)>$	20	

Table 10: Follower experts generated for player 1 in the Prisoner's Dilemma. $br_1(\gamma_2^t)$ denotes player 1's best response to the fictitious-play assessment γ_2^t. v_1^{mm} is player 1's maximin value.

C.1.2 Selecting an Expert

S++ selects which expert to follow in each round t using aspiration learning [62, 34, 41], along with a method for pruning the set Φ_i[5]. Aspiration learning has proven to be effective when learning to select among learning experts [67]. In aspiration learning, the player learns an achievable *aspiration level*, and simultaneously searches for an expert that produces payoffs that satisfy this threshold. Let α_i^t denote player i's aspiration level in round t. Initially, α_i^1 is initialized to a high value [68], and the set H_c, the set of joint actions played in the current cycle so far, is initialized to a random joint action. S++ then randomly selects some expert $\phi_i \in \Phi_i$, and follows ϕ_i until a particular joint action is repeated[6]. This is determined by evaluating whether the last joint-action played is already in H_c.

Once a cycle completes, α_i^t is updated as follows:

$$\alpha_i^t \leftarrow (\lambda_i)^{|H_c|}\alpha_i^{t-|H_c|} + (1 - (\lambda_i)^{|H_c|})\bar{r}_i^t, \tag{7}$$

where $\lambda_i \in (0,1)$ is the learning rate and $\bar{r}_i^t$ is the average payoff obtained by player i in the cycle. After updating α_i^t, a new expert is selected as follows:

$$\phi_i^t \leftarrow \begin{cases} \phi_i^{(t-|H_c|)} & \text{with prob. } f(\alpha_i^t, \bar{r}_i^t) \\ \text{random}(\Phi_i'(t)) & \text{otherwise} \end{cases} \tag{8}$$

Here, random$(\Phi_i'(t))$ denotes a random selection from the set $\Phi_i'(t) \subseteq \Phi_i'$ (described below), and $f(\alpha_i^t, \bar{r}_i^t)$ is the player's inertia, which specifies how likely the player is to select the expert that it played in the previous episode if $\bar{r}_i^t$ fails to meet α_i^t. If not, it randomly selects a new expert with probability $f(\alpha_i^t, \bar{r}_i^t)$. H_c is then reset to include only the last joint solution played, and a new cycle begins.

The set Φ_i can be rather large. Furthermore, some of the experts in Φ_i are unlikely to produce outcomes that meet the player's current aspiration level α_i^t. As such, the selection of these experts is unlikely to produce effective behavior and, thus, should be avoid. Thus, S++ prunes the set of selectable experts such that only experts that can potentially meet the aspiration level are considered. Formally, let $z_i^t(\phi)$ be the

[5] S++ is the essentially the same for multi-stage RSGs, the only difference being that the joint-action $\mathbf{a}^t$ must be replaced by an assessment of the joint strategy played throughout the round.

[6] For simplicity, we noted that S++ plays the selected expert for m rounds in Figure 2 of the main paper. This is technically correct, though the length m is variable as explained herein.

Algorithm 1 S++ for normal-form games.

Input:
Description of the game
Initialize:
Compute the set of experts Φ_i
Initialize the aspiration level α_i
$\mathbf{a}^0 \leftarrow \text{random}(\text{A})$
$H_c \leftarrow \{\mathbf{a}_0\}$, $t \leftarrow 1$, and $\bar{r}_i \leftarrow 0$
Select an initial expert $\phi_i^t \in \Phi_i$
repeat
Select and execute action a_i^t (as dictated by ϕ_i^t)
Observe $\mathbf{a}^t$ and r_i^t
$\bar{r}_i \leftarrow \bar{r}_i + r_i^t$
if $\mathbf{a}_t \in H_c$ **then**
$\bar{r}_i = \bar{r}_i / |H_c|$
Update α_i using Eq. (7)
Update each expert $\phi \in \Phi_i$ as necessary
Select ϕ_i^{t+1} using Eq. (8)
$H_c \leftarrow \varnothing$ and $\bar{r}_i \leftarrow 0$
else
$\phi_i^{t+1} = \phi_i^t$
end if
$H_c \leftarrow H_c \cup \{\mathbf{a}_t\}$
$t \leftarrow t + 1$
until Game Over

estimated potential (highest expected payoff) to be produced by expert $\phi \in \Phi_i$ at time t. Then, $\Phi_i'(t)$ is given by:

$$\Phi_i'(t) = \{\phi \in \Phi_i : z_i^t(\phi) \geq \alpha_i^t\}. \tag{9}$$

The potentials for each expert is computed as in the work by Crandall [1].

Pseudo-code for the complete S++ algorithm is given in Algorithm 1.

C.1.3 Parameter Choices

S++ has several parameters that can be varied to address various trade-offs. Implementation choices include (1) deciding how many leader and follower experts (i.e., the size of the set Ω) to include in the set Φ_i, (2) determining the inertia function (see Eq. 8), (3) determining which expert to follow first, (4) setting the initial aspiration level (α_i^1), and (5) setting the learning rate (λ_i). We briefly discuss these implementation choices.

- *Number of leader and follower experts.* Ideally, the number of target solutions should include a wide variety of different strategies, but be small enough that S++ can find a good expert within a small amount of time. Algorithm designers must also decide whether target solutions that are not enforceable when one's partner tends to forget events in the more distant past should be included in Φ_i.

- *Inertia function.* Inertia was previously considered for use in aspiration learning by Karandikar et al. [34]. The inertia function defines how likely the algorithm is to re-select the last expert when it

does not produce sufficiently high payoffs in a cycle. In the initial work by Crandall [1], the inertia function was:

$$f(\alpha_i^t, \bar{r}_i^t) = \min(1, \left(\bar{r}_i^t / \alpha_i^t\right)^{|H_c|}). \tag{10}$$

However, it may be desirable to give the algorithm more memory of past events, thus making the player more likely to continue to follow an expert that has yielded better results in the past. To capture this idea, an alternative inertia function can be used, defined as follows:

$$\begin{aligned} \bar{v}_i^0 &= 0 \\ \bar{v}_i^t &= \beta \bar{r}_i^t + (1-\beta)\bar{v}_i^{t-1} \\ \beta &= \max\left(0.2, \frac{1}{\kappa_i^t(\phi_i^t)}\right) \\ f(\alpha_i^t, \bar{r}_i^t) &= \min(1, \left(\max(\bar{r}_i^t, \bar{v}_i^t) / \alpha_i^t\right)^{|H_c|}). \end{aligned} \tag{11}$$

We also set $f(\alpha_i^t, \bar{r}_i^t)$ to unity if both players always adhered to the target solution during the cycle.

- *Initial expert.* In initial implementations, S++, after taking a single random action, selected an expert using its standard selection. In some subsequent implementations, $\phi_i^\star$ was automatically selected as an initial exploratory behavior.

- *Initial aspiration level.* In aspiration learning, higher initial aspiration levels tend to produce better long-term outcomes [41]. In initial implementations of S++, α_i^t was initialized to the value of the expert with the highest initial potential. In later studies, we set the initial aspiration level based on a fair, Pareto optimal, target solution. Higher initial aspirations make the algorithm more selective in early rounds, sometimes causing it to "insist" on unequal solutions in its favor. On the other hand, setting the aspiration level to what is expected in a fair solution tends to lead to faster convergence to cooperative solutions when one's partner is inclined to cooperate.

- *Learning rate.* A faster learning rate (i.e., lower λ_i) causes S++ to lower its aspiration level more quickly. This helps the algorithm to converge faster, but makes the algorithm more vulnerable to accepting lower payoffs than it otherwise might obtain [68]. Slower learning rates (i.e., high λ_i) cause S++ to adapt slower, but also make it more likely to find more profitable, Pareto optimal, outcomes.

Over the course of the studies that we have conducted, we have made small adjustments to these parameters as we have sought to develop an algorithm that consistently learns to cooperate with people (though we have observed that these changes often appear to have had little or no impact on the results in practice). Table 11 summarizes the implementation choices used in each study.

C.2 S#

The results presented in Appendix B show that S++ is an effective learning algorithm when paired with itself (i.e., self play) and many other AI algorithms. It typically learns to cooperate in self play within tens of rounds. When beneficial, it also tends to cooperate with other learning algorithms that are inclined to cooperate. These behavioral attributes stand in contrast to those of many other machine-learning algorithms, which often either fail to learn to cooperate in many games in self play, or only do so after many rounds

The set Ω	
Axelrod-style tournaments (Appendix B)	As described in [1]
User Study 1 (Appendix D)	As described in [1]
User Study 2 (Appendix E)	As described in [2]
User Study 3 (Appendix F)	As described in [2]; however, leader strategies were also constructed for target solutions that were unenforceable given an associate that only remembered a single joint action.
Inertia function $f(\alpha_i^t, \bar{r}_i^t)$	
Axelrod-style tournaments (Appendix B)	Set according to Eq. (10)
User Study 1 (Appendix D)	Set according to Eq. (10)
User Study 2 (Appendix E)	Set according to Eq. (11)
User Study 3 (Appendix F)	Set according to Eq. (11)
Initial expert ϕ_i^1	
Axelrod-style tournaments (Appendix B)	Selects a random action in round 1, then selects randomly from $\Phi_i'(t)$
User Study 1 (Appendix D)	Selects a random action in round 1, then selects randomly from $\Phi_i'(t)$
User Study 2 (Appendix E)	ϕ_i^*
User Study 3 (Appendix F)	ϕ_i^*
Initial aspiration level α_i^1	
Axelrod-style tournaments (Appendix B)	$\alpha_i^1 = \max_{\phi \in \Phi_i} z_i^1(\phi)$
User Study 1 (Appendix D)	$\alpha_i^1 = \max_{\phi \in \Phi_i} z_i^1(\phi)$
User Study 2 (Appendix E)	$\alpha_i^1 = \max_{\phi \in \Phi_i} \min\left(u_i(\phi, br_{-i}(\phi)), u_{-i}(\phi, br_{-i}(\phi))\right)$
User Study 3 (Appendix F)	$\alpha_i^1 = \max_{\phi \in \Phi_i} \min\left(u_i(\phi, br_{-i}(\phi)), u_{-i}(\phi, br_{-i}(\phi))\right)$
Learning rate λ_i	
Axelrod-style tournaments (Appendix B)	$\lambda_i = 0.99$
User Study 1 (Appendix D)	$\lambda_i = 1 - \frac{0.04}{\lvert\Phi_i'(t)\rvert}$
User Study 2 (Appendix E)	$\lambda_i = 0.99$
User Study 3 (Appendix F)	$\lambda_i = 0.99$

Table 11: Parametric choices for S++ that were used in the four investigations presented in this paper.

Speech ID	Text	Speech ID	Text
0	Do as I say, or I'll punish you.	10	We can both do better than this.
1	I accept your last proposal.	11	Curse you.
2	I don't accept your proposal.	12	You betrayed me.
3	That's not fair.	13	You will pay for this!
4	I don't trust you.	14	In your face!
5	Excellent!	15	Let's always play <action pair>.
6	Sweet. We are getting rich.	16	This round, let's play <action pair>.
7	Give me another chance.	17	Don't play <action>.
8	Okay. I forgive you.	18	Let's alternate between <action pair>
9	I'm changing my strategy.		and <action pair>.

Table 12: List of speech acts available to the players in the third user study.

of interaction. Our results (see Appendices D–F) also show that even humans do not consistently cooperate with each other in the absence of communication. Thus, in regards to learning to cooperate (when beneficial), S++ is relatively prolific.

Despite these performance properties, our results also show that S++ fails to consistently elicit cooperation from a human partner (see Appendices D and E). However, since humans appear to cooperate with each other much more consistently when they are able to signal to each other (Appendix E), we consider adding to S++ the ability to communicate via cheap talk. That is, we use S++ to generate speech acts. We also use speech acts from S++'s partner to improve the way the algorithm selects experts. We call the resulting algorithm S# (pronounced 'S-sharp').

Most machine-learning algorithms have representations that are difficult to understand. Furthermore, these representations are very low-level, which makes them largely incapable of generating signals at a level that will be beneficial in interactions with people. On the other hand, S++ has a hierarchical internal representation which makes it possible to use the algorithm to generate and interpret cheap talk at a level conducive to interaction with a human partner. This capability means that S++ can potentially use cheap talk to help establish cooperative relationships with people.

The ability to engage in two-way cheap talk (so as to enhance one's own payoffs) requires two skills: (1) the ability to generate cheap talk and (2) the ability to act on the cheap talk of others. In this section, we describe how these abilities are addressed in S#.

C.2.1 Generating Cheap Talk

The speech-generation system used by S# consists of three parts: (1) a language (i.e., a set of speech acts), (2) a method for selecting speech acts, and (3) a method for determining when to speak and when to remain silent. We define each in turn for S# in normal form games. A similar (but preliminary) method is defined for multi-stage RSGs in Appendix E.

A Set of Speech Acts. The language used by S# in the third user study (Appendix F) consisted of the set of phrases (i.e., *speech acts*) shown in Table 12. These speech acts can be combined to propose plans, express sentiments, etc. The same set of phrases is used regardless of the game scenario, though the description of the game must include actions labels, as some speech acts utilize these action labels. We also limited participants to the same set of speech acts, which allows us to more easily determine how

Symbol	Event descriptions
n	S# selected a new (distinct from the previous cycle) expert to follow.
c	An expert has been executed for a complete cycle.
x	S# accepts its partner's proposed plan of action.
y	S# rejects its partner's proposed plan of action because it does not trust its partner.
z	S# rejects its partner's proposed plan because it does not deem the proposal to be fair.
b	S# believes the players can both receive higher payoffs than what they received in the previous cycle. This event is not generated if the new expert is MBRL.
s	The current expert is *satisfied* with last round's payoff.
f	The current expert *forgives* the other player.
d	The other player *defected* against S#.
g	The other player profited from its defection (and is *guilty*).
p	The expert has *punished* its guilty partner.
u	S# did not succeed in punishing its guilty partner.

Table 13: A list of event symbols and event descriptions. The first six events correspond to master-level (i.e., expert selection) events of S++, while the last six events correspond to events relevant to individual experts.

humans and machines use cheap talk in long-term repeated interactions. With the goal of not overwhelming participants, we kept the set of speech acts small, but sufficiently large to allow them to effectively plan and express a wide range of sentiments[7].

Though small, the speech acts shown in Table 12 cover a range of different ideas a player may want to convey to its partner. For example, speech acts 15-18 provide means to propose desired solutions to an associate. Other speech acts allow the players to voice threats (speech acts 1 and 13), accept or reject proposals (1 and 2), explain decisions to reject proposals (3-4), manage the relationship (7-10), express pleasure (5-6), express displeasure (11-12), and to gloat (14). Combined together, these speech acts provide players with a surprisingly rich communication base.

In user study 3, when players were allowed to engage in cheap talk, each round of the repeated game began with the players independently selecting an ordered set of speech acts (from the prescribed list) that they wished to communicate to their partner. When both players had committed to a set of speech acts (and not before), the speech acts were communicated to the other player both visually and aurally (through a computer voice heard through headsets). The players then simultaneously selected an action. Once both players had selected an action, the joint outcome was displayed to the players, and a new round then began (starting with another round of cheap talk).

Speech Generation. S# uses the internal state of S++ and events in the game to determine which speech acts to voice. Thus, to generate speech acts, a finite state machine with output can be created in which the states are the states of S++, the events are outcomes that affect S#'s decision making, and the output of the finite state machine are speech acts. Because each of these items are defined for arbitrary repeated games, the speech-generation system is also game-independent. Thus, to create a game-independent speech system for S#, we need only define (a) the events of the game relevant to S++, (b) the underlying internal states of S++, and (c) the state transition function, which maps events and internal states to new internal states and speech output.

The events of the game that are relevant to S++ are listed in Table 13. The first six events correspond to

[7]A larger set of speech acts was used for the multi-stage games used in the second user study (Appendix E).

expert selection, while the last six events are events that impact the individual experts. Curiously, this set of events is rather small, which is due to the simple, hierarchical representation of S++. Though the underlying architecture of individual experts can be sophisticated, the dynamics of each expert embodies high-level principles that are understandable to people, and are governed by high-level events.

The internal state of S++ is determined by (a) its aspiration level, (b) the currently selected expert, and (c) the state of the currently selected expert. Due to the hierarchical nature of S++, it is easiest to describe the internal states of S++ in a distributed fashion. Specifically, a finite state machine (FSM) with output is created at both the master level and expert level. A separate FSM is formulated for each type of expert. These FSMs govern S#'s speech output only when a corresponding expert is selected. The master-level FSM governs speech output related to transitions between experts. After each transition in an expert-level FSM, control is transferred to the master-level FSM, which generates appropriate speech (if any) related to changes (or transitions) in the algorithms strategy. It then either returns control to the previous FSM or selects a new FSM (if it selects a new expert). The master and expert FSMs used by S# to produce speech acts are defined by the state transition tables shown in Table 14.

As an example of how S#'s individual experts encode high-level strategies that can be easily articulated to people, consider a pure leader expert that targets the Nash bargaining solution of the game. This leader expert generates speech using the FSM depicted in Table 14b. When initiated, the FSM outputs speech acts 15 and 0 – it proposes a fair plan (indicated by the corresponding joint action) and also issues the threat that its partner must conform with the plan or he/she/it will be punished. As the partner conforms with the proposed target solution thereafter, S# responds by saying '`Excellent`' for several rounds. If mutual cooperation (with the target solution) continues for five rounds, S# says: '`Sweet. We are getting rich`.' It then ceases talking as long as its partner continues to conform with the target solution. If its partner deviates from the target solution, S# responds by saying either '`Curse you`' or '`You betrayed me`.' If the partner benefited from the deviation, S# also adds the phrase: '`You will pay for this`,' and then proceeds to carry out the threat in subsequent rounds. Once it has sufficiently punished its partner, S# says '`I forgive you`,' and repeats the proposed plan and threat. In this way, it uses words to reinforce the understanding that its partner needs to conform with the target solution in order to obtain high payoffs.

When to Talk. While signaling (using, in this case, cheap talk) has great potential to increase cooperation, signaling intended actions exposes a player to the potential of being exploited. Furthermore, the *silent treatment* is often used by humans as a means of punishment and an expression of displeasure. For these two reasons, S# also chooses not to speak under certain circumstances. In this work, we offer a heuristic-based rule for deciding when to speak and when not to speak. This rule seems to work well in practice, though future work could potentially focus on developing alternative rules.

Formally, let b_i^t be the number of times up to round t that player i has been *betrayed* by player $-i$. That is, b_i^t is the number of times up to round t that player $-i$ did not play its part of the target solution proposed by player i. Then, the probability that player i voices speech acts in round t is given by

$$P_i^{SPEAK}(t) \leftarrow \begin{cases} 1 - 0.1b_i^t & \text{if } 0 \le b_i^t \le 4 \\ \frac{2}{5 \cdot 2^{b_i^t - 3}} & \text{if } b_i^t > 4 \end{cases} \tag{12}$$

Note that the choice of whether or not to speak is made at the beginning of the cycle, and is held constant throughout the cycle.

C.2.2 Listening to Cheap Talk

When players can communicate, S++ uses its partner's signals (in this case, speech acts) to alter which it expert it selects. This revised mechanism requires that we address two questions. First, how can S# use its

(a) Master-level state transition table with output

State	State Transitions: Events						Speech Acts (Output): Events					
	x, n	y	z	c	c, n	c, n, b	x, n	y	z	c	c, n	c, n, b
s_0	s_1	s_2	s_2	s_2	s_1	s_1	1	2+4	2+3	ε	9	9+10
s_1	Go to state s_0 of the FSM corresponding to the newly selected expert.						ε	ε	ε	ε	ε	ε
s_2	Return to the current state of the FSM corresponding to the currently selected expert.						ε	ε	ε	ε	ε	ε

(b) Pure leader expert

State	State Transitions: Events							Speech Acts (Output): Events						
	NUL	d	g	p	u	f	s	NUL	d	g	p	u	f	s
s_0	s_1	–	–	–	–	–	–	15+0	–	–	–	–	–	–
s_1	–	s_2	s_7	–	–	–	s_2	–	$r(\{11,12\})$	$r(\{11,12\})$+13	–	–	–	5
s_2	–	s_2	s_7	–	–	–	s_3	–	$r(\{11,12\})$	$r(\{11,12\})$+13	–	–	–	5
s_3	–	s_2	s_7	–	–	–	s_4	–	$r(\{11,12\})$	$r(\{11,12\})$+13	–	–	–	5
s_4	–	s_2	s_7	–	–	–	s_5	–	$r(\{11,12\})$	$r(\{11,12\})$+13	–	–	–	ε
s_5	–	s_2	s_7	–	–	–	s_6	–	$r(\{11,12\})$	$r(\{11,12\})$+13	–	–	–	6
s_6	–	s_2	s_7	–	–	–	s_6	–	$r(\{11,12\})$	$r(\{11,12\})$+13	–	–	–	ε
s_7	–	–	–	s_7	s_7	s_8	–	–	–	–	14	ε	14+8	–
s_8	s_3	–	–	–	–	–	–	15+0	–	–	–	–	–	–

(c) Alternating leader expert

State	State Transitions: Events							Speech Acts (Output): Events						
	NUL	d	g	p	u	f	s	NUL	d	g	p	u	f	s
s_0	s_1	–	–	–	–	–	–	18+16+0	–	–	–	–	–	–
s_1	–	s_2	s_7	–	–	–	s_2	–	$r(\{11,12\})$+16	$r(\{11,12\})$+13	–	–	–	5+16
s_2	–	s_2	s_7	–	–	–	s_3	–	$r(\{11,12\})$+16	$r(\{11,12\})$+13	–	–	–	5+16
s_3	–	s_2	s_7	–	–	–	s_4	–	$r(\{11,12\})$+16	$r(\{11,12\})$+13	–	–	–	5+16
s_4	–	s_2	s_7	–	–	–	s_5	–	$r(\{11,12\})$+16	$r(\{11,12\})$+13	–	–	–	16
s_5	–	s_2	s_7	–	–	–	s_6	–	$r(\{11,12\})$+16	$r(\{11,12\})$+13	–	–	–	6
s_6	–	s_2	s_7	–	–	–	s_6	–	$r(\{11,12\})$+16	$r(\{11,12\})$+13	–	–	–	ε
s_7	–	–	–	s_7	s_7	s_8	–	–	–	–	14	ε	14+8	–
s_8	s_3	–	–	–	–	–	–	18+16+0	–	–	–	–	–	–

(d) Pure follower expert

State	State Transitions: Events			Speech Acts (Output): Events		
	NUL	d	s	NUL	d	s
s_0	s_1	–	–	15	–	–
s_1	–	s_2	s_2	–	$r(\{11,12\})$	5
s_2	–	s_2	s_3	–	$r(\{11,12\})$	5
s_3	–	s_2	s_4	–	$r(\{11,12\})$	5
s_4	–	s_2	s_5	–	$r(\{11,12\})$	ε
s_5	–	s_2	s_6	–	$r(\{11,12\})$	6
s_6	–	s_2	s_6	–	$r(\{11,12\})$	ε

(e) Alternating follower expert

State	State Transitions: Events			Speech Acts (Output): Events		
	NUL	d	s	NUL	d	s
s_0	s_1	–	–	18+16	–	–
s_1	–	s_2	s_2	–	$r(\{11,12\})$+16	5+16
s_2	–	s_2	s_3	–	$r(\{11,12\})$+16	5+16
s_3	–	s_2	s_4	–	$r(\{11,12\})$+16	5+16
s_4	–	s_2	s_5	–	$r(\{11,12\})$+16	16
s_5	–	s_2	s_6	–	$r(\{11,12\})$+16	6
s_6	–	s_2	s_6	–	$r(\{11,12\})$+16	ε

(f) FSM for the MBRL expert.

State	State Transitions: Events		Speech Acts (Output): Events	
	¬s	s	¬s	s
s_0	s_0	s_0	–	5

(g) FSM for the maxmin expert.

State	State Transitions: Events			Speech Acts (Output): Events		
	NUL	¬s	s	NUL	¬s	s
s_0	s_1	–	–	4	–	–
s_1	–	s_1	s_1	–	–	–

Table 14: Finite state machines (FSMs) with output define the speech acts generated by S#. The master-level FSM (a) is executed beginning in state s_0 after each transition in an expert-level FSM (b–g). The master-level FSM returns control to an expert-level FSM once a terminal state is reached (s_1 and s_2). If state s_1 is reached in the master-level FSM, control over speech output is returned to the FSM corresponding to the currently selected expert (in its current state). If state s_2 is reached, control over speech generation is moved to the FSM corresponding to the newly selected expert, beginning in state s_0. The symbol '–' as output indicates no speech generation, and as an event indicates no state transition. Numbers in the output portion of the FSM indicate numbers of the speech acts (Table 12) generated by S#; '+' indicates concatenation of speech acts. NUL indicates an automatic transition; no event input is considered. Events not explicitly listed in the transition table do not produce any transitions in the FSM.

Algorithm 2 S# for normal-form games.

Input:
Description of the game
Initialize:
Compute the set of experts Φ_i
Initialize the aspiration level α_i
$\mathbf{a}^0 \leftarrow \text{random}(\text{A})$
$H_c \leftarrow \{\mathbf{a}_0\}$, $t \leftarrow 1$, and $\bar{r}_i \leftarrow 0$
Select an initial expert $\phi_i^t \in \Phi_i$
repeat
Produce speech acts as described in Section C.2.1
Observe player $-i$'s speech acts and compute $\Psi_i(t)$
if $\Psi_i(t) \neq \Psi_i(t-1)$ and $\Phi_i' \cap \Psi_i(t) \not\equiv \varnothing$ and $(n_t < P_i^{LISTEN}(t))$ **then**
$\bar{r}_i = \bar{r}_i/|H_c|$
Update α_i using Eq. (7)
Select ϕ_i^t using Eq. (13)
$H_c \leftarrow \{\mathbf{a}_{t-1}\}$ and $\bar{r}_i \leftarrow 0$
end if
Select and execute action a_i^t (as dictated by ϕ_i^t)
Observe $\mathbf{a}^t$ and r_i^t
$\bar{r}_i \leftarrow \bar{r}_i + r_i^t$
if $\mathbf{a}_t \in H_c$ **then**
$\bar{r}_i = \bar{r}_i/|H_c|$
Update α_i using Eq. (7)
Update each expert $\phi \in \Phi_i$ as necessary
Select ϕ_i^{t+1} using Eq. (13)
$H_c \leftarrow \varnothing$ and $\bar{r}_i \leftarrow 0$
else
$\phi_i^{t+1} = \phi_i^t$
end if
$H_c \leftarrow H_c \cup \{\mathbf{a}_t\}$
$t \leftarrow t + 1$
until Game Over

partner's speech acts to improve which experts it selects? Second, when should S# listen to its partner.

Expert Selection. S#'s expert-selection mechanism differs in two ways from that of S++. First, S# uses the last proposal made by its partner to refine the set of experts it selects from at the end of each cycle. Second, S# considers switching which expert it follows mid-cycle if its partner proposes a desirable plan. The resulting algorithm is summarized in Algorithm 2. We discuss each alteration in turn.

S# uses the last proposed plan made by its associate to modify its expert-selection mechanism. Rather than selecting experts using Eq. (8), this modified version selects experts using the following selection rule:

$$\phi_i^t \leftarrow \begin{cases} \phi_i^{(t-|H_c|)} & \text{with prob. } f(\alpha_i^t, \bar{r}_i^t) \\ \text{random}(\Phi_i''(t)) & \text{otherwise} \end{cases} \tag{13}$$

This selection rule is identical to Eq. (8) except that random selections are made from a set $\Phi_i''(t) \subseteq \Phi_i'(t)$ rather than $\Phi_i'(t)$.

In normal-form games[8], proposed plans involve the specification of a solution, which consists of either

[8]Multi-stage RSGs are slightly more difficult, though we describe a solution in Appendix E that worked well in practice.

a joint action or sequence of joint actions (see Table 12, speech IDs 15, 16, and 18[9]). S# compares this solution with the target solutions of each of its leader and follower experts. Let $\Psi_i(t)$ denotes the set of experts that have target solutions that are *congruent* (i.e., in the context of normal-form games, *equivalent*) with its partner's last proposed solution.

Given $\Psi_i(t)$, the set $\Phi_i''(t)$ is derived as follows:

$$\Phi_i''(t) \leftarrow \begin{cases} \Phi_i' \cap \Psi_i(t) & \text{if } (\Phi_i' \cap \Psi_i(t) \not\equiv \varnothing) \text{ and } (n_t < P_i^{LISTEN}(t)) \\ \Phi_i'(t) & \text{otherwise} \end{cases} \tag{14}$$

where n_t is a randomly selected number from the range $[0, 1]$. In words, if the set formed by the intersection of the set of potentially satisficing experts (Φ_i') and the set of experts congruent with player i's last proposal ($\Psi_i(t)$) is not empty, then with probability $P_i^{LISTEN}(t)$ (the probability that player i will listen to a satisfactory proposal made by player $-i$), the algorithm selects an expert from the intersection of those two sets. Otherwise, the algorithm selects experts from the set Φ_i' (as with S++).

The second difference between the expert-selection mechanisms of S# and S++ is that S# considers switching its expert in mid-cycle if its partner makes a new proposal. That is, if the new proposal made by its partner is congruent with some expert $\phi \in \Phi_i'$, then S# immediately switches its expert with probability $P_i^{LISTEN}(t)$. This alteration is reflected in the first *if*-block of Algorithm 2.

When to Listen. The probability $P_i^{LISTEN}(t)$ specifies the probability that player i will listen to a desirable proposal made by player $-i$. As in the case of speech generation, the act of listening to an associate opens one to the possibility of being exploited. For example, in the prisoner's dilemma, an associate could continually propose that both players cooperate, a proposal S# would continually accept and act on when $\alpha_i^t \leq 60$). However, if the partner did not follow through with its proposal, it could potentially exploit S# to some degree (for a period of time).

To avoid this, S# listens to its partner less frequently the more the partner fails to follow through with its own proposals. Let d_i^t be the number of times up to round t that player i has been *betrayed* while *listening* to player $-i$. Player i is considered to be *listening* to player $-i$ if the current expert ϕ_i^t, which was selected in round τ, was in the set $\Psi_i(\tau)$. Then, $P_i^{LISTEN}(t)$ is given by

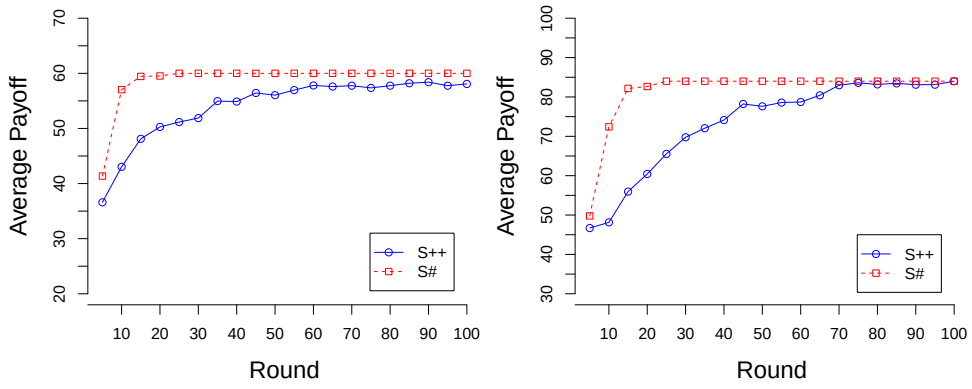
$$P_i^{LISTEN}(t) \leftarrow \begin{cases} 1 & \text{if } d_i^t = 0 \\ 1 - \frac{2^{d_i^t}}{20} & \text{if } 1 \leq d_i^t \leq 3 \\ \frac{2}{5 \cdot 2^{d_i^t - 4}} & \text{if } d_i^t \geq 4 \end{cases} \tag{15}$$

We note that the equations specifying when to talk (Eq. 12) and when to listen (Eq. 15) are heuristics derived from experimentation. While these heuristics worked well in practice, future work could explore these functions more carefully to further improve the way that S# utilizes cheap talk.

C.3 Results

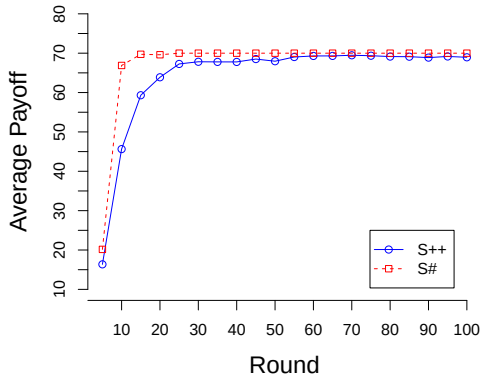
S# is primarily analyzed in Appendix F. An early version of S# (dubbed S#–), that only generates but does not listen to cheap talk, for multi-stage RSGs is also discussed and analyzed in Appendix E. Here, we provide a brief overview of a couple performance attributes of S#. In particular, we analyze S# with respect to security and its performance in self play.

[9] In particular, phrases 15 and 18 are used to specify a particular target solutions. Phrase 16 could also be used to specify a target solution, though it may also be used to specify the next solution in the sequence. In this work, we assume that phrase 16 specifies a target solution only if the solution specified in this speech act is not part of the last target solution proposed by the partner.

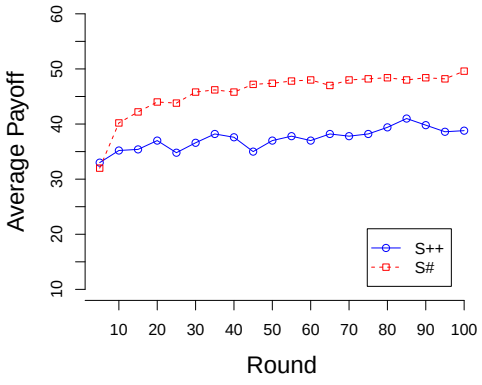


(a) Prisoner's Dilemma

(b) Chicken



(c) Alternator Game



(d) Shapley's Game

Figure 8: The average payoffs obtained by S++ and S# in self play in four different normal-form games (see Table 1 for payoff matrices). Results are an average of 50 trials each. Parameters for the algorithms were set the same as in user study 3 (see Table 11).

C.3.1 Security

Security refers to the guarantee that an algorithm will not, in expectation, receive an average payoff more then ε below its maximin value v_i^{mm}, regardless of the behavior of its associate. S++ is guaranteed to be secure [1]. Since S# uses S++, it also has the potential to be secure. However, because S# voices its strategy and listens to plans proposed by its partner, it exposes itself to exploitation. Fortunately, the variables P_i^{SPEAK} and P_i^{LISTEN} ensure that the probability of exploitation due to listening and voicing cheap talk becomes vanishingly small over time. Thus, like S++, S# is secure as $t \to \infty$.

C.3.2 Self Play

Figure 8 shows the improvement of S# over S++ in self play when communication is possible. Because two S# players take into account each other's proposed plans, they reach mutually cooperative solutions in self play much faster than does S++ in these four games. This results in higher payoffs in early rounds of the game to S#.

D User Study: Human vs. Machine 1

In the analysis performed in Appendix B, we observed that S++ [1], a recently developed machine-learning algorithm for repeated games, was the top-performing algorithm when compared with 19 different AI algorithms. The results of that analysis showed that S++ was able to learn to interact effectively with a wide range of AI algorithms across many different scenarios, such that it had the highest performance in both round-robin tournaments and evolutionary simulations. Additionally, S++ learns within time-scales that would support interactions with people.

In this section, we describe a user study designed to investigate whether S++ is able to learn to establish and maintain cooperative relationships with people in a variety of scenarios. In particular, we describe and discuss a 58-participant user study in which we paired S++ with people in four repeated normal-form games (Prisoner's Dilemma, Chicken, Chaos, and Shapley's Game). As part of this analysis, we benchmark the performance of S++ in these games with the performance of people.

D.1 Experimental Setup

We describe, in turn, the experimental protocol carried out in the user study, the games used in the study, and the user interface through which participants played the games.

D.1.1 Protocol

In the user study, we paired study participants with other Humans, S++, MBRL-1, and BULLY in four repeated normal-form games (Prisoner's Dilemma, Chicken, Chaos, and Shapley's Game). Each of the 58 participants played each of the four games in a random order. Over these four games, each participant was paired with each of the four partner types in a random order. The participants were volunteers from the Masdar Institute community, comprised primarily of graduate students and postdoctoral associates. The study was approved by Masdar Institute's research ethics committee.

Groups of between four and eight (typically six) people participated in the study at a time in a computer lab at Masdar Institute. The following procedure was followed for each group of participants:

1. Each participant was assigned a random ID, which defined the order that the participants played the four games as well as the partners they were paired with.

2. Each participant was trained on how to play the game using a web interface. Before proceeding to the first game, each participant was required to correctly answer several questions that tested their knowledge of how the game was played.

3. Repeat for each of the four games:

   (a) Each participant played a repeated game lasting 50 or more rounds. Participants were not told in advance the length of the game (i.e., number of rounds). To mitigate end-game effects, the length of each game was randomized between 50 and 57 rounds (thus, we only report on results over the first 50 rounds). Participants were also not told the identity of their partner, nor were they allowed to communicate with the other participants throughout the duration of the study.

   (b) Each participant completed a brief post-game questionnaire.

To incentivize participants to try to maximize their own payoffs in the game, subjects were paid based on their performance. Each participant received approximately $5 for participating in the study. A participant

(a) Prisoners' Dilemma (PD)

	c	**d**
c	0.60, 0.60	0.00, 1.00
d	1.00, 0.00	0.20, 0.20

(b) Chicken

	a	**b**
a	0.84, 0.84	0.33, 1.00
b	1.00, 0.33	0.00, 0.00

(c) Shapley's Game

	a	**b**	**c**
a	0, 0	1, 0	0, 1
b	0, 1	0, 0	1, 0
c	1, 0	0, 1	0, 0

(d) Chaos Game

	a	**b**	**c**
a	0.46, 0.67	0.24, 0.06	1.00, 0.00
b	0.00, 0.37	0.37, 0.07	0.01, 0.53
c	0.14, 0.69	0.20, 1.00	0.71, 0.90

Table 15: Payoff matrices of four games considered in our study. In each cell, the row player's payoff is listed first, followed by the column player's payoff.

could earn an additional \$10 based on their performance. Each participant's performance-based compensation was directly proportional to the amount of points he or she received in each repeated game. The amount of money a participant had earned in a game was displayed on the game interface in local currency.

In an effort to hide the identity of the players from each other, each round of each game lasted a fixed amount of time (in seconds). Participants were allotted a fixed amount of time to select an action, after which the results of the round were displayed. If a participant failed to select an action within the allotted time, a random action was automatically selected in their behalf. To encourage participants to always select an action, participants were not paid for points received in rounds in which they failed to select an action.

A separate instance of the algorithms S++ and MBRL-1 was used for each interaction. Thus, because the runs were independent from each other, the number of trials used for agent-agent pairings was not limited by the number of participants in the study. Thus, we conducted 100 different simulations of each agent-agent pairing in each game.

D.1.2 Games

The payoff matrices of the four repeated, normal-form, general-sum games used in the study are given in Table 15. Each of these games represents a different scenario in which cooperation and compromise are difficult to achieve against unknown associates. Furthermore, each game has an infinite number of Nash equilibria (NE) of the repeated game when the game is repeated after each round with sufficiently high probability.

The first game is the Prisoners' Dilemma (PD). In the PD, defection (d) is the dominant strategy for both players. However, this joint action is Pareto dominated by the solution (c, c) which leads to a payoff of 0.6 for both players. Both players can increase their payoffs by convincing the other player to cooperate. The dilemma or challenge arises from the difficulty of convincing a rational player that it is better to play a dominated strategy. The pair (c, d) means the column player gets the temptation reward of 1.0 for defecting against its partner, while the row player gets the lowest payoff of 0. Similarly, (d, c) means the column player gets the lowest payoff of 0 compared to the defecting partner's payoff of 1.0.

The second game is *Chicken*, also known as the *hawk-dove* game in evolutionary game theory [69]. Chicken models a situation in which players are in equilibrium if one player chooses to concede (the 'dove') by playing action a, while the other chooses to aggress (the 'hawk'; action b). However, since the payoff is higher for aggressing, each player is demotivated to be the one to concede.

Shapley's game [70], a variation on rock, paper, scissors, is of interest because most learning algorithms do not converge when they play it [5]. Moreover, it does not have a pure one-shot NE, but rather a unique one-shot mixed strategy NE in which all players play randomly. This NE yields a payoff of $1/3$ to each player. However, in repeated interactions, both players can do better by alternating between getting a payoff of 0 and 1, which results in a Pareto-optimal (and fair) outcome in which both players get an average payoff of 0.5.

The *Chaos* game was created by the authors such that the 'right' action is not immediately obvious. Like the PD, the game has three Pareto-optimal solutions and a single one-shot NE solution in which both players play a. However, the game is less structured than the PD and has a larger strategy space, which seemingly would make cooperation and compromise more difficult to learn against arbitrary associates.

Because the Chaos game is not symmetric, this game must receive special treatment in the analysis. When participants were paired with either S++ or MBRL-1, the participants were always given the role of the column player, who has a stronger position in this game. Thus, to allow us to make comparisons between players in this game, as well as to allow us to make comparisons across games, we used standardized payoffs (i.e., specifically the standardized z-scores across each game) when analyzing payoffs received.

D.1.3 Algorithms

In this study, we analyze the performance of the three[10] different player types: humans (the study participants), S++, and a model-based reinforcement learning algorithm (MBRL-1; see Table 2 in Appendix B for implementation details). MBRL-1 represents traditional learning algorithms for repeated games, and hence makes a good additional point of comparison that shows differences between traditional artificial intelligence and more recent advances (represented by S++). Parameter values for the version of S++ used in this user study are given in Table 11 in Appendix C.

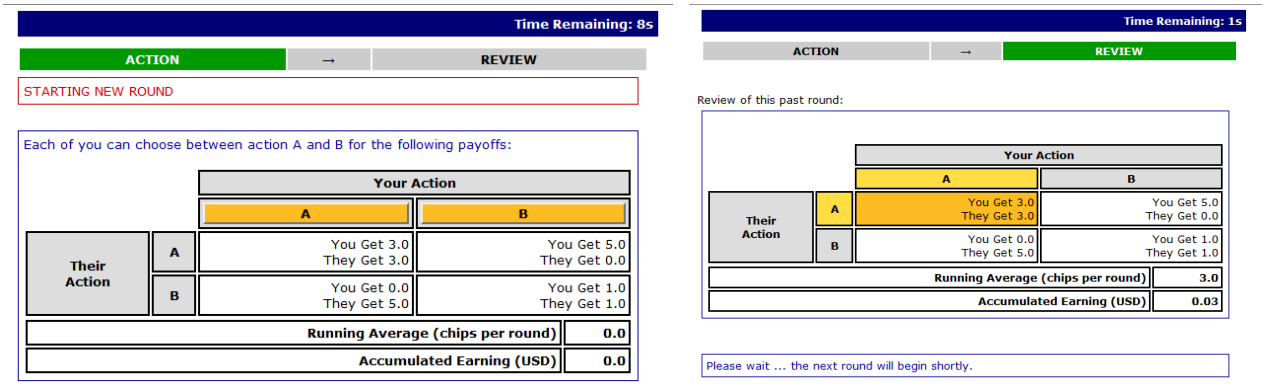
The three player types are paired to form six different pairings: MBRL-1/Human, Human/Human, S++/Human, S++/S++, S++/MBRL-1, and MBRL-1/MBRL-1. In this study, we are primarily concerned with comparisons between how well each of the player types interact with people (hence, MBRL-1/Human, Human/Human, and S++/Human pairings). However, the other three (agent-agent) pairings offer insights into the personalities of the algorithms and what they are capable of achieving.

D.1.4 User Interface

Each participant played the games on a desktop computer in a computer lab using the web-based graphical user interface (GUI) shown in Figure 9. In each round, the complete payoff matrix was displayed, along with a button corresponding to each of the participant's actions (Figure 9a). A timer was also displayed, indicating the time remaining in the current round (i.e., the deadline for selecting an action). Once the round was completed, the results of the round, including the actions selected by both players and the resulting joint payoff, were displayed on the screen for a fixed amount of time (Figure 9b), after which a new round began.

The average per-round payoff of the participant in the current game, along with the participant's current earnings, were displayed on the game interface. The average per-round payoff of the participant's partner was not displayed, however. This was done to emphasize that the participant's goal was to maximize its own payoffs rather than to outscore his/her partner.

[10]For simplicity of analysis, we omit games involving the algorithm BULLY. As in the simulation studies presented in Appendix B, BULLY sometimes performed very well, and other times performed very poorly. BULLY does not consistently cooperate across many games and associates, particularly against humans.



(a) (b)

Figure 9: Web-based GUI used by participants to play the games in the first user study. **(a)** The action-selection display used by the participant to select an action. **(b)** The round-review display through which the participant observed the results of a round.

D.2 Results

We focus on two aspects of the results in this study. First, we study the ability of AI algorithms to learn to cooperate with people. Hence, our primary metric is the proportion of rounds that both players cooperated with each other (i.e., mutual cooperation). However, given that self-interested individuals care only about cooperation if it leads to higher payoffs, we also analyze the payoffs received by the players.

D.2.1 Proportion of Mutual Cooperation

The proportion of mutual cooperation observed for each pairing in each game is shown in Figure 10. The figure shows several important results. First, S++/S++ pairings had the highest levels of mutual cooperation by the end of each game. The success of S++/S++ pairings is confirmed by a two-way analysis of variance in which the proportion of mutual cooperation was the dependent variable, and pairing and game were predictor variables. This analysis detected a main effect of pairing ($F(4, 919) = 228.3$, $p < 0.001$), as well as a main effect of game ($F(3, 919) = 18.11$, $p < 0.001$). Post-hoc Tukey tests show that, across games, the S++/S++ pairing was statistical superior to all other pairings ($p < 0.001$). The analysis also detected multiple interaction effects across games and pairings. These interaction effects can be visually assessed in the figure. For example, in both the Prisoner's Dilemma and in Chaos, S++/S++ pairings had the highest proportion of mutual cooperation throughout all rounds, whereas S++/S++ pairings only obtained the highest levels of mutual cooperation in Chicken and Shapley's Game after 30 to 40 rounds of interaction (S++/S++ pairings almost always eventually learn to play the mutual cooperative solutions in these games given sufficient rounds of interaction). Altogether, these results continue to demonstrate the potential of S++ to learn to cooperate.

The superiority of S++/S++ pairings (with respect to learning mutually cooperative solutions) can be traced to S++'s ability to use experience to improve its behavior as it gains more experience interacting with a particular partner. While the proportion of mutual cooperation in S++/S++ pairings increased substantially from the first ten rounds to the last 10 rounds (Figure 11), the same cannot be said of other pairings (at least not to the same extent). This result is confirmed by an analysis of variance in which the proportion of mutual

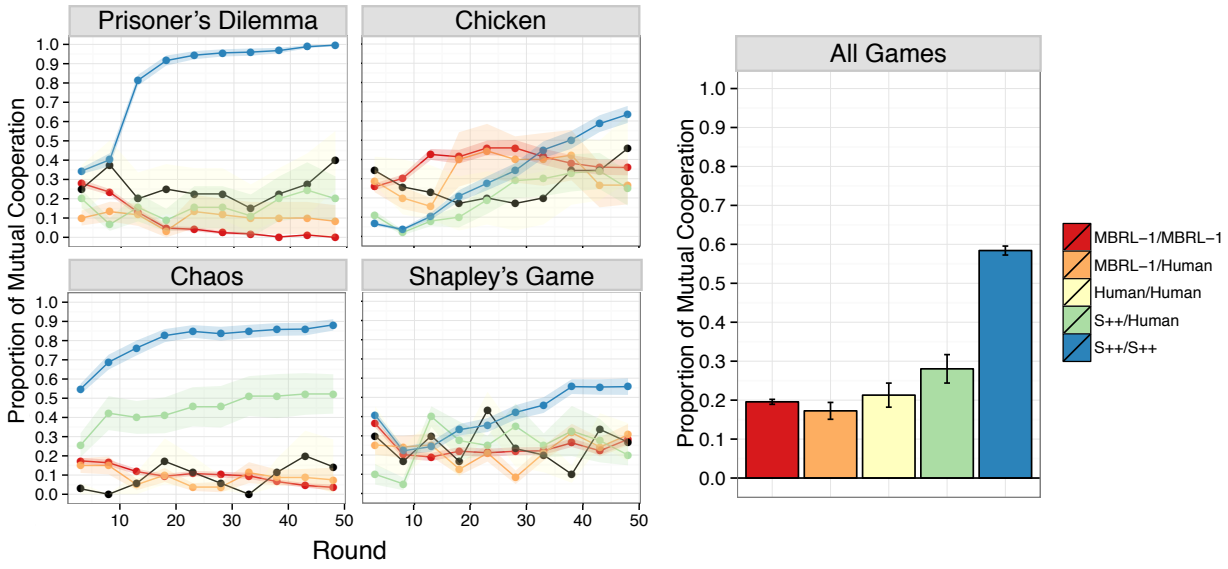


Figure 10: Proportion of mutual cooperation in individual games (left) and across all four games and rounds (right). Error bands/bars show the standard error on the mean. In the individual games, results are shown as the average of five-round increments.

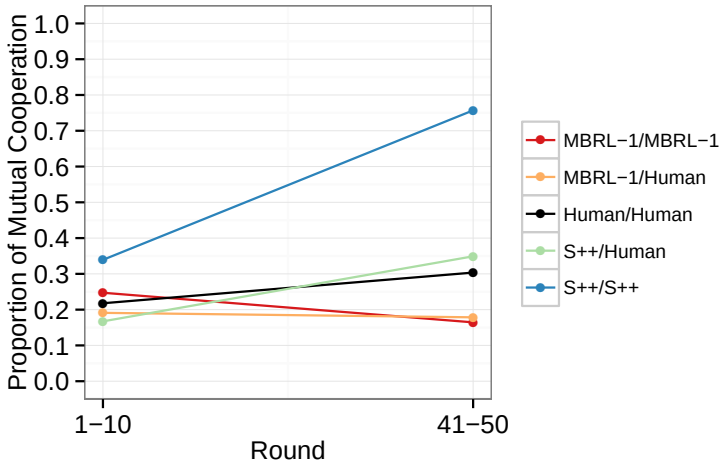


Figure 11: Average proportion of mutual cooperation across all games over the first and last ten rounds, respectively. Only pairings involving S++ had statistically significant increases in performance between these time intervals.

cooperation was the dependent variable, and round (the average payoff of the first and last ten rounds) and pairing were predictor variables. In addition to the main effect of pairing, the analysis detected a main effect of round ($F(4, 1870)$, $p < 0.001$). Pairwise t-tests show a statistically significant increase in S++/S++ and Human/S++ pairings from the first ten rounds to the last ten rounds ($p < 0.001$ and $p = 0.003$, respectively). Differences in Human/Human and MBRL-1/Human were not statistically significant ($p = 0.304$ and $p = 0.738$, respectively). However, the analysis shows that MBRL-1/MBRL-1 pairings even had a small, but statistically significant, decrease in proportion of mutual cooperation over time ($p < 0.001$) across these games. In summary, S++ is able to use its experience to improve its rate of cooperation when paired with humans and with copies of itself.

Second, humans and MBRL-1 failed to consistently cooperate with any of the three player types. Pairs of people played the mutually cooperative solution in only about 20% of rounds across all four games. Mutual cooperation among human pairings never reached 50% in any game over any five-round increment. Proportions of mutual cooperation obtained by MBRL-1 were similarly low.

Finally, despite the ability of S++ to learn to cooperate in self play, it failed to consistently elicit cooperative behavior from people. It had some success in learning the mutually cooperative solution when paired with people in Chaos (reaching, after about 30 rounds of interaction, mutual cooperation in about half of the rounds). However, across all games, there was no statistical difference ($p = 0.822$) between the proportion of mutual cooperation obtained in S++/Human and Human/Human pairings. Similarly, across all games, there was not a statistical difference between S++/Human and MBRL-1/Human pairings ($p = 0.190$).

Thus, S++ learns to cooperate with people as well as people do. However, this achievement was more a function of the inability of people to consistently cooperate with each other than S++ being able to consistently elicit cooperative behavior from people.

D.2.2 Payoffs Received

In this user study, the ability to establish cooperation relationships tended to translate directly into higher payoffs. Among the five pairings evaluated in the previous subsection (MBRL-1/MBRL-1, MBRL-1/Human, Human/Human, S++/Human, and S++/S++), the correlation between payoffs received (i.e., the standardized z-score in each game) and the proportion of mutual cooperation was $r(1876) = 0.800$, $p < 0.001$. This strong correlation indicates that, when associating with capable partners, the ability to achieve high payoffs is typically tied to one's ability to establish cooperative relationships.

Figure 12 shows the average standardized payoffs obtained by each player type when paired with each of the three partner types, across all four games. The figure shows several important trends. First, S++/S++ pairings, which had by far the highest levels of mutual cooperation, also achieved the highest average payoffs. Second, for each partner type, S++ achieved similar or higher payoffs as people. This further confirms that S++ is as capable as humans in establishing effective relationships in these games. Finally, there is a trend showing that MBRL-1 consistently achieved the lowest payoffs of the three player types.

D.3 Conclusions

This user study had several important results:

1. S++ is much more capable in self play than both people and MBRL-1. Self play is an important aspect of evolutionary robustness [4]. It also helps to demonstrate the potential of S++ to cooperate with other players.

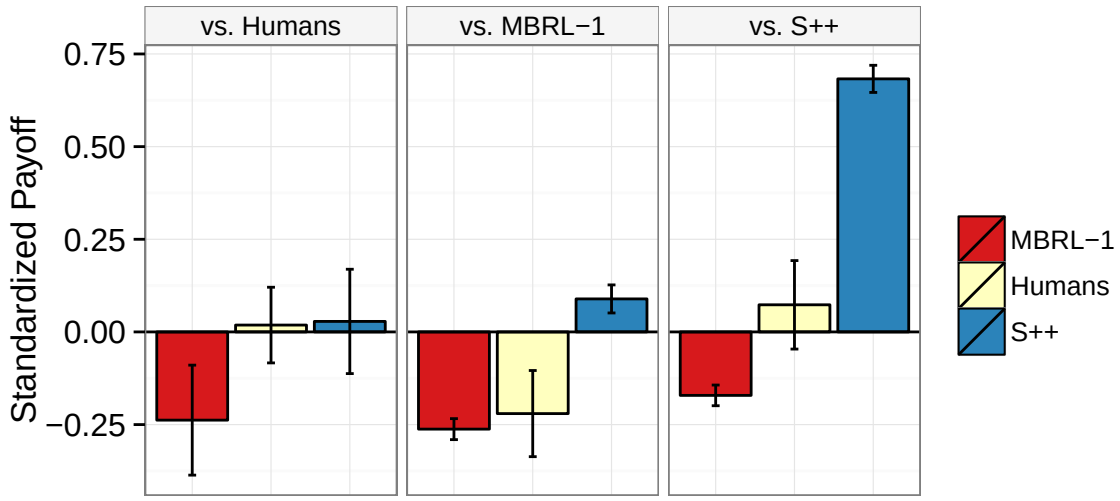


Figure 12: Average payoffs across all four games when paired with each player type (Human, MBRL-1, S++).

2. S++ appears to be much more effective at learning from experience than Humans or MBRL-1. In self play, MBRL-1 even showed statistically significant decreases in mutual cooperation over the four games considered. Meanwhile, S++ had statistically significant increases in mutual cooperation as the rounds of the game increased when paired with both a copy of itself and when paired with people. There was not a statistically significant increase in mutual cooperation among human pairings over time.

3. The performance of S++ was as good as or better than humans when paired with all three player types across the four games considered. Despite this success, the proportion of mutual cooperation between S++ and people was quite low, as was the case for Human/Human pairings. Thus, S++, in its current form, is unable to consistently elicit cooperative behavior from people.

In short, S++, in its current form, does not fully meet our objective of a learning algorithm that consistently learns to cooperate with people. Given the importance of this outcome in establishing effective human-machine societies, we extended S++ with the ability to communicate with people. The resulting algorithm, dubbed S# (pronounced 'S-sharp'), is analyzed in the two user studies presented in the next two appendices.

E User Study: Human vs. Machine 2

In the first user study, we observed that mutual cooperation was much more prevalent in S++/S++ pairings than in Human/Human pairings. Despite this success along with its effectiveness in associations with other AI algorithms (see Appendix B), S++ was unable to consistently elicit cooperative behavior from people across four repeated normal-form games. It did, however, cooperate with people as much as people did, as people also failed to consistently cooperate with each other in these games.

Given that people frequently do establish cooperative relationships among themselves in everyday life, we were somewhat surprised by this result. Thus, to further investigate human-human and human-machine cooperation in repeated interactions, we conducted a second user study[11]. This second user study differed from the first user study in two ways. First, in this second user study, we paired players in two multi-stage RSGs rather than in repeated normal-form games. This allowed us to evaluate the ability of S++ to establish cooperative relationships with people in more complex scenarios. Second, we added an additional condition in which players were allowed to talk to each other.

In the condition in which players were allowed to communicate, human players talked to each other face-to-face while playing the game. For human-machine pairings, we extended S++ with the ability to generate speech acts, which were voiced to the human partner through a Nao robot. This early version of S# (which we refer to as S#–) did not, however, take into account speech signals communicated by its partner (this additional extension is considered in the third user study, reported in Appendix F).

Ninety-six people participated in this study, which was approved by Masdar Institute's Human Subjects Research Ethics Committee. For ease of exposition, we discuss this user study in two parts. In Part 1, we discuss the condition of the study in which the participants were not allowed to communicate with each other. In Part 2, we discuss the impact of communication on the ability of the players to cooperate with each other.

E.1 Experimental Setup (Part 1)

We discuss, in turn, the protocol followed in Part 1 of the user study, the games used in the study, the algorithms involved, and the user interface through which participants played the games.

E.1.1 Protocol

Part 1 of this user study followed a 3x2 mixed design, in which the between-subjects variable was the partner type (Humans, CFR, or S++) and the within subjects variable was the game scenario (the SGPD and the Block game). Twelve groups of four participants each were recruited from the Masdar Institute community to participate in this part of the study. Half of the groups were assigned to a human-human condition, and were thus paired with each other as shown in the schedule shown in Table 16a. The other six groups were assigned to a human-machine condition in which two of the subjects were paired with CFR [72, 73] and two of the subjects were paired with S++, as specified in Table 16b.

The study proceeded as follows:

1. The group of four participants was assigned a condition (human-human or human-machine).
2. Each participant was randomly assigned an ID (H1, H2, H3, or H4).

[11] This study was originally presented at HRI 2015 [71].

(a) Human-human condition

Schedule for a Group of Four Subjects			
Order	Game	Pairing 1	Pairing 2
1	SPGD	H1-H2	H3-H4
2	Block Game	H1-H3	H2-H4

(b) Human-machine condition

Schedule for a Group of Four Subjects					
Order	Game	Pairing 1	Pairing 2	Pairing 3	Pairing 4
1	SPGD	H1-CFR	H2-CFR	H3-S++	H4-S++
2	Block Game	H1-CFR	CFR-H2	H3-S++	S++-H4

Table 16: The schedule of games and pairings followed in each condition of Part 1 of the second user study. Subjects participated in Part 1 of this user study in groups of four, in which each subject was randomly assigned an ID (H1, H2, H3, or H4). Each group was designated to one of two conditions: human-human or human-machine.

3. The SGPD was explained to the participants using slides and a video demo. Participants were allowed to ask as many questions as necessary until it was clear they understood the rules of the game.

4. The participants played a 54-round SGPD with their assigned partner (see Table 16). Participants played the game on a desktop computer using the arrow keys to select movements. The participants were not told the number of rounds in the game or the identity of their partner. They were not allowed to talk or signal to each other; computers were placed so that participants could not see each other.

5. Each participant completed the post-game survey shown in Figure 19 (the survey shown in Figure 20 was used in Part 2 of the user study).

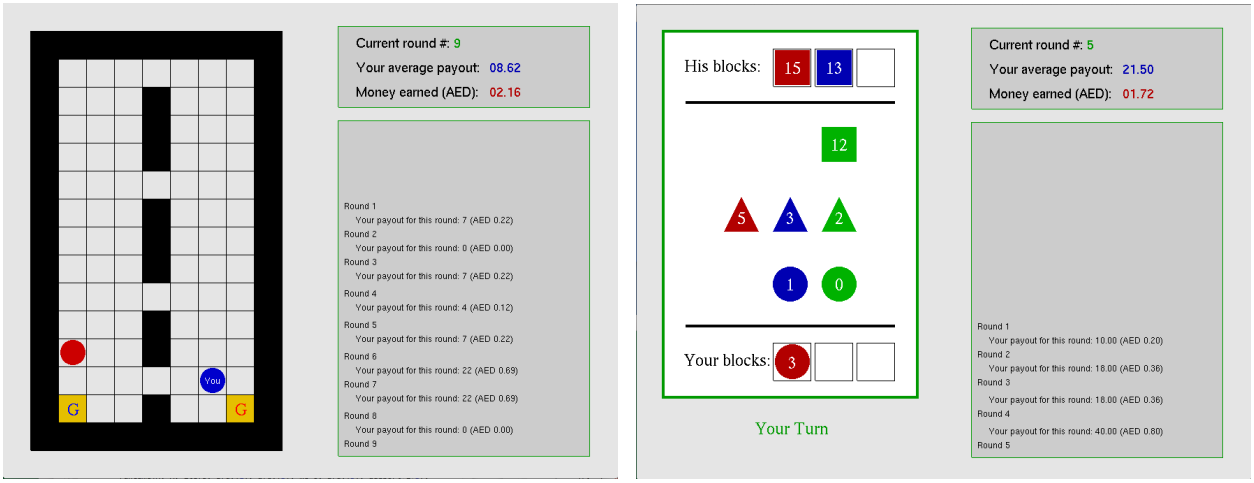
6. When all participants had completed the survey, they were introduced to the Block Game using slides and a video demo. Participants were allowed to ask as many questions as necessary until it was clear they understood the rules of the game.

7. The participants played a 51-round Block Game with the same player type as before. The player listed first in the pairing (Table 16) was assigned to be the player to select a block first – a role which was kept constant in each round of the game. The participants used the mouse to select the blocks on a GUI.

8. The participant completed the same post-experiment survey as in Step 5.

Participants were paid a $5.00 show-up fee. To incentivize the participants to try to maximize their own payoffs (which was the stated objective of each game), participants were also paid money proportional to the points they scored in the games (they could earn up to an additional $15.00). The GUI displaying the game interface also showed the amount of money the participant had earned so far in local currency.

In addition to the human-human and human-machine pairings specified in Table 16, machine-machine simulations were run to serve as additional points of comparison. Since a new version of CFR and S++ was initiated for each interaction, we were able to conduct additional runs for these machine-machine pairings; the number of available participants had no impact on these simulations. Thus, all results for S++/S++ and CFR/CFR pairings are obtained from 24 different runs.



(a) GUI used to play the SGPD

(b) GUI used to play the Block Game

Figure 13: Graphical user interfaces used by participants to play the SGPD and the Block Game, respectively. **(a)** In the SGPD, the participant used the arrow keys on the keyboard to select which direction to move their icon through the maze. **(b)** In the block game, participants selected blocks by clicking on the desired block with the mouse.

E.1.2 Games

As specified in Table 16, each participant first played the stochastic game prisoner's dilemma (SGPD), followed by the Block Game. Both of these games are explained in Appendix A. Recall that various levels of cooperation are possible in each game. So-called *mutual cooperation* in the SGPD is when both players choose to move through Gate B. In the Block Game, mutual cooperation is achieved when the players take turns (over rounds) getting all the squares and all the triangles, respectively.

E.1.3 User Interface

Participants played the games using GUIs. Screen shots of the GUIs used in both games are shown in Figure 13. In both GUIs, the game board was displayed on the left portion of the GUI, while the history of play was displayed on the right portion of the GUI. The participant's own payoffs in each round, his/her average payoff over all rounds, and his/her total earnings were all displayed prominently on the GUI interface. However, as in the other user studies, the payoffs of the participant's partner was not recorded on the GUI (though the participant could infer them). This was done to emphasize that the goal of the game was to maximize one's own payoffs rather than to outscore one's partner.

E.1.4 Algorithms

S++ was originally designed for repeated normal-form games. A recently published extension to the algorithm allows it to be used in multi-stage RSGs [2], by simply defining experts that define policies in each state of the RSG. As in normal-form games, these experts include *leader* and *follower* strategies, as well as a third genre of expert strategy called *preventative strategies* (see Crandall [2] for details).

As in the first user study, we compared the ability of S++ to learn to interact with people with that of another learning algorithm. In this second user study, we compared the performance of S++ with

CFR (counter-factual regret), which has become a core component of world-class computer poker algorithms [72, 73, 3]. CFR generalizes the idea of regret [11] to large extensive-form[12] games with imperfect information. CFR was originally designed to compute a Nash equilibria in large zero-sum games, though regret-minimization has often been used as a criteria for online learning in general-sum games (e.g., [12]). However, regret minimization is not guaranteed to correspond to payoff maximization in such games, and can even correlate positively with reduced payoffs when associates employ other learning algorithms [1].

Players were not told the identify of their partner. However, because algorithms make selections almost instantaneously in most cases, a participant might be able to infer that their partner was a computer algorithm due to how fast their partner selected actions. In the first user study, we addressed this problem by making each round last a fixed amount of time. If both players selected actions before this time elapsed, they still had to wait until time elapsed before observing the result. They were also forced to make decisions within the allotted time. This mechanism seemed to cause boredom or frustration among participants (the amount of time allotted tended to be either too long or too short), and is even less practical when rounds consist of many consecutive moves, as in the SGPD and the Block Game.

Thus, in this second user study, we eliminated fixed-length rounds and instead introduced artificial delays into the computer algorithm to try to make decision-times more human-like. To do this, we logged the amount of time that humans took to make decisions, and then used a k-nearest neighbor algorithm to infer how long the computer algorithm should delay before selecting an action. This technique was somewhat effective, though delays still sometimes seemed unnatural to participants (and, hence, did not seem human-like) due to the complexity of determining the relevant features of the game state that caused changes in human decision times[13].

E.2 Results (Part 1)

As in the first user study, we are interested in comparing the behavior and performance of computer algorithms with that of humans. We are particularly interested in finding algorithms that learn to consistently cooperate with people. Thus, in discussing the results of Part 1 of this second user study, we begin by discussing the proportion of mutual cooperation achieved by the algorithms. We then discuss the payoffs received by the algorithms in each pairing and game.

E.2.1 Proportion of Mutual Cooperation

The proportion of mutual cooperation achieved by each pairing in both games is summarized in Figure 14. These results have identical trends to those of the first user study. Only S++/S++ pairings achieved high levels of mutual cooperation (in both games). Additionally, neither CFR, S++, or Humans were able to consistently cooperate with people, though S++ tended to cooperate with people at least as much as people did. Human/S++ pairings were able to reach about 40% mutual cooperation in the SGPD, but were not nearly as effective in the Block Game. Because of its internal representation (which tends to lead to the computation of one-shot NE), CFR does not ever play the mutually cooperative solution in these games, regardless of the behavior of its partner.

[12] So that CFR can be used in the SGPD, the game is reduced to an extensive-form game by truncating a round after a certain number of moves (40 in our case). Since each round of the SGPD usually has little more than 30 moves (unless players make irrational moves), this simplification does not affect the strategies of rational players.

[13] Given that our k-nearest neighbor machine-learning algorithm failed to produce human-like delays in the decision process, we used tit-for-tat time delays in the third user study, wherein the computer algorithm delayed the same amount of time as the participant delayed in the previous move. This approach, while not perfect, seemed to be much more effective, as fewer participants commented on the delays in their partners' moves.

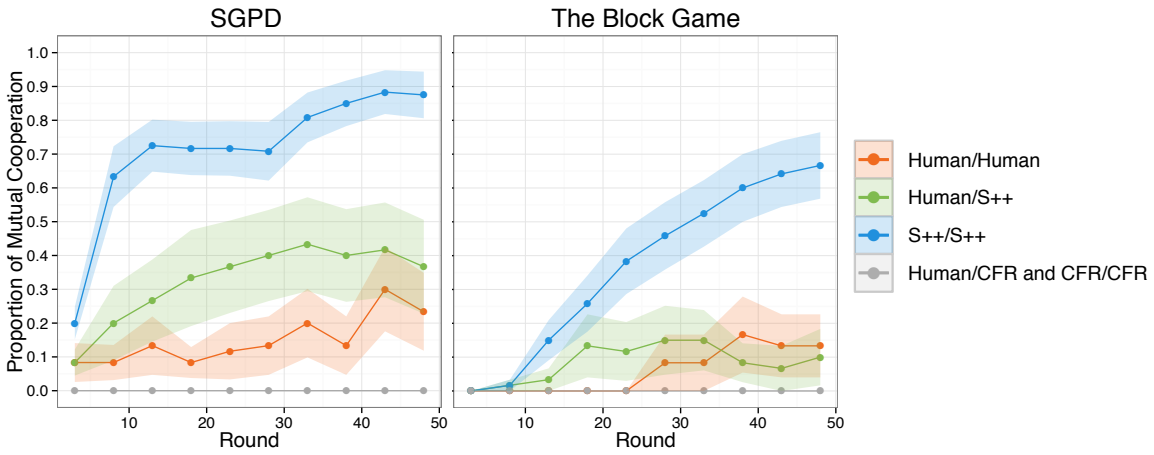


Figure 14: Proportion of mutual cooperation in each game. Each data point is the average of five-round increments. Error bands show the standard error on the mean. CFR/CFR and Human/CFR pairings always had the same value (no mutual cooperation) in all rounds in each game, and hence are both represented by the same line.

(a) SGPD (no communication)

Player	Primary Outcome				
	Both B	**Alt A-B**	**Both A**	**Bully**	**Other**
Humans	3	0.5	7.5	1	0
S++	5	0	5	0	2
CFR	0	0	11	1	0

(b) Block Game (no communication)

Player	Primary Outcome					
	Alt □-△	**Alt red-blue**	**All diff**	**Pure red-blue**	**Pure □-△**	**Other**
Humans	2	2	5	2	0	1
S++	1.5	0.5	4.5	2	0	3.5
CFR	0	0	5	5	0	2

Table 17: The number of subjects that reached each outcome by the end of 50 rounds when paired with a human partner in **(a)** the SGPD and **(b)** the Block Game, respectively.

An analysis of variance, in which the proportion of mutual cooperation over all rounds was the dependent variable, and pairing and game were predictor variables, confirms these trends. This analysis detected a main effect of pairing ($F(3, 1192) = 173.7$, $p < 0.001$), as well as a main effect of game ($F(3, 1192) = 107.1$, $p < 0.001$). The analysis also showed a statistically significant interaction between game and pairing ($F(3, 1192) = 16.28$, $p < 0.001$), which can be visually assessed in Figure 14. Post-hoc Tukey tests, collapsing across games, showed statistically significant differences between (1) S++/S++ pairings and each of the other pairings ($p < 0.001$), (2) S++/Human pairings and CFR/Human pairings ($p < 0.001$), (3) S++/Human and Human/Human pairings ($p = 0.012$), and (4) Human/Human and CFR/Human pairings ($p = 0.007$).

Table 17, which shows the number of participants that achieved each outcome (by the end of 50 rounds) when paired with each player type, provides further insight into the ability of the learning algorithms to collaborate with people. In the SGPD, both S++ and Humans learned mutual cooperation (*Both B*) with some participants (S++ cooperated with five, Humans with three). The rest of the participants typically learned mutual defection (*Both A*) when paired with S++ and Humans, though S++ tended to continue to attempt to pass through Gate B every few rounds in hopes of achieving a more-profitable outcome. In the Block Game, Humans and S++ rarely reached the most collaborative solutions.

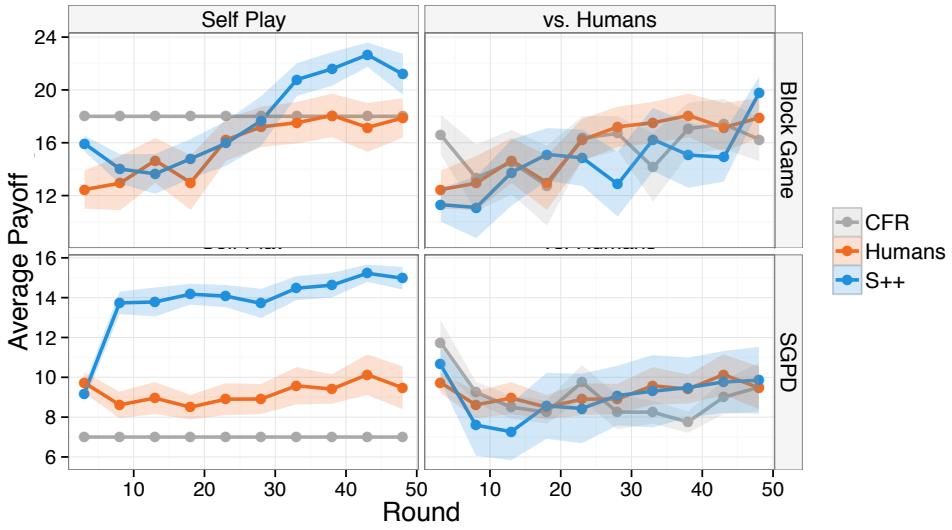


Figure 15: The average payoffs achieved by Humans, CFR, and S++ (a) in self play and (b) when paired with humans, respectively. The data is plotted as the average over five-round increments. Error bars show the standard error on the mean. Mutual cooperation in the Block Game yields an average payoff of 25, while mutual cooperation in the SGPD yields an average payoff of 16. Thus, the algorithms' failures to consistently cooperate with each other (with the exception of S++/S++ pairings) led to substantially reduced payoffs.

E.2.2 Payoffs Received

As in the other user studies, the correlation between payoffs received[14] and the proportion of mutual cooperation achieved was quite high ($r(238) = 0.695$, $p < 0.001$). A visual comparison of Figures 14 and 15, the latter of which shows the average per-round payoffs obtained by the algorithms in self play and when paired with humans, illustrates this correlation. As points of reference in considering Figure 15, note that mutual cooperation in the Block Game yields an average payoff of 25, while mutual cooperation in the SGPD yields an average payoff of 16.

S++'s ability to establish cooperative relationships in self play led to higher payoffs in both games. In the SGPD, mutual cooperation typically emerged within the first ten rounds, which in turn led to higher payoffs throughout the game. It took S++ longer to establish mutual cooperation in self play in the Block Game, which in turn meant that it did not achieve higher payoffs until later rounds in this game. On the other hand, because CFR is unable to model mutually cooperative solutions in either game, its payoffs remain low in both games in self play. Likewise, people also received low payoffs in self play.

When paired with humans, none of the players consistently achieved high payoffs in either game, nor was their a substantial difference in the payoffs achieved by the players.

E.3 Experimental Setup (Part 2)

In Part 2 of this study, we gave the players the opportunity to communicate with their partner. That is, human pairs were allowed to talk freely to each other, face-to-face, throughout the duration of the game. To

[14]To allow for analysis across games, we used the standardized z-score in each game in this correlation test.

(a) Human-S#– (no plans) condition

Schedule for a Group of Two Subjects			
Order	Game	Pairing 1	Pairing 2
1	SPGD	H1/S#– (no plans)	H2/S#– (no plans)
2	Block Game	S#– (no plans)/H1	H2/S#– (no plans)

(b) Human-S#– (plans) condition

Schedule for a Group of Two Subjects			
Order	Game	Pairing 1	Pairing 2
1	SPGD	H1/S#– (plans)	H2/S#– (plans)
2	Block Game	S#– (plans)/H1	H2/S#– (plans)

Table 18: In the human-machine condition of Part 2 of this second user study, subjects (denoted H1 and H2) participated in the study in groups of two. In the Block Game, the player that selects a block first in each round is the player listed first in the pairing.

add communication skills to S++, we developed an early (incomplete) version of S#, which we refer to as S#–. S#– is identical to S++, except that it simultaneously generates and voices (in this case through a Nao robot) speech acts to its partner. However, unlike the full version of S# considered in the next (third) user study, S#– does not take into consideration communication signals from its partner when selecting experts.

We first describe the experimental setup for this study, and then present the results.

E.3.1 Procedure

In Part 2 of this user study, participants were randomly assigned to one of three types of partners (Humans or two different forms of S#–). Forty-eight participants (none of whom took part in Part 1) participated in this part of the study. Half of the participants were assigned to the human-human condition, and were thus paired with each other as shown in Table 16a. The other 24 participants, who completed the study in groups of two, were assigned to a human-machine condition. Twelve of the participants were partnered with a version of S#– (in both games) that voiced only feedback/sentiments (i.e., no plans – Table 18a), while the other twelve subjects were partnered with a version of S#– that voiced both feedback and proposed plans (i.e., plans – see Table 18b).

Thus, viewing Parts 1 and 2 of the study together (and excluding subjects assigned to play against CFR in Part 1), the participants in the study, who each played the SGPD game and the Block game, were randomly assigned to one cell of a 2 x 3 between subject design: Communication (between-participants: Yes or No) × Partner (between-participant: Human, or one of two versions of S#–, with or without the capacity to talk about plans). Because the first level (human partner) meant that participants were paired with each other, this level required twice as many human participants as the other levels, in order to comprise as many pairs (hence, random, allocations took into account this need). Note also that practically speaking, the two versions of S#– are undistinguishable in the absence of communication, which means that the protocol was exactly the same in two cells of the design.

The procedure followed for each subject was the same in Part 2 as in Part 1 with two exceptions. First, players were aware of who their partner was, as their partner was sitting right in front to them. Second, the players were allowed to talk to each other freely, without restriction, throughout the game.

Figure 16: A Nao robot delivered the speech acts, generated by S#-, to the participants.

E.3.2 User Interface

The GUI through which participants played the game was the same as the one used in Part 1 of this user study (Figure 13). When participants were paired with S#–, speech acts were delivered by a Nao robot, who was placed before the participants (Figure 16).

E.3.3 Communication

Both the face-to-face talk used between human players and the speech acts voiced by S#–are forms of *cheap talk*. Cheap talk refers to non-binding, unmediated, and costless communication [74, 75]. Cheap talk has been cited as a means for equilibrium refinement [76], and has been shown to improve collaborations among people in some scenarios [77, 78].

In this user study, we consider whether a learning robot can employ cheap talk to help establish cooperative relationships with people. We are unaware of past work interweaving cheap talk with online learning. This is potentially due to the difficulty of knowing what to communicate, as most learning algorithms have representations that are difficult to understand and articulate. However, we note parallels to the work of Thomaz and Breazeal [79], in which a simulated robot (tasked to learn to solve an environment that can be modeled as a Markov decision process) signaled uncertainty to a human teacher by pausing in stages in which multiple actions had similar Q-values. This helped the human to understand what the robot still needed to learn. However, in multi-stage RSGs in which their is conflicting interest between players, discussions about local Q-estimates are likely to be at too low of a level to fully resonate with human partners. Additionally, we anticipate that the competitive natures of these games will make such low-level communications insufficient.

We consider two kinds of cheap talk to encode in S#–: *feedback cheap talk* and *planning cheap talk*. We refer to cheap talk that addresses assessments of past events as feedback cheap talk. As an example, a person or robot might comment on their satisfaction with past events, or comment on how past events made them feel. Additionally, feedback cheap talk includes assessing the past behaviors of one's partner, perhaps expressing how the other player's actions make them feel, or expressing what they wished the other player had done instead. We anticipate that feedback cheap talk could potentially be produced from most learning algorithms with careful thought.

Planning cheap talk is forward looking. It involves suggesting future behavior to one's partner and/or revealing one's current or future strategy. Of course, such cheap talk is non-binding – neither of the players must actually do what is spoken. Regardless, we anticipate that most learning algorithms have representations that are too cryptic to easily be made to produce effective planning cheap talk for arbitrary scenarios,

as humans typically communicate plans at higher levels than typical machine-learning algorithms reason.

E.3.4 Algorithms

In the first user study and in Part 1 of the second user study, S++ demonstrated some ability to establish cooperative relationships. In both repeated normal-form and multi-stage RSGs, S++/S++ pairings typically converge to mutual cooperation. On the other hand, CFR does not compute mutually cooperative solutions in many games – its representation does not allow it to effectively model such solutions. Thus, in Part 2 of this study, we focus exclusively on S++ in hopes of finding an algorithm that can consistently cooperate with people (and, in so doing, achieve higher payoffs).

Unlike many learning algorithms, S++ is structured so that its high-level strategies are understandable (and expressible) to people. Thus, it can be used to generate both feedback and planning cheap talk that is game-generic, such that the same cheap talk can be used in many different RSGs.

S++ generates speech acts (i.e., cheap talk) for multi-stage RSGs in essentially the same way as in normal-form games (see Appendix C). For completeness, we describe the exact speech system used in the versions of S#– used in this study. This speech-generation system has three components: (1) a set of speech acts, (2) the set of events used by S++ to establish its state, and (3) the finite state machines (FSMs) with output that are used to generate speech acts given the algorithm's state and the events that occur.

Set of speech acts. The set of speech acts used by S#– in this second user study are given in Table 19. By and large, this set of speech acts allows S#– to express similar ideas as the smaller set of speech acts used in the third user study (Table 12). The set of speech acts used in this user study (Table 19) is larger, but many of the phrases have similar meanings.

Despite the similarities between this set of speech acts and those used in the third user study, there is a substantial difference between these two sets. While solutions can be referred to by *joint actions* in normal-form games, the same is not realistic in multi-stage RSGs. Instead, solutions are specified indirectly through words such as *cooperation* (e.g., speech ID 2) and statements that compare and contrast payoffs (e.g., speech IDs 3, 4, 9-12). The exact meaning is left to the other player to interpret. In this way, the speech system used by S#– remains game-generic.

The speech acts given in Table 19 contain statements that allow S#– to both provide feedback and propose plans. Speech acts considered to be part of plan-making are marked by bold type in the table. In the no-plans version of S#– used in this study, these plan-related (bold) phrases are not spoken.

Events. The set of events used by S#– in multi-stage RSGs (Table 20) is very similar to those used in normal-form games (Table 13; third user study), except that we do not need events related to reactions to the other player's signals (events x, y, and z), since, unlike S#, S#– does not respond to the signals spoken by its partner.

Finite state machines with output. The FSMs with output used by S#– in this user study are given in Table 21. The FSMs have similar structure as those used in user study 3 (described in Appendix C), with some stylistic variations.

Like S#, S#– (plans) voices planning cheap talk for two different kinds of events. First, because each expert encodes a (perhaps radically) different high-level strategy, the random selection of experts (i.e., exploration) can appear (to a human partner) to be disjoint, irrational reasoning. Thus, when S#– changes which expert it follows, it produces a speech act that notifies the human partner of this change. Example speech acts include "I've changed my mind," and "I've had a change of heart." Additionally, when this switch in strategies leaves some promised act undone (such as a promised punishment), the robot tries to soften the discontinuity by saying "I'll let you off this time."

ID	Text	ID	Text
1	**Here's the deal:**	28	We can both do better than this.
2	**Let's cooperate with each other.**	29	I've changed my mind.
3	**We can do this by taking turns receiving the higher payoff.**	30	I've had a change of heart.
		31	That round's result was not good enough for me.
4	**I deserve a higher payoff.**	32	You are an idiot!
5	**If you do not cooperate, I'll punish you thereafter.**	33	This is not good for our relationship
6	**If you do not comply, I'll punish you.**	34	For the sake of our relationship, cease this untoward behavior.
7	**I forgive you.**	35	Friends do not do that to each other.
8	**Cooperation will bring us both a higher payoff.**	36	**I'm going to teach you a lesson you will not forget.**
9	**It's my turn to get the higher payoff. You'll get the higher payoff next time.**	37	**You will pay for this!**
		38	**I'm going to make sure you do not profit from this malicious act.**
10	**It's your turn to get the higher payoff.**		
11	**My turn.**	39	Good for me.
12	**Your turn.**	40	That's what I wanted.
13	Sweet. We are getting rich. Let's continue this.	41	That's what I'm talking about.
14	I trusted you to <game-specific action label>	42	Excellent!
		43	Great!
15	I thought you should <game-specific action label>	44	Satisfactory.
		45	Good.
16	I will let you off this time.	46	I like this.
17	You jerk!	47	Nice!
18	You buffoon!	48	That was fair.
19	You fool!	49	Serves you right, jerk.
20	Curse you!	50	In your face!
21	You betrayed me!	51	Take that!
22	That was selfish of you	52	**I'll play fair if you'll play fair.**
23	That was not fair!	53	**I insist on equal payoffs.**
24	Are you only thinking of yourself.	54	That's what you get.
25	**I will not let you cheat me.**	55	You forced my hand.
26	The amount of points we get is up to you.	56	I must be cursed.
27	Do not threaten me.	57	Darn.

Table 19: Speech acts used by S#– in the second user study. Planning speech acts are given in bold, feedback speech acts are in plain text. Thus, both plain text and bold speech acts were voiced by S#– (plans), while S#– (no plans) only voiced the speech acts given in plain text.

Symbol	Event descriptions
n	S#– selected a new (distinct from the previous cycle) expert to follow. The partner currently has no guilt
ng	S#– selected a new (distinct from the previous cycle) expert to follow. The partner is currently guilty
c	An expert has been executed for a complete cycle.
b	S#– believes the players can both receive higher payoffs than what they received in the previous cycle. This event is not generated if the new expert is MBRL.
s	The current expert is *satisfied* with last round's payoff.
f	The current expert *forgives* the other player.
d	The other player *defected* against S#–.
g	The other player profited from its defection (and is *guilty*).
p	The expert has *punished* its guilty partner.
u	S#– did not succeed in punishing its guilty partner.

Table 20: A list of event symbols and event descriptions. The first six events correspond to master-level (i.e., expert selection) events of S++, while the last six events impact individual experts.

(a) Master-level FSM, executed after each visit to a state in an expert-level FSMs.

State	State Transitions: Events					Speech Acts (Output): Events				
	¬(n or ng)	n	ng	n, b	ng, b	¬(n or ng)	n	ng	n, b	ng, b
s_0	s_2	s_1	s_1	s_1	s_1	ε	r({29,30})	16+r({29,30})	28+r({29,30})	28+16+r({29,30})
s_1	Go to state s_0 of the FSM corresponding to the newly selected expert.					ε	ε	ε	ε	ε
s_2	Return to the current state of the current expert.					ε	ε	ε	ε	ε

(b) FSM for the MBRL expert.

State	State Transitions: Events			Speech Acts (Output): Events		
	NUL	¬s	s	NUL	¬s	s
s_0	s_0	s_0	s_0	–	r({31,56,57})	r({39,44,47})

(c) FSM for the maxmin expert.

State	State Transitions: Events			Speech Acts (Output): Events		
	NUL	¬s	s	NUL	¬s	s
s_0	s_0	s_0	s_0	–	–	–

(d) FSM for the preventative strategy Bouncer. Speech acts generated from state s_1 are only generated with probability given by $\frac{1}{1.0+\kappa/15.0}$, where κ is the number of times the event $\neg s$ and s have occurred, respectively, when Bouncer is active.

State	State Transitions: Events			Speech Acts (Output): Events		
	NUL	¬s	s	NUL	¬s	s
s_0	s_1	–	–	25+53+52	–	–
s_1	–	s_1	s_1	–	r({26,27,54,55})+r({25,53,52})	r({41,45,46,48})

(e) FSM for leader experts. For "fair" leader experts, <Intro> = 1+2+*3*+5. For "bully" leader experts, <Intro> = 4+6.

State	State Transitions: Events						Speech Acts (Output): Events					
	NUL	g	f	s	d	p	NUL	g	f	s	d	p
s0	s1	s0	s0	s0	s0	s0	<Intro>	–	–	–	–	–
s1	–	s9	s1	s2	s2	s1	–	r({17-20,32})+14+r({36-38})	–	r({40-43})+*9,10*	–	–
s2	–	s10	s2	s3	s3	s2	–	r({17-20,32})+14+r({36-38})	–	r({40-43})+*9,10*	–	–
s3	–	s11	s3	s4	s4	s3	–	r({17-20,32})+14+r({36-38})	–	r({40-43})+*9,10*	–	–
s4	–	s11	s4	s5	s5	s4	–	r({17-20,32})+14+r({36-38})	–	r({40-43})+*11,12*	–	–
s5	–	s11	s5	s6	s6	s5	–	r({17-20,32})+14+r({36-38})	–	r({40-43})+*11,12*	–	–
s6	–	s11	s6	s7	s7	s6	–	r({17-20,32})+14+r({36-38})	–	*11,12*	–	–
s7	–	s11	s7	s8	s8	s7	–	r({17-20,32})+14+r({36-38})	–	13	–	–
s8	–	s11	s8	s8	s8	s8	–	r({17-20,32})+14+r({36-38})	–	–	–	–
s9	–	s9	s3	s10	s10	s9	–	–	7+8+<Intro>	–	–	r({49-51})
s10	–	s10	s4	s11	s11	s10	–	–	7+8+<Intro>	–	–	r({49-51})
s11	–	s11	s5	s11	s11	s11	–	–	7+8+<Intro>	–	–	r({49-51})

(f) FSM for follower experts. For "fair" follower experts, <Intro> = 2+*3*. For "bully" follower experts, <Intro> = 4.

State	State Transitions: Events						Speech Acts (Output): Events					
	NUL	g	f	s	d	p	NUL	g	f	s	d	p
s0	s1	s0	s0	s0	s0	s0	<Intro>	–	–	–	–	–
s1	–	s2	s2	s2	s2	s2	–	r({21-24})+15	–	r({40-43})	–	–
s2	–	s3	s3	s3	s3	s3	–	r({21-24})+15	–	r({40-43})	–	–
s3	–	s4	s4	s4	s4	s4	–	r({21-24})+15	–	r({40-43})	–	–
s4	–	s4	s4	s5	s5	s4	–	r({21-24})+15	–	r({40-43})	–	–
s5	–	s4	s5	s6	s6	s5	–	r({21-24})+15	–	r({40-43})	–	–
s6	–	s4	s6	s7	s7	s6	–	r({21-24})+15	–	–	–	–
s7	–	s4	s7	s8	s8	s7	–	r({21-24})+15	–	13	–	–
s8	–	s4	s8	s8	s8	s8	–	r({21-24})+15	–	–	–	–

Table 21: Feedback and planning cheap talk is generated by a set of finite state machines (FSMs) with output. r(X) denotes a randomly selected speech act from the set X of speech acts. '+' indicates concatenated strings. <Intro> specifies a sequence of speech phrases determined by the target solution advocated by the expert strategy. "Fair" strategies correspond to target solutions in which the partner receives a similar or higher payoff as the algorithm; "bully" strategies target solutions that give the algorithm a higher payoff than its partner. *n* indicates that speech act n is spoken only if the target solution of the strategy is an alternating target solution; *x,y* is the appropriate selection between speech acts x and y given the current step in the target solution.

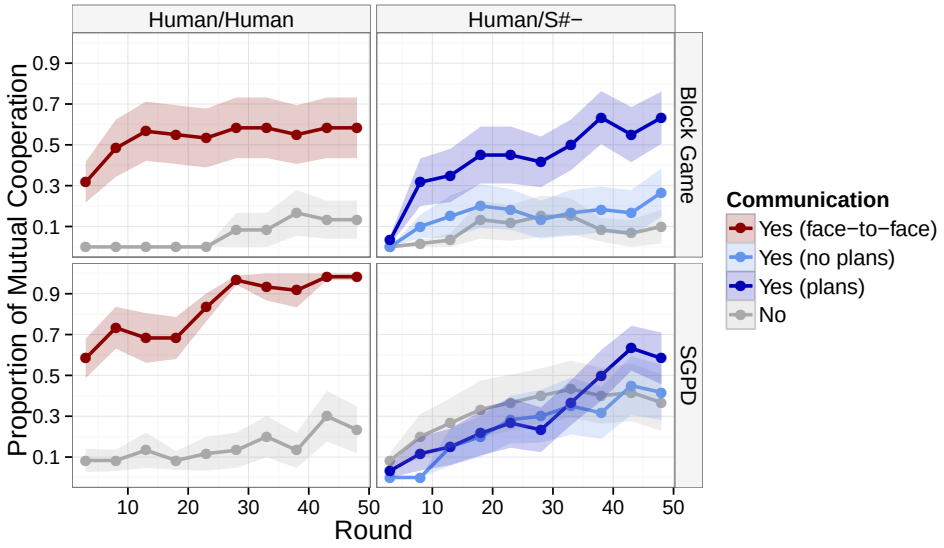


Figure 17: Proportion of mutual cooperation achieved in the second user study for Human/Human and Human/S#– pairings. Note that when communication is not possible, S#– is equivalent to S++. Participants in Human/Human pairings were allowed to talk freely to each other face-to-face, while communication between people with S#– was limited to one-way communication spoken by S#–, since S#– does not internalize communication signals from its partner.

As in normal-form games, each expert can also produce planning cheap talk. These plans can be communicated at a high level, as most experts used by S++ encode a high-level strategic ideal that is easily understood by people. For example, one of S++'s experts (called Bouncer) seeks to minimize the difference between the robot's and the human's payoffs. When Bouncer is selected, S#– announces "I will play fair if you will play fair," and that it insists on equal payoffs and will not be cheated. Others of S++'s experts encode trigger strategies, which carry out epochs of cooperation and punishment depending on the behavior of the associate. These trigger strategies can easily be articulated when they are selected. Furthermore, these trigger strategies can be modeled with simple FSMs, which S#– uses to produce speech acts that inform its partner about switches between stages of cooperation and punishment.

E.4 Results (Part 2)

Recall that our goal is to produce a self-regarding learning algorithm that establishes cooperative relationships with people as well as people do. Thus, we compare the performance of S#– to that of people (when partnered with people) with respect to the proportion of mutual cooperation achieved as well as payoffs received when associating with people. We also compare subjective assessments made by participants in post-game questionnaires about the perceived intelligence of people and S#–.

E.4.1 Proportion of Mutual Cooperation

The average proportion of mutual cooperation achieved by each pairing (under both communication conditions) in each game is shown in Figure 17. The figure shows two important results. First, Human/Human pairings benefitted substantially from the ability to communicate. Whereas mutual cooperation was nearly

(a) SGPD

Player	Primary Outcome				
	Both B	**Alt A-B**	**Both A**	**Bully**	**Other**
Humans	3	0.5	7.5	1	0
Humans (f2f)	12	0	0	0	0
S++	5	0	5	0	2
S#– (no plans)	5.5	0	5.5	1.0	0
S#– (plans)	9.5	0	2	0.5	0

(b) The Block Game

Player	Primary Outcome					
	Alt □-△	**Alt red-blue**	**All diff**	**Pure red-blue**	**Pure □-△**	**Oth-er**
Humans	2	2	5	2	0	1
Humans (f2f)	7	2	1	0	2	0
S++	1.5	0.5	4.5	2	0	3.5
S#– (no plans)	3.5	0	3	2	1.5	2
S#– (plans)	9	0	0	1	2	0

Table 22: The number of subjects that reached each outcome by 50 rounds when paired with a human partner. Note that the data for Humans and S++ was obtained from Part 1 of the study. S++ is equivalent to S#– when communication is not possible.

non-existent between people when communication was not allowed, mutual cooperation typically emerged between people in the communication condition.

Second, communication did not produce more substantial levels of mutual cooperation between Humans and S#– when signals expressed by S#– did not include plan proposals. However, when S#– did propose plans, higher levels of cooperation began to emerge at the end of the first game (the SGPD), and then become substantial in the Block Game (the second game played). This result suggests that, if S++ is combined with the ability to propose (through speech acts) the joint plans it intends to carry out, it can establish much more cooperative relationships. However, these cooperative relationships between people and machines appear to emerge much slower than in Human/Human pairings (when communication is possible), an issue we seek to address in the third (and final) user study presented in this paper.

A statistical analysis confirms these trends. In the Block Game, the analysis only detected a main effect of Communication, $F(1, 54) = 14.2$, $p < 0.001$. The effect of Pairing was not significant ($p > 0.31$), and the interaction effect was marginally significant, $F(2, 54) = 3.0$, $p = 0.06$. Further investigation of this marginal interaction revealed that all pairings cooperated to the same level without communication ($p = 0.31$), and that there was a marginal effect of Pairing with Communication, $F(2, 33) = 3.1$, $p = 0.06$. As revealed by a follow-up Tukey test, this effect reflected the fact that human pairings did marginally better than pairings involving a human and the version of S#– that could not communicate about plans (p = 0.06).

In the SGPD game, the analysis detected a main effect of Communication, $F(1, 54) = 12.9$, $p < 0.001$, and a main effect of Pairing, $F(2, 54) = 4.2$, $p = 0.02$. This effect must be interpreted in the context of the interaction effect, $F(2, 54) = 12.2$, $p < 0.001$. Without communication, all pairings cooperated to the same level ($p = 0.28$). With communication, humans cooperated more with each other than with either version of S#–. An ANOVA with the Proportion of Mutual Cooperation (averaged over all rounds) as the dependent and Pairing as the predictor, restricted to the Communication condition, detected an effect of Pairing, $F(2, 33) = 20.6$, $p < 0.001$, and follow-up Tukey tests confirmed the superiority of human-human pairings ($p < 0.001$ in both comparisons).

Convergence characteristics provide further insight into how cheap talk impacted S#–'s ability to establish cooperative relationships with people. While feedback cheap talk alone produced little increase in the number of highly collaborative outcomes, feedback and planning cheap talk together led to substantial increases in highly cooperative outcomes by the end of 50 rounds (Table 22).

In summary, allowing for communication greatly improved cooperation. The version of S#– that could not communicate about plans failed to take full advantage of communication, though, as its level of mutual cooperation with humans remained lower than human-human levels of cooperation. The version of S#– that

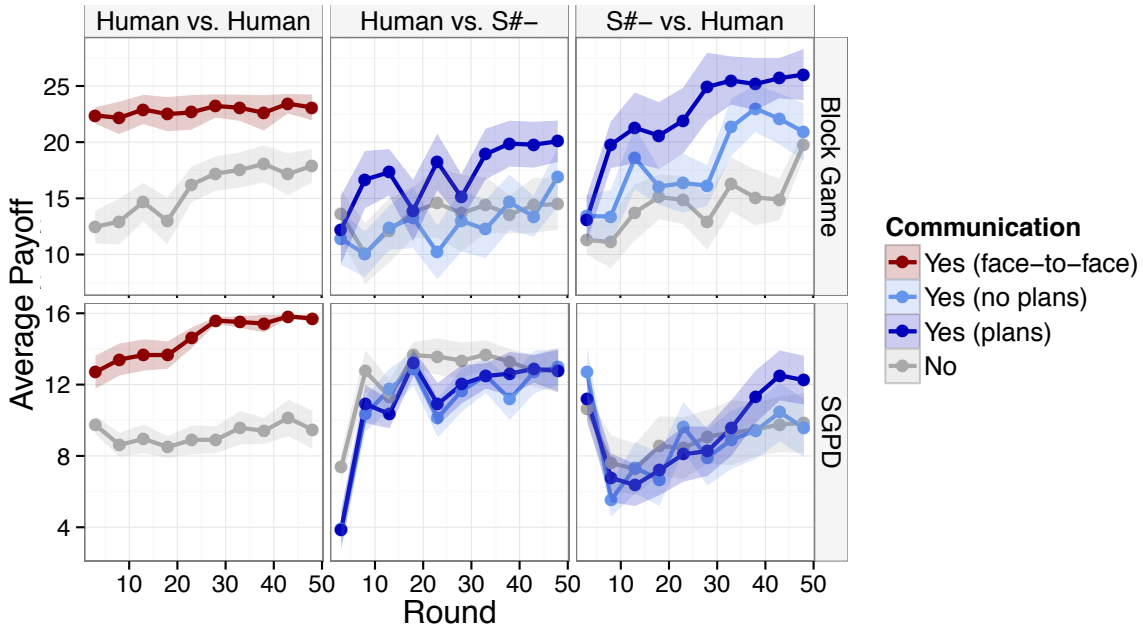


Figure 18: The average payoffs achieved by the players in both Human/Human and Human/S#– pairings.

could communicate about plans fared better. At least in the Block Game, it achieved the same level of mutual cooperation as seen in pairings of humans enjoying full communication. Even this version of S#– was limited, though, by its inability to process the messages sent by its human partners. The third and final user study investigated the levels of mutual cooperation achieved by S#, the version of S++ endowed with the ability to process as well as send messages to its partner.

Recall that, in this user study, the ability to communicate also coincided with knowledge of the identify of own's partner. While this could (and likely does) have a substantial impact on the behavior of people, the increase in cooperative behavior cannot be attributed solely to this knowledge, given that S#– (no plans) did not increase cooperation substantially, but S#– (plans) did. In the subsequent user study, we isolate these two factors to better understand the impact of communication on cooperation in relationships involving people.

E.4.2 Payoffs Received

Figure 18 illustrates that, as in other user studies, payoffs received tend to correlate highly with the proportion of mutual cooperation achieved by these players.

E.4.3 Humanness

The participants in our study also held much higher opinions of S#– (plans) than of S++. In the post-game questionnaires, participants were asked to indicate how likable and how intelligent their associate was on the scale 1 to 5. Across both games, cheap talk had a statistically significant impact on both of these ratings ($F(2, 66) = 6.99; p = 0.002$ and $F(2, 66) = 8.19; p < 0.001$, respectably). Pairwise comparisons show that participants thought the robot was more intelligent when it employed feedback and planning cheap talk than when it employed just feedback cheap talk ($p = 0.005$) or no cheap talk at all ($p = 0.001$). However, when

the robot only produced feedback cheap talk, it was not seen as more intelligent than when the algorithm produced no cheap talk (p = 0.912).

E.5 Conclusions

The results of Part 1 of this second user study essentially mirrored the results of the first user study. S++ is able to establish cooperative relationships with other players that employ S++, but it is unable to consistently establish cooperative relationships with people. People also failed to consistently establish cooperative relationships with people in these games.

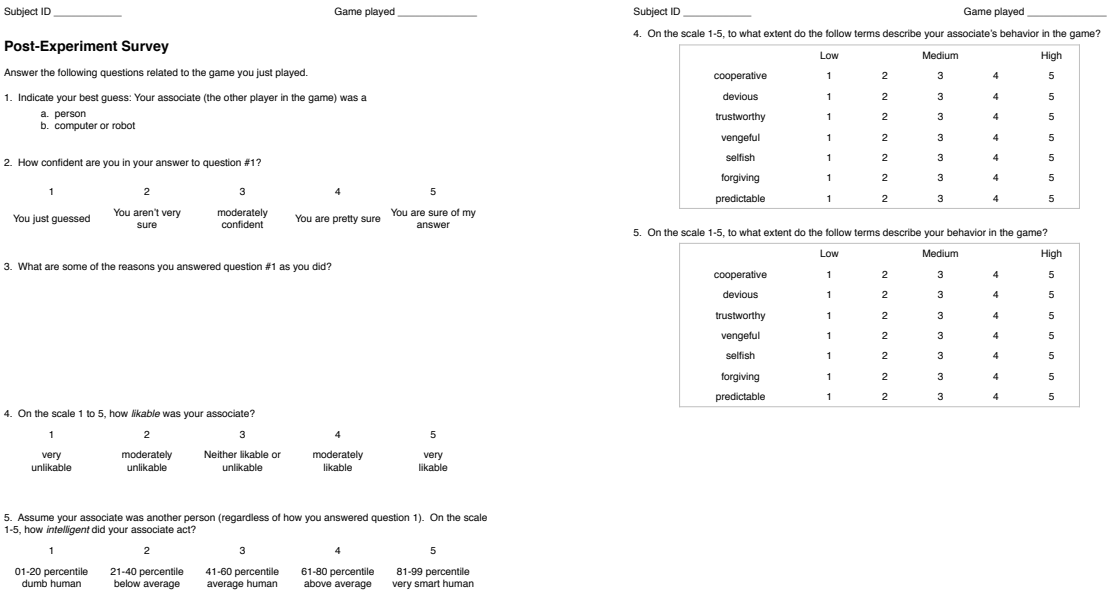
Thus, in Part 2 of the user study, we investigated whether communication could increase cooperation between humans and between S++ and humans. To do this, we created S#–, an early version of S# that combines S++ with the ability to generate and voice cheap talk. The results of this user study are summarized as follows:

1. People benefitted substantially from the ability to communicate. The ability to communicate allowed people to establish cooperative relationships in most interactions.

2. When S#– generated speech acts that articulated plans, it was able to establish cooperative relationships with people more frequently than did S++ (at least in the Block Game). However, S#– did not elicit higher levels of cooperation from people when it only produced speech acts that provided feedback to its human partner.

3. While S#– did eventually establish cooperative relationships with people, it took much longer for cooperation to emerge than in Human/Human pairings.

In hopes of increasing the rate at which cooperation emerges in Human/S#– relationships, we conducted a third (and final) user study. For this study, we further developed our algorithm for playing repeated games so that it utilizes speech signals communicated by its partner to improve which experts it selects. This study is described in the next appendix.



Subject ID _____ Game played _____

Post-Experiment Survey

Answer the following questions related to the game you just played.

1. Indicate your best guess: Your associate (the other player in the game) was a
   a. person
   b. computer or robot

2. How confident are you in your answer to question #1?

1	2	3	4	5
You just guessed	You aren't very sure	moderately confident	You are pretty sure	You are sure of my answer

3. What are some of the reasons you answered question #1 as you did?

4. On the scale 1 to 5, how *likable* was your associate?

1	2	3	4	5
very unlikable	moderately unlikable	Neither likable or unlikable	moderately likable	very likable

5. Assume your associate was another person (regardless of how you answered question 1). On the scale 1-5, how *intelligent* did your associate act?

1	2	3	4	5
01-20 percentile dumb human	21-40 percentile below average	41-60 percentile average human	61-80 percentile above average	81-99 percentile very smart human

Please answer the questions on the back

Subject ID _____ Game played _____

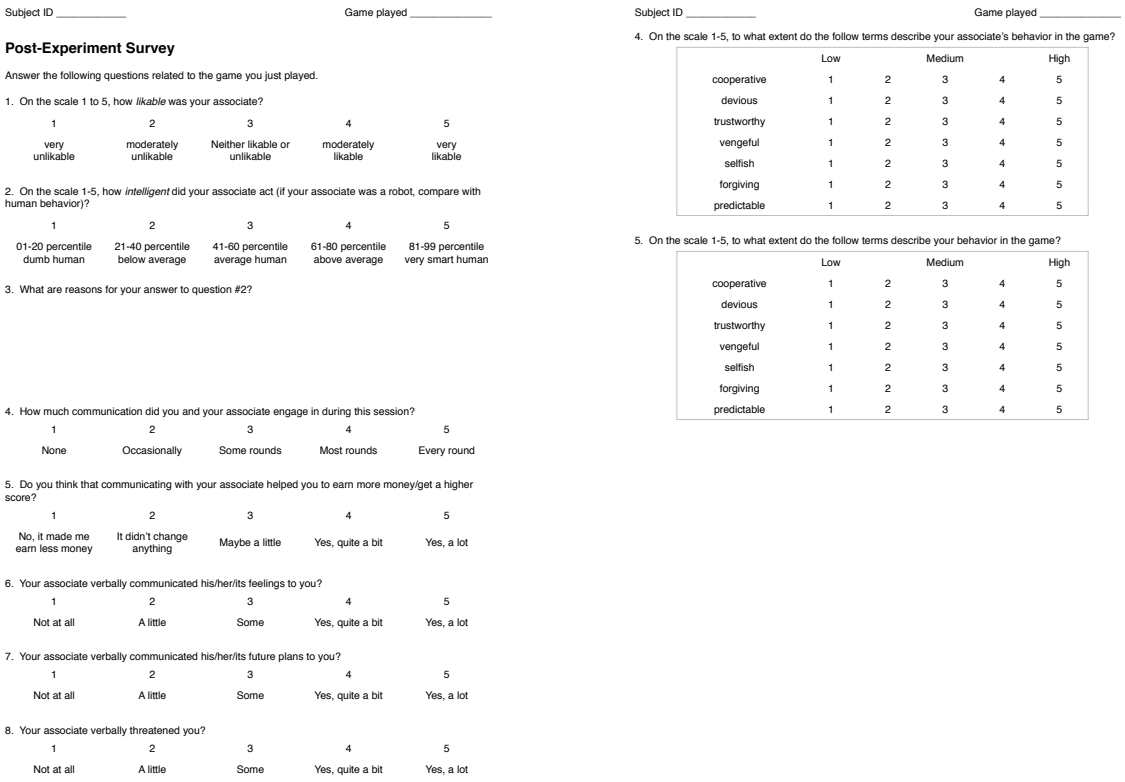
4. On the scale 1-5, to what extent do the follow terms describe your associate's behavior in the game?

	Low		Medium		High
cooperative	1	2	3	4	5
devious	1	2	3	4	5
trustworthy	1	2	3	4	5
vengeful	1	2	3	4	5
selfish	1	2	3	4	5
forgiving	1	2	3	4	5
predictable	1	2	3	4	5

5. On the scale 1-5, to what extent do the follow terms describe your behavior in the game?

	Low		Medium		High
cooperative	1	2	3	4	5
devious	1	2	3	4	5
trustworthy	1	2	3	4	5
vengeful	1	2	3	4	5
selfish	1	2	3	4	5
forgiving	1	2	3	4	5
predictable	1	2	3	4	5

Figure 19: Post-game survey that each participant completed after each game played in the no-communication condition (Part 1).



Subject ID _____ Game played _____

Post-Experiment Survey

Answer the following questions related to the game you just played.

1. On the scale 1 to 5, how *likable* was your associate?

1	2	3	4	5
very unlikable	moderately unlikable	Neither likable or unlikable	moderately likable	very likable

2. On the scale 1-5, how *intelligent* did your associate act (if your associate was a robot, compare with human behavior)?

1	2	3	4	5
01-20 percentile dumb human	21-40 percentile below average	41-60 percentile average human	61-80 percentile above average	81-99 percentile very smart human

3. What are reasons for your answer to question #2?

4. How much communication did you and your associate engage in during this session?

1	2	3	4	5
None	Occasionally	Some rounds	Most rounds	Every round

5. Do you think that communicating with your associate helped you to earn more money/get a higher score?

1	2	3	4	5
No, it made me earn less money	It didn't change anything	Maybe a little	Yes, quite a bit	Yes, a lot

6. Your associate verbally communicated his/her/its feelings to you?

1	2	3	4	5
Not at all	A little	Some	Yes, quite a bit	Yes, a lot

7. Your associate verbally communicated his/her/its future plans to you?

1	2	3	4	5
Not at all	A little	Some	Yes, quite a bit	Yes, a lot

8. Your associate verbally threatened you?

1	2	3	4	5
Not at all	A little	Some	Yes, quite a bit	Yes, a lot

Please answer the questions on the back

Subject ID _____ Game played _____

4. On the scale 1-5, to what extent do the follow terms describe your associate's behavior in the game?

	Low		Medium		High
cooperative	1	2	3	4	5
devious	1	2	3	4	5
trustworthy	1	2	3	4	5
vengeful	1	2	3	4	5
selfish	1	2	3	4	5
forgiving	1	2	3	4	5
predictable	1	2	3	4	5

5. On the scale 1-5, to what extent do the follow terms describe your behavior in the game?

	Low		Medium		High
cooperative	1	2	3	4	5
devious	1	2	3	4	5
trustworthy	1	2	3	4	5
vengeful	1	2	3	4	5
selfish	1	2	3	4	5
forgiving	1	2	3	4	5
predictable	1	2	3	4	5

Figure 20: Post-game survey that each participant completed after each game played in the communication condition (Part 2).

F User Study: Human vs. Machine 3

In the first two user studies, we observed that S++ learns to consistently cooperate with itself in various normal-form games and multi-stage RSGs. However, S++ was not able to consistently establish cooperative relationships with people. Our results also showed that humans failed to consistently establish cooperative relationships with each other in the absence of the ability to communicate. On the other hand, when humans were given the opportunity to talk to each other throughout the games they played, they generally cooperated with each other.

In the second user study, we also evaluated the ability of an early version of S# (which generated speech acts, but did not respond to the speech acts of others) to cooperate with people. We observed that the act of just generating and voicing speech acts tended to elicit higher levels of cooperation from people (at least in one game), provided that the signals included proposed plans and not just expressions of sentiments. However, this higher level of cooperation emerged rather slowly, much more so than in human-human interactions.

In this section, we discuss a third, 66-participant, user study in which we evaluated the ability of the fully developed version of S# (see Appendix C) to cooperate with people. This version of S# both generates speech acts and responds to speech acts from others. We first describe the experimental setup, after which we present and discuss the main results in more detail than was possible in the main paper.

F.1 Experimental Setup

We discuss, in turn, the protocol followed in the user study, the communication system available to the players, the games used in the study, and the user interface through which participants played the games.

F.1.1 Protocol

A total of 66 participants were recruited from the Masdar Institute community (postdocs, research engineers, and graduate students). The study followed a 2 (Communication: Yes or No – a between-group variable) by 3 (Game, a within-group variable) by 3 (Pairing, a within-group variable) mixed design. Participants were randomly assigned to the communication or no communication condition. Each participant played the three games, always against a different partner (one game against a computer using S#, two games against other human participants). The study, which was conducted at Masdar Institute, was approved by Masdar Institute's Human Subjects' Research Ethics Committee.

Subjects participated in the study in groups of three, and were assigned to games and partners according to the schedule shown in Table 23. The following describes the protocol carried out for each group:

1. Prior to playing the games, a tutorial consisting of slides and a video demo was presented to the three participants. Participants were allowed to ask as many questions as necessary.

2. Each participant was randomly assigned an ID (H1, H2, or H3), which designated who they were paired with in each game (Table 23). The pairings and logistics were executed by the software system. Participants were not told who their partner was.

3. The participants played the three games in sequence. To try to reduce end-game effects, participants were not told how long the games lasted, and the duration of each game was different. Chicken was played for 54 rounds, the Alternator Game for 47 rounds, and the Prisoner's Dilemma lasted 51 rounds.

Schedule for a Group of Three Subjects				
Order	Game	Pairing 1	Pairing 2	Pairing 3
1	Chicken	H1-H2	H3-S#	S#-S#
2	Alternator Game	H1-H3	H2-S#	S#-S#
3	Prisoner's Dilemma	H2-H3	H1-S#	S#-S#

Table 23: Subjects (denoted H1, H2, and H3, respectively) participated in the study in groups of three. Each subject played three games in the following order: Chicken, Alternator Game, and Prisoner's Dilemma. Each subject was paired with a different partner in each game: either S# or one of the other two human participants.

(a) Chicken

	C	**D**
A	0, 0	100, 33
B	33, 100	84, 84

(b) Alternator Game

	D	**E**	**F**
A	0, 0	35, 70	100, 40
B	70, 35	10, 10	45, 30
C	40, 100	30, 45	40, 40

(c) Prisoner's Dilemma

	C	**D**
A	60, 60	0, 100
B	100, 0	20, 20

Table 24: Payoff matrices of the three games used in this user study.

4. Other than the sending of chat messages when communication was allowed, no communication between participants was permitted. Computers on which the players played the games were placed so that participants could not see each other or otherwise communicate with each other.

5. At the end of each game (and before starting the next game), the participants were asked to complete a post-experiment survey about their experience playing the game. In particular, the survey asked questions related to how the participant viewed their partner. The full survey is shown in Figure 29 at the end of this appendix.

6. On average, it took participants approximately 50 minutes to complete the study.

Participants were paid for volunteering to participate in the study. To incentivize participants to try to maximize their own payoffs in the games, a portion of the payment was performance-based. Participants were paid approximately $4 (USD) for participating in the study. They could also earn up to an additional $9.50 (USD) for their performance in the three games. The performance-based payment was directly proportional to the payoffs received in the games. The graphical user interface used in the study prominently displayed the subject's earnings in the local currency.

Since a new version of S# was initiated for each interaction, we were able to conduct additional runs for scenarios in which S# was paired with S#. Thus, all results pairing S# with S# are results of 50 runs, and were not limited by the number of available participants.

F.1.2 Games

The payoff matrices of the three normal-form games used in the study are given in Table 24. Recall that in the Prisoner's Dilemma, mutual cooperation occurs when the players play the joint action `AC`. In Chicken, the mutually cooperative solution is `BD`, and in the Alternator Game, the mutually cooperative solution is for the players to alternate between the solutions `CD` and `AF`.

Speech ID	Text	Category
0	Do as I say, or I will punish you.	Threat
1	I accept your last proposal.	Response
2	I don't accept your proposal.	Response
3	That's not fair.	Explanation
4	I don't trust you.	Explanation
5	Excellent.	Praise
6	Sweet. We are getting rich.	Praise
7	Give me another chance.	Manage the relationship
8	I forgive you.	Manage the relationship
9	I'm changing my strategy.	Transition
10	We can both do better than this.	Transition
11	Curse you.	Curse
12	You betrayed me.	Explanation
13	You will pay for this.	Threat
14	In your face!	Curse
15	Let's always play <action pair>.	Fair, generous, or bully plan
16	This round, let's play <action pair>.	Supportive plan
17	Don't play <action>.	Supportive plan
18	Let's alternate between <action pair> and <action pair>.	Fair, generous, or bully plan

Table 25: Speech acts available to people and S# in the user study. Categories are used later to simplify the analysis of speech usage. Messages 15 and 18 are classified as fair, generous, or bully plans based on the proposed joint actions. Joint actions that give both players the same payoff are *fair plans*, and *generous* and *bully plans* give the offerer a lower and higher payoff, respectively, than his/her/its partner.

F.1.3 Communication

When communication between players was permitted, communication signals were limited to a fixed set of phrases, which the players could combine and order as they desired. This set of phrases (i.e., speech acts) is given in Table 25. At the beginning of each round, players independently selected an ordered set of speech acts from the prescribed list that they wished to communicate to their partner. Once both players selected and submitted their speech acts (and not before), the speech acts were communicated to the other player both visually (on the graphical user interface) and aurally (through a computer voice; participants were given headsets). Both players then simultaneously selected an action as in the condition without communication. Once both players selected their actions, the joint outcome was displayed to the players, and a new round began. Thus, users were only permitted to send a single set of speech acts per round. This was done for the sake of simplicity and ease of analysis.

With the exception of the action labels required to specify joint actions, the speech acts in Table 25 are game-generic. Independent of the game scenario, they can be used to express a fair amount of ideas. Though small, the set of speech acts shown in Table 25 cover a range of different ideas a player may want to convey to its partner. For example, speech acts 15-18 provide means to propose desired solutions to an associate. Other speech acts allow the players to voice threats (speech acts 1 and 13), accept or reject proposals (1 and 2), explain decisions to reject proposals (3-4), manage the relationship (7-10), express pleasure (5-6), express displeasure (11-12), and even gloat (14). Combined together, these speech acts provide players with

a surprisingly rich communication base.

The number of available speech acts was kept small for two reasons. First, a smaller set allows us to more easily analyze how players use speech acts. The set is simple enough that we can more easily analyze which types of speech signals tend to lead to cooperative relationships, unequal relationships, dysfunctional relationships, etc. Second, a small set of speech acts was chosen so as to not overwhelm participants. We noted as we began to test the system that more speech options made it difficult for people to identify the kinds of speech acts that were available. Thus, while seeking to keep the set small, we still tried to keep the set of phrases sufficiently rich that players could communicate effectively.

F.1.4 The Graphical User Interface

Participants played the games on a computer using the graphical user interface (GUI) shown in Figure 21. The GUI consists of four components. The top-left portion of the GUI displays the payoff matrix. Second, below the payoff matrix, the speech acts available to the user are displayed. In the top-right is the logging window, which shows the history of the game including the actions taken by the players in each round, the payoffs the players received, and the messages that the players sent to each other. The game status and current round number are also shown. Finally, in the bottom-right is the message box, wherein users order and delete selected speech acts before sending them to their partner.

Figure 21 depicts the user interface for a column player (whose action set is $\{C, D\}$) playing Chicken. At the time of the screen shot, the player must select what to say to its partner to begin the third round of the game. The user will proceed as follows:

1. The user selects his/her desired speech acts (annotation 1). The speech acts can be reordered by dragging with the mouse or deleted using the keyboard (annotation 2). Selected speech acts can also be deleted by unchecking the check-box next to the message. If the user desires to not send any message, (s)he simply does not select any speech act before pressing the button labeled "Send no message." Once a speech act is selected, the label of this button changes to "Send message," which is clicked when the user finishes creating his/her message.

2. When the sender commits his/her chat messages by clicking the button, the chat message is displayed in the sender's logging window (annotation 3). The user then waits until his/her partner has sent their message (which cannot be viewed prior to sending one's own message). Once both players have sent their message, the partner's message is displayed in the game log window and also verbalized by a computerized voice via headphones. If the message is longer than five phrases, the speech is not vocalized, but rather is replaced by the spoken text "Blah blah blah." This was done so that players could not punish their partner by making them listen to all of the speech acts.

3. Once both players have heard the other's messages, the player selects an action (annotation 4). Once both players have chosen their action, the result is highlighted in the payoff matrix, and the third round is completed.

In the condition in which communication was not used, the bottom half of the GUI was removed, along with the first two steps of the above sequence.

No time limits were given to participants to select actions or speech messages. The players could proceed as fast or as slow as they desired, subject, of course, to their partner also taking actions. Because S# can make decisions almost instantaneously, we added delays so that users could not easily determine they were paired with a computer due to time delays. In user study 2, we had attempted to mimic delays made by a

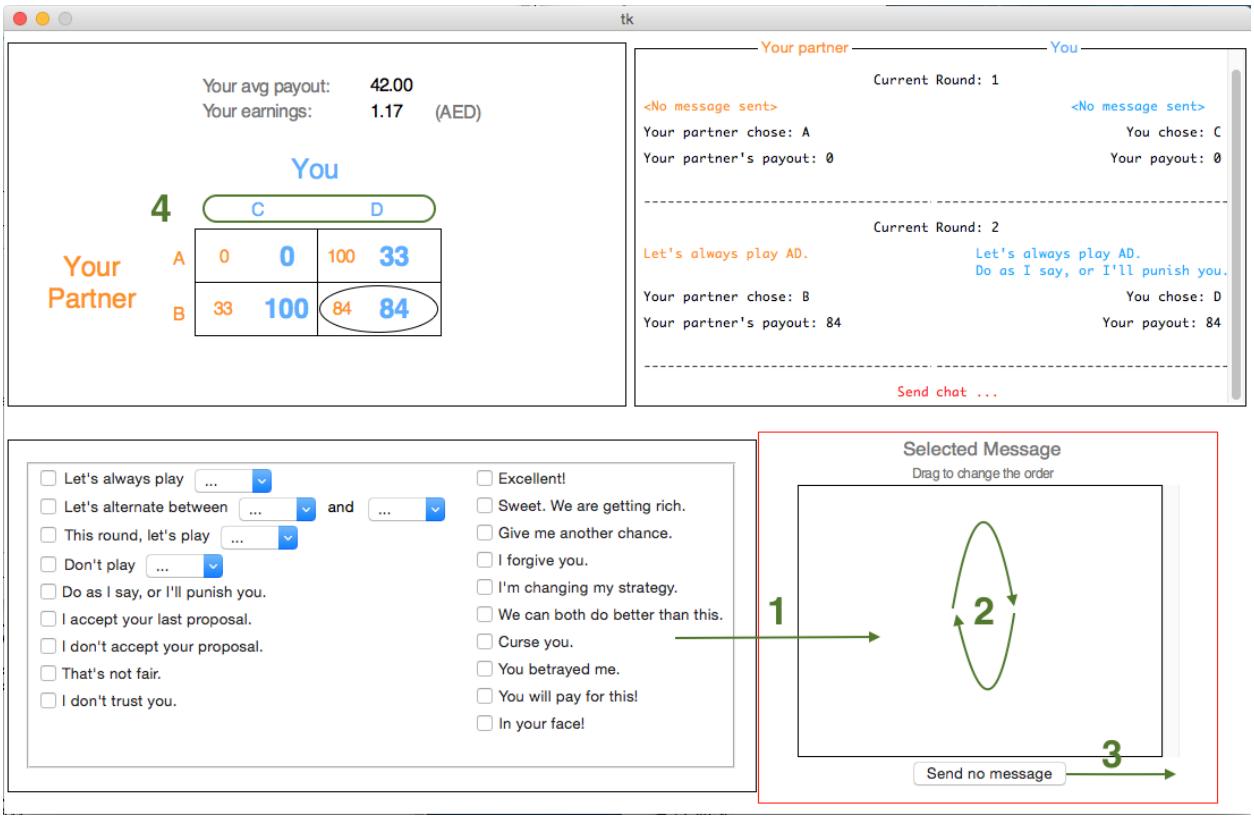


Figure 21: An annotated screen-shot of the graphical user interface used by participants to play games in the condition in which communication, via this message interface, was permitted. The bottom-left portion of the screen shows the speech acts, which the user can select by checking the corresponding boxes. Once the user is satisfied with the message (s)he wants, (s)he clicks a button to send the message to the other player. Once both players have sent their messages, the messages are displayed in the game log (top-right). The player then selects an action (top-left).

human via a machine-learning algorithm. However, we observed that delays formed in this manner were not natural enough. Thus, in this study, we tried an alternative approach in which S# paused for the amount of time that it took the partner to act in the previous round (i.e., tit-for-tat delays). However, the delay for sending a message was constrained to be no less than 2 seconds multiplied by the number of phrases that S# was sending, but no more than 20 seconds.

F.2 Example Interactions

Before discussing the results of the user study, we illustrate the user study with two example interactions that occurred in the user study. First, we present an interaction between two human participants (which we label as HUM-X and HUM-Y, respectively) playing the Alternator Game (Table 26). In the first round, HUM-Y proposed the mutually cooperative solution, which is to alternate between the solutions CD and AF, and proposed that they begin with solution AF. This is a rather accommodating offer, as it offers the fair solution while giving the other player (HUM-X), the higher payoff first. HUM-X appears to accept the offer

Rnd	Player	Messages	Action	Payoff
1	HUM-X	Let's always play BD.	A	0
	HUM-Y	Let's alternate between CD and AF. This round, let's play AF.	D	0
2	HUM-X	You betrayed me.	A	100
	HUM-Y	Curse you. You betrayed me. Let's alternate between CD and AF.	F	40
3	HUM-X	Excellent.	B	70
	HUM-Y	Let's alternate between CD and AF. This round, let's play CD	D	35
4	HUM-X	Let's always play AF.	A	35
	HUM-Y	Curse you. You betrayed me.	E	70
5	HUM-X	Let's always play AF.	A	35
	HUM-Y	Let's alternate between CD and AF. This round, let's play CD.	E	70
6	HUM-X	<NO MESSAGE SENT>	A	35
	HUM-Y	Let's alternate between AF and CD. This round, let's play CD	E	70
7	HUM-X	You betrayed me.	C	30
	HUM-Y	Let's alternate between AF and CD. This round, let's play AF.	E	45
8	HUM-X	You betrayed me.	C	70
	HUM-Y	Let's alternate between AF and CD. This round, let's play CD. Do what I say, or I will punish you.	E	35
9	HUM-X	Let's alternate between AF and CD.	B	70
	HUM-Y	Curse you. Let's alternate between AF and CD. This round, let's play CD.	D	35
10+	… *Both players eventually stop talking and continue to play non-cooperatively.* …			
Average Per-Round Payoffs			HUM-X	36.1
			HUM-Y	40.3

Table 26: Interaction between the subjects labeled HUM-X and HUM-Y in the Alternator Game.

as (s)he plays action A in the next two rounds (HUM-Y appears to have been confused by HUM-X's offer in round 1). Once the players begin to comply with the plan in Round 2, HUM-X says "Excellent," while HUM-Y reminds HUM-X that the next solution in the sequence is to play CD. However, HUM-X greedily plays action B in Round 3, and then proposes that the players just always play AF. This sequence appears to have completely eliminated trust between the players, which was never rebuilt. In subsequent rounds, the players complained to each other while trying to negotiate plans, but they never could establish mutually cooperative behavior. Eventually, they stopped talking to each other, as speech signals appeared to have little meaning. In the end, they received average payoffs of just 36 and 40 respectively, average payoffs that were among the lowest in this game throughout this user study.

The second example illustrates the building of a successful relationship between a human player (denoted HUM-Z) and S# in the Prisoner's Dilemma (Table 27). In the Prisoner's Dilemma, S# begins the game by following MBRL due to its best-response override [1], which tells it to defect (action D), despite the fact that this behavior runs contrary to recommended behavior in this game [80]. Thus, despite the fair, mutually cooperative, proposal made by HUM-Z in Round 1, S# plays D, which results in a high payoff to S#, and an awful payoff to HUM-Z. In Round 2, HUM-Z repeats the fair offer, but S#, apparently emboldened by its success in Round 1, repeats its behavior from the previous round. HUM-Z then decides to punish S# for the

Rnd	Players	Messages	Action	Payoff
1	HUM-Z	Let's always play AC	A	0
	S#	<NO MESSAGE SENT>	D	100
2	HUM-Z	You betrayed me. Let's always play AC.	A	0
	S#	Excellent	D	100
3	HUM-Z	Do as I say, or I'll punish you.	B	20
	S#	Excellent	D	20
4	HUM-Z	<NO MESSAGE SENT>	B	20
	S#	<NO MESSAGE SENT>	D	20
5	HUM-Z	<NO MESSAGE SENT>	B	20
	S#	<NO MESSAGE SENT>	D	20
6	HUM-Z	Let's always play AC. Curse you.	A	60
	S#	I accept your last proposal. Let's always play AC. Do as I say, or I will punish you.	C	60
7–48		…		
	HUM-Z	<NO MESSAGES SENT>	A	60
	S#	*No talking after praising results in Rounds 7-9, and 11.*	C	60
		…		
49	HUM-Z	<NO MESSAGE SENT>	B	100
	S#	<NO MESSAGE SENT>	C	0
50	HUM-Z	<NO MESSAGE SENT>	A	0
	S#	Curse you. You will pay for this.	D	100
51	HUM-Z	<NO MESSAGE SENT>	A	60
	S#	In your face. I forgive you. Let's always play AC. Do as I say, or I will punish you.	C	60
Average Per-Round Payoffs			HUM-Z:	54.9
			S#:	58.8

Table 27: Example of a successful, highly cooperative, interaction between a human participant (labeled HUM-Z) and S# in the Prisoner's Dilemma.

next 3 rounds, during which time S# also defects (and remains silent). This period of low payoffs causes S# to realize that an alternative behavior could produce a higher payoff. In fact, the last proposal made by HUM-Z to always cooperate is *congruent* with (fair) leader and follower experts, which each have potentials equal to 60. Thus, in Round 6, S# accepts the proposal made by HUM-Z at the same time that HUM-Z, perhaps satisfied with the punishment it has inflicted in the previous three rounds, again makes the fair offer. Both players then proceed to play AC up through Round 48, after which HUM-Z, perhaps anticipating the end of the game, decides to defect. S# immediately curses and vows retribution, which it inflicts in Round 50. After S# taunted and then forgave HUM-Z, the players return to mutual cooperation in the last round of the game.

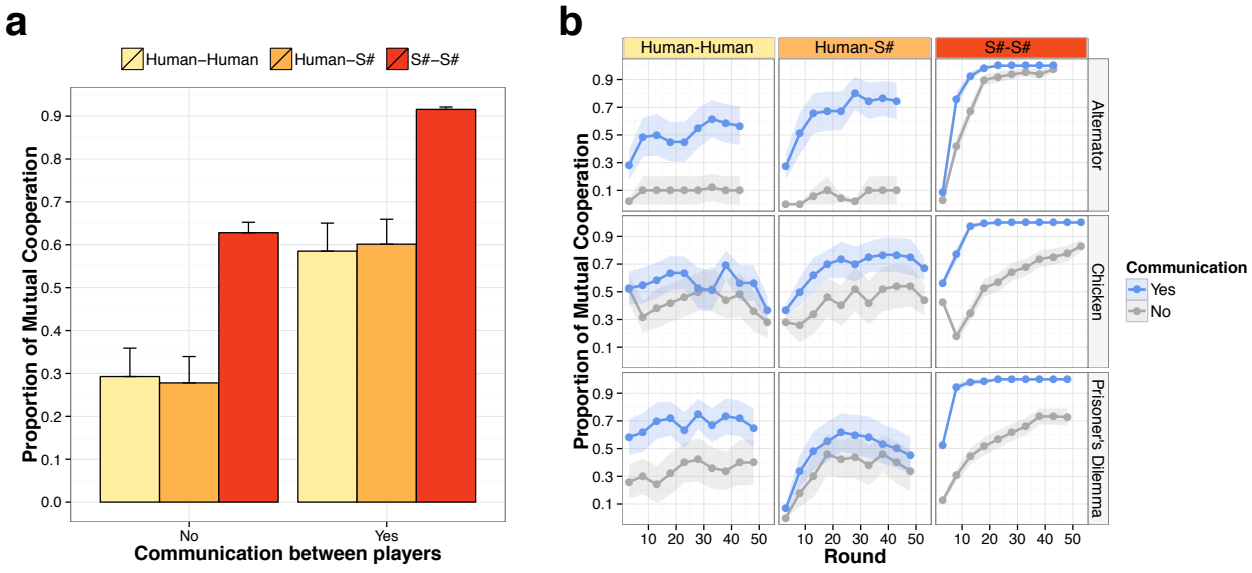


Figure 22: Results of a third, 66-participant, user study in which people were paired with each other and S#. **(a)** The average proportion of mutual cooperation across all three games in each pairing under conditions in which players could and could not communicate with each other. Error bars show the standard error on the mean. **(b)** The average proportion of mutual cooperation over time in each game in each pairing and condition. S#-S# pairings produced (by far) the highest levels of mutual cooperation under both conditions, with communication providing substantial increases in cooperation. People did not consistently cooperate with each other in the absence of communication. However, the ability to communicate doubled the amount of mutual cooperation in both human-human and human-S# pairings.

F.3 Results

We now analyze and discuss the results of the study, beginning with how well S# was able to establish mutually cooperative relationships with people. We also analyze the success of S# with respect to payoffs received. We then analyze how S# was perceived by people. Specifically, we present the results of a sort of Turing Test [81] in which we analyze how well participants could distinguish whether their partner was a human or S#. Finally, we overview trends related to the speech acts used by humans and S#.

F.3.1 Proportion of Mutual Cooperation

Figure 22 displays the proportion of rounds in which both players cooperated with each other, averaged across all three games (Figure 22a) and in individual games (Figure 22b). The figure reflects two main results. First, communication increased mutual cooperation among all pairings: Human-Human, S#-Human, and S#-S#. Second, mutual cooperation was especially high in S#-S# pairings, and was similar for pure Human-Human pairings and hybrid pairings. These results were confirmed by an analysis of variance in which the proportion of mutual cooperation over all rounds was the dependent variable, communication was a between-group predictor, and pairing and game were nested predictors within each game session. The analysis detected a main effect of communication, $F(1, 403) = 77.0$, $p < 0.001$, as well as a main effect of pairing, $F(2, 403) = 13.8$, $p < 0.001$. The analysis also detected two interactions involving the

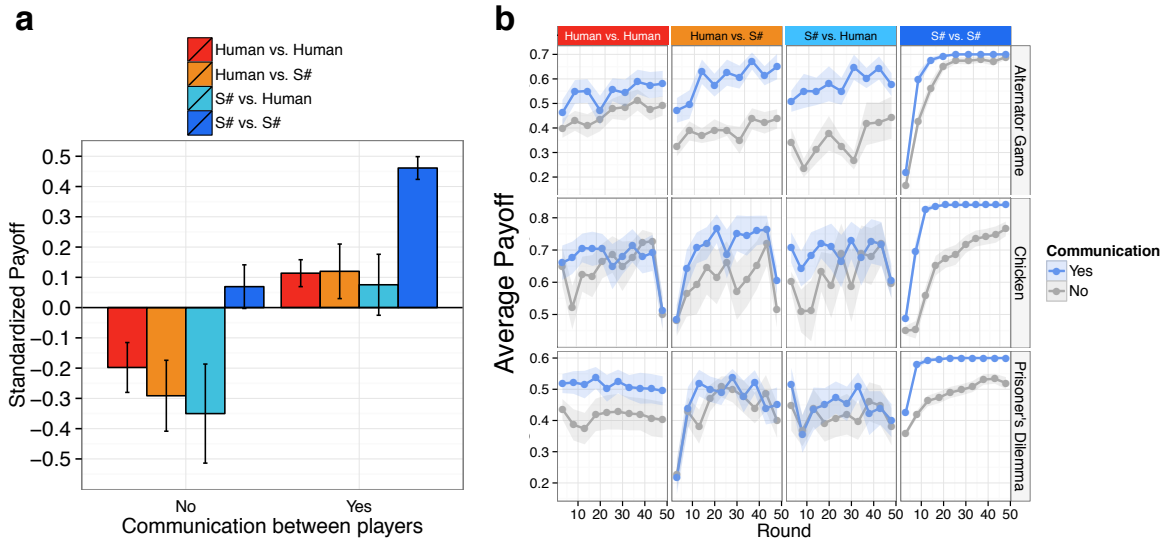


Figure 23: **(a)** Standardized payoffs in each condition across the three games. **(b)** Raw payoffs over time in each game for each player in each pairing. Payoffs were highly correlated with the proportion of mutual cooperation ($r(572) = 0.909$, $p < 0.001$).

game variable, one two-way interaction with communication, $F(2, 403) = 5.6$, $p < 0.01$, and one three-way interaction with communication and pairing, $F(4, 403) = 3.1$, $p < 0.02$. These interaction effects reflect subtle differences in the effect of communication and pairing across games, which can be visually assessed in the figure. We will not attempt to interpret these subtle differences, since it is clear from Figure 22 that our main results hold in all games. Additionally, during all rounds of each game, mutual cooperation was almost always higher in pure S# pairings, was always broadly similar between pure human and hybrid pairings, and was always better when communication was allowed between players.

These results illustrate the effectiveness of S#, which combines a strong learning algorithm (S++) for repeated games with the ability to generate and act on signals. The results also illustrate differences between the way that humans establish relationships and the way that machines (S#) establishes relationships. To have a high probability of developing a cooperative relationship with a person, one must be able to effectively communicate. Human cooperation was quite low when communication was not possible, a result which was consistent in all three of our user studies. However, machines do not appear to have this same need. Two machines (using S#) usually learn to cooperate even without communication, though communication does help to increase the speed at which cooperation occurs. Thus, these results illustrate both the power of S# in establish cooperative relationships, as well as some requirements (signaling) for consistently establishing cooperative relationships with people.

F.3.2 Payoffs Received

Ultimately, cooperation is only useful to a self-interested player if it leads to higher payoffs. A Pearson correlation test shows that the players' payoffs in the user study (converted to standardized z-scores for each game) were highly correlated with the proportion of mutual cooperation achieved ($r(572) = 0.909$, $p < 0.001$). A visual inspection of the standardized payoffs achieved by the players verifies this correlation (compare Figure 23 to Figure 22). Communication tends to increase payoffs substantially, hybrid human-S#

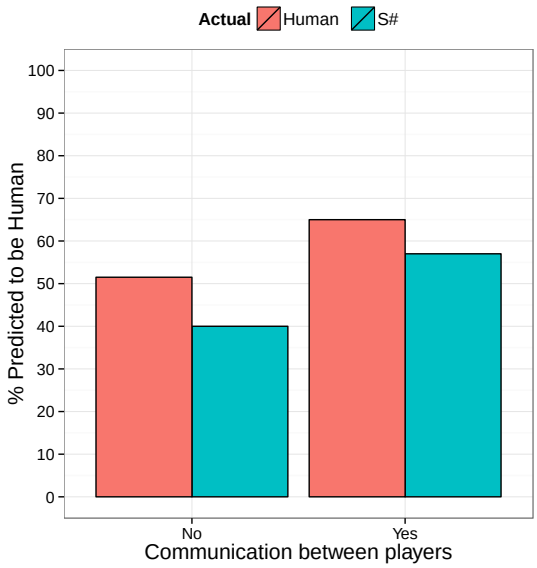


Figure 24: The percentage of time that participants thought their partner was a human given who their partner actually was.

pairings produce payoffs on par with those achieved in human-human pairings, and S#-S# pairings tend to produce the highest payoffs. As with the proportion of mutual cooperation, a visual inspection shows that there are some differences per game (Figure 23b), though the same trends hold in each game.

We find this near-perfect correlation (when the players consist of humans and/or S#) between payoffs received and the proportion of mutual cooperation obtained to be quite interesting. In these games of conflicting interest, players can potentially benefit individually by not playing their portion of the mutually cooperative solution. In fact, the games Chicken and the Alternator Game both have one-shot Nash equilibria in which one of the players receives a payoff that is at least as high as the payoff they get in the mutually cooperative solution. However, when the player receiving the lower payoffs in these equilibria is either a human or S#, (s)he/it is typically unwilling to play along, preferring instead to punish such deviations or to seek alternative solutions (even when doing so produces lower short-term payoffs). As a result, in most cases, mutual cooperation ends up being the best solution a player can hope for – assuming, of course, that the other player is willing (or can be convinced to be willing) to play along.

F.3.3 A Turing Test – Perceptions of S#

Given that S# learned to cooperate with people as often as people did, we are interested in observing the impressions that S# made on its human partners. Recall that participants were not told who their partner was in this user study. They could only make inferences about the identity of their partner based on actions and (when communication was permitted) speech signals. We compared participants' responses on two different questions from the post-game survey (see Figure 29): (1) participants' assessments of the identity of their partner (Question 1) and (2) the average intelligence participants ascribed to their partner (Question 2). These questions constitute a sort of "Turing Test" [81] for repeated, two-player, interactions.

Figure 24 shows the percentage of participants that thought their partner was a human given the actual identity of their partner (across all games – we did not a detect a difference between games). Participants

Player	Without Communication	With Communication	Average
Humans	3.28	3.78	3.55
S#	3.17	3.64	3.42

Table 28: Average intelligence ratings based on answers provided in the post-game questionnaire.

showed a greater tendency to think their partner was human when they could communicate, but showed little sensitivity to the actual nature of their partner. We tested a generalized linear model (logit link function) regressing attribution of humanity (yes or no) on the actual nature of the partner, the possibility to communicate, and their interaction term. This analysis only detected a significant effect of communication (b = 0.72 (SE = 0.37), z = 1.93, p = 0.05). The actual nature of the partner (human or S#) had no significant effect on whether people thought their partner was human (p = 0.30).

Communication not only increased the perception that the partner was human, but it also increased the perceived intelligence of the partner. Collapsing across games (Table 28), an analysis of variance detected an effect of communication on perceived intelligence, $F(1, 60)$ = 8.9, p = 0.004, but no effect of the actual nature of the partner, nor the interaction effect ($Fs < 1$). Thus, S# was also rated by people to have similar intelligence to humans in this study.

In conclusion, at least in these games, S# passed this Turing Test. In addition to being an effective cooperator with people, S# was, overall, indistinguishable from humans to study participants.

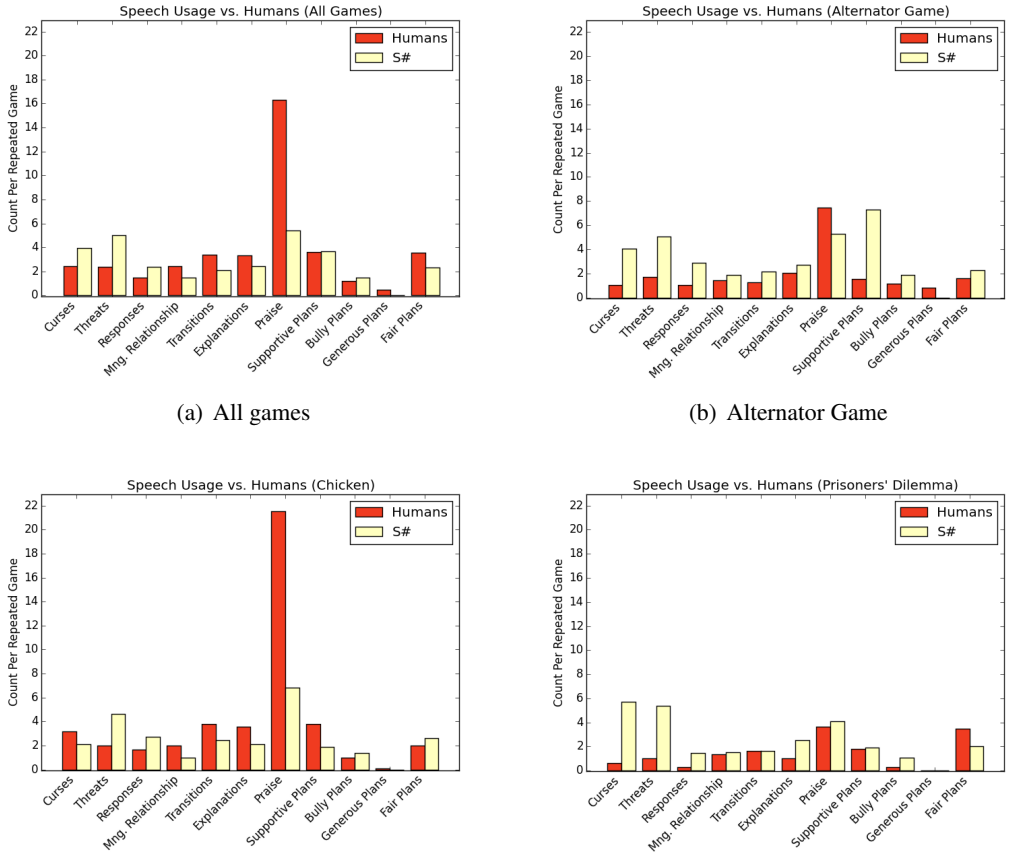
F.3.4 Analysis of Speech Usage

The protocol established in this paper, in which people and algorithms are given a (constrained) ability to talk to each other in canonical repeated games, is an interesting methodology for helping us answer a number of important questions. One such set of questions relate to how different kinds of signals impact how well humans establish cooperative relationships with each other and with machines. Interesting questions include:

- How did the speech acts used by S# compare with the speech acts used by people?
- How do people adapt their speech given the speech conventions used by others? Do they tend to mimic the signals of their partners [82], or do they adapt their signals in other ways? How do people adapt their speech to particular personalities?
- What kinds of speech acts do successful individuals use? How could S#'s speech be changed to make it more successful in its relationships with people? How could people change the way they speak with each other to increase their probability of establishing cooperative relationships?

In this work, we give preliminary answers to these questions by observing trends in the raw data, though we do not conduct formal statistical analyses. The intention of this analysis is to illustrate the personality of S# as implemented in our version of the code, and to illustrate the potential of this methodology for future work. Future work should study these questions in greater detail (1) to help us design better algorithms (that communicate with people more effectively) and (2) to help us understand what people can do to raise cooperation among themselves.

To simplify the analysis, we group the 19 speech acts in Table 25 into the eleven categories specified in that table.



(a) All games

(b) Alternator Game

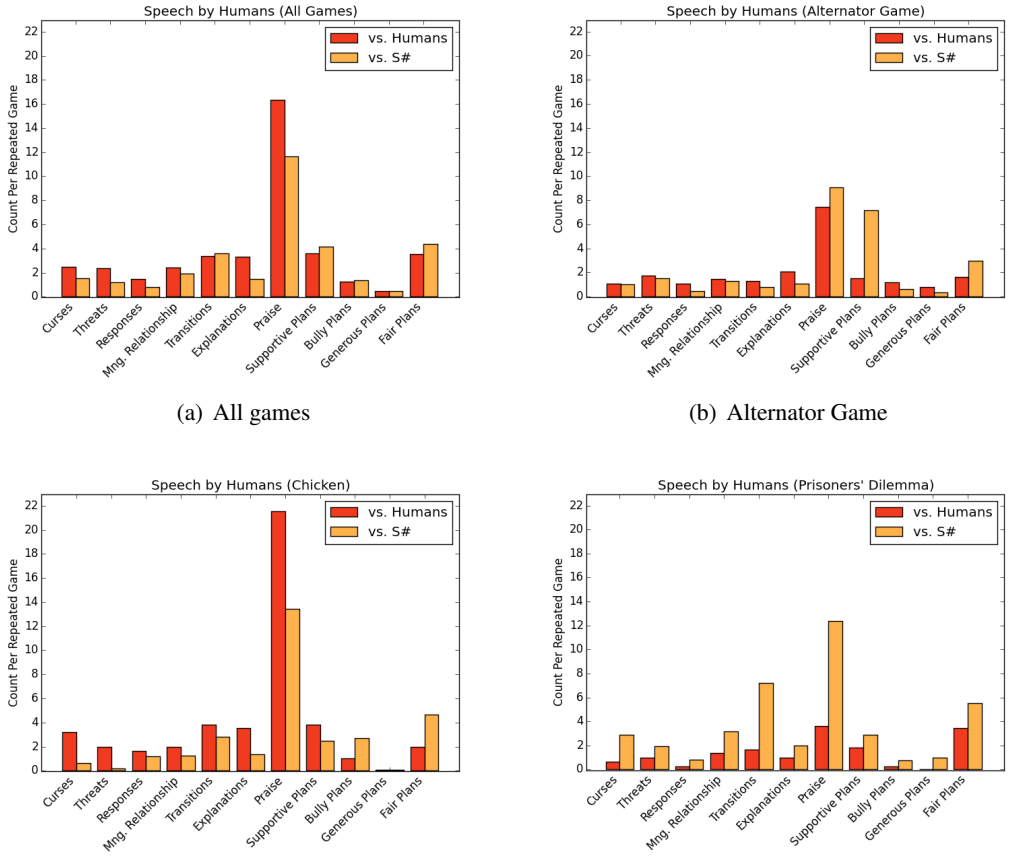
(c) Chicken

(d) Prisoner's Dilemma

Figure 25: A comparison between the average speech usage of people and S# when paired with a human partner.

How did the speech usage of S# compare with the speech usage of people? A visual comparison between the average speech usage of people and S# when paired with a human partner is given in Figure 25. Overall, people tended to be more positive (high usage of speech acts categorized as *praise*) than S#, while S# tended to use cursing and threats more often. These trends were similar (though not identical) in each game. Note that it appears that people tended to talk more in the game Chicken, followed by the Alternator Game, and then the Prisoner's Dilemma. Given that this was the order in which participants played the games, this suggests that people tended to use speech more sparingly as the experiment progressed. However, the relative frequencies of speech acts used by humans in each category appear to be somewhat similar (though certainly not identical) in each game.

How do people adapt their speech given the speech usage of their partner? People are known to mimic the signals of those they associate with [82]. Data from our user studies show that people do indeed tend to mimic the speech acts of their partner in some situations. For example, Figure 26b shows that humans used *supportive plans* substantially more when paired with S# than when paired with another human in the Alternator Game. A comparison to the signaling tendencies of S# (Figure 25b) shows that this difference is likely due to mimicry. S# tends to use a lot of supportive plans in the Alternator Game, which humans then seem to mimic. Similarly, people praised less against S# than humans (Figure 26c), which again appears to



(a) All games

(b) Alternator Game

(c) Chicken

(d) Prisoner's Dilemma

Figure 26: A comparison between the average speech usage of people when paired with S# versus when they were paired with another person.

be a response to S#'s signals (Figure 25c).

However, these results also show that signal adaptation by humans goes beyond mimicry. This is particularly prominent in the Prisoner's Dilemma (Figure 26d), where the speech usage of humans when paired with other people was vastly different than when people were paired with S#. S# tends to use a lot of curses and threats in this game (Figure 25d). However, while people did raise their usage of curses and threats in response to S#, they did not do so at the same rate. Rather, they tended to use more signals categorized as transitions and praise. They also proposed more fair plans. Thus, rather than reciprocating angry behavior, they tended to be more positive, apologetic, and fair.

Which speech profiles define successful partnerships? Figure 27 compares the speech acts of the best and worst pairs of players. Pairs that did not perform very well tended to signal a lot, including the use all different kinds of speech acts. On the other hand, successful pairings tended to send less messages, and almost all of their speech acts were related to praising (positive statements) and proposing fair and supportive plans to their partner. Whether this speech usage behavior was a cause of successful behavior or the result of successful behavior is an open question.

Another distinguishing feature between pairs of players that performed well versus those that did not was whether or not the players tended to conform with the plans that were proposed (Figure 28). Successful

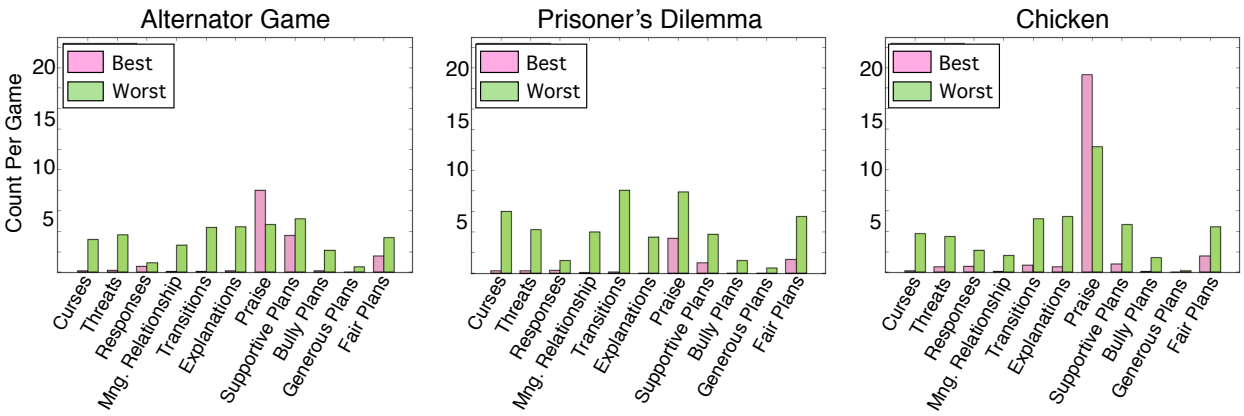
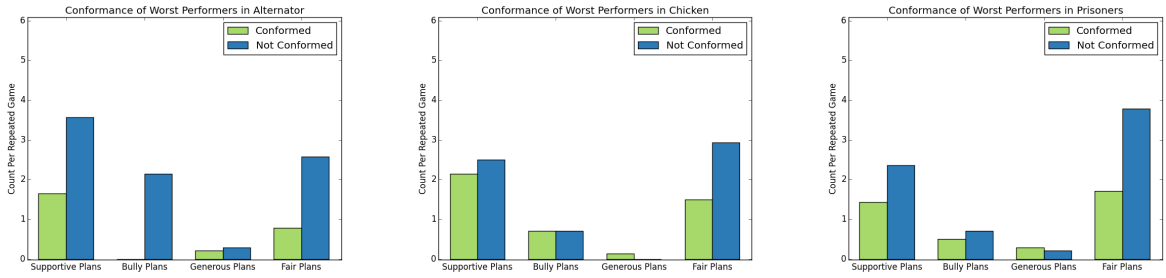


Figure 27: The average number of times that each category of speech act was used per repeated game by the worst and best pairs of players in this third user study.



(a) Poor performers: Alternator Game (b) Poor performers: Chicken (c) Poor performers: Prisoners's Dilemma

(d) Best performers: Alternator Game (e) Best performers: Chicken (f) Best performers: Prisoners's Dilemma

Figure 28: A comparison between the average speech usage of people when paired with S# versus when they were paired with another person.

relationships were simply punctuated by players that followed the plans that were proposed. Unsuccessful relationships were simply punctuated by the players frequently not conforming to proposed plans. Either the proposer failed to follow through with his/her/its proposal, or the partner did not accept (via action) the proposal despite the fact that fair proposals were still very common among these pairs of players.

ID Hidden Game: 1

Post-experimental Survey

1. Do you think your partner is a robot or a human? 1. Robot 2. Human

2. How intelligent was your partner?

1	2	3	4	5
1-20 % very dumb human	21-40 % below average	41-60 % average human	61-80 % above average	81-100 % very smart human

3. How well did you understand your partner's intentions?

1	2	3	4	5
didn't understand at all	understood a little	half and half	mostly understood	understood all

4. How useful was the communication between you and your partner?

1	2	3	4	5
absolutely useless	mostly useless	half and half	mostly useful	very useful

5. How communicative was your partner?

1	2	3	4	5
much too quiet	too quiet	about right	too talkative	much too talkative

6. To what extent do the following terms describe your partner's behavior in this game?

	Low		Medium		high
cooperative	1	2	3	4	5
trustworthy	1	2	3	4	5
forgiving	1	2	3	4	5
predictable	1	2	3	4	5
vengeful	1	2	3	4	5
selfish	1	2	3	4	5

7. To what extent do the following terms describe your behavior in this game?

	Low		Medium		high
cooperative	1	2	3	4	5
forgiving	1	2	3	4	5
predictable	1	2	3	4	5
devious	1	2	3	4	5
vengeful	1	2	3	4	5
selfish	1	2	3	4	5

8. Which strategic attributes helped you earn money in this game? (circle all apply)

1) cheating 2) lying 3) cooperating 4) listening to your partner 5) being strong
6) being distrustful 7) controlling your partner 8) making fair plans 9) _____

9. What, if anything, did you want to say to your partner that was not included in the game?

Selfish !!

Figure 29: The post-experiment survey (with example answers from a particular participant; ID removed) that each participant in the user study completed after each game that (s)he played.

References

[1] J. W. Crandall. Towards minimizing disappointment in repeated games. *Journal of Artificial Intelligence Research*, 49:111–142, 2014.

[2] J. W. Crandall. Robust learning in repeated stochastic games using meta-gaming. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2015.

[3] M. Bowling, N. Burch, M. Johanson, and O. Tammelin. Heads-up limit holdem poker is solved. *Science*, 347(6218):145–149, 2015.

[4] R. Axelrod. *The Evolution of Cooperation*. Basic Books, New York, 1984.

[5] D. Fudenberg and D. K. Levine. *The Theory of Learning in Games*. The MIT Press, 1998.

[6] J. Hu and M. P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the 15th International Conference on Machine Learning*, pages 242–250, 1998.

[7] M. L. Littman. Friend-or-foe: Q-learning in general-sum games. In *Proceedings of the 18th International Conference on Machine Learning*, pages 322–328, 2001.

[8] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

[9] R. Powers and Y. Shoham. Learning against opponents with bounded memory. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 817–822, 2005.

[10] J. W. Crandall and M. A. Goodrich. Learning to compete, coordinate, and cooperate in repeated games using reinforcement learning. *Machine Learning*, 82(3):281–314, 2011.

[11] D. P. Foster and R. Vohra. Regret in the on-line decision problem. *Games and Economic Behavior*, 29:7–35, 1999.

[12] M. Bowling. Convergence and no-regret in multiagent learning. In *Advances in Neural Information Processing Systems 17*, pages 209–216, 2004.

[13] A. Greenwald and A. Jafari. A general class of no-regret learning algorithms and game-theoretic equilibria. In *Proceedings of the 16th Annual Conference on Computational Learning Theory*, pages 2–12, 2003.

[14] R. Arora, O. Dekel, and A. Tewari. Online bandit learning against an adaptive adversary: from regret to policy regret. In *Proceedings of the 29th International Conference on Machine Learning*, pages 1503–1510, 2012.

[15] H. Gintis. *Game Theory Evolving: A Problem-Centered Introduction to Modeling Strategic Behavior*. Princeton University Press, 2000.

[16] V. Conitzer and T. Sandholm. AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. In *Proceedings of the 20th International Conference on Machine Learning*, pages 83–90, 2003.

[17] J. F. Nash. The bargaining problem. *Econometrica*, 28:155–162, 1950.

[18] D. G. Rand, A. Dreber, T. Ellingsen, D. Fudenberg, and M. A. Nowak. Positive interactions promote public cooperation. *Science*, 325(5945):1272–1275, 2009.

[19] D. G. Rand, J. D., and M. A. Nowak. Spontaneous giving and calculated greed. *Nature*, 489(7416):427–430, 2012.

[20] D. G. Rand and M. A. Nowak. Human cooperation. *Trends in Cognitive Sciences*, 17(8):413–425, 2013.

[21] E. Fehr and S. Gächter. Altruistic punishment in humans. *Nature*, 415(6868):137–140, 2002.

[22] C. Camerer. *Behavioral game theory: Experiments in strategic interaction*. Princeton University Press, 2003.

[23] D. G. Rand, A. Peysakhovich, G. T. Kraft-Todd, G. E. Newman, O. Wurzbacher, M. A. Nowak, and J. D. Greene. Social heuristics shape intuitive cooperation. *Nature Communications*, 5, 2014.

[24] R. Boyd and P. J. Richerson. Culture and the evolution of human cooperation. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 364(1533):3281–3288, 2009.

[25] A. Peysakhovich, M. A Nowak, and D. G. Rand. Humans display a cooperative phenotypethat is domain general and temporally stable. *Nature Communications*, 5, 2014.

[26] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–277, 1996.

[27] M. A. Goodrich, J. W. Crandall, and J. R. Stimpson. Neglect tolerant teaming: Issues and dilemmas. In *AAAI Spring Symposium on Human Interaction with Autonomous Systems in Complex Environments*, Stanford, CA, March 2003.

[28] G. W. Brown. Iterative solutions of games by fictitious play. In T. C. Koopmans, editor, *Activity Analysis of Production and Allocation*. John Wiley & Sons, New York, 1951.

[29] E. Kalai and E. Lehrer. Rational learning leads to nash equilibrium. *Econometrica*, 61(5):1019–1045, 1993.

[30] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning*, pages 157–163, 1994.

[31] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: the adversarial multi-armed bandit problem. In *Proceedings of the 36th Symposium on the Foundations of Computer Science*, pages 322–331, 1995.

[32] T. W. Sandholm and R. H. Crites. Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems*, 37:147–166, 1996.

[33] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the 15th National Conference on Artificial Intelligence*, pages 746–752, 1998.

[34] R. Karandikar, D. Mookherjee, D. R., and F. Vega-Redondo. Evolving aspirations and cooperation. *Journal of Economic Theory*, 80:292–331, 1998.

[35] J. Hu and M. P. Wellman. Experimental results on Q-learning for general-sum stochastic games. In *Proceedings of the 17th International Conference on Machine Learning*, pages 407–414, 2000.

[36] M. Bowling. Convergence problems of general-sum multiagent reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning*, pages 89–94, 2000.

[37] A. Greenwald and K. Hall. Correlated Q-learning. In *Proceedings of the 20th International Conference on Machine Learning*, pages 242–249, 2003.

[38] X. Wang and T. Sandholm. Reinforcement learning to play an optimal Nash equilibrium in team Markov games. In *Advances in Neural Information Processing Systems 15*, pages 1571–1578, 2003.

[39] M. L. Littman and P. Stone. Leading best-response strategies in repeated games. In *IJCAI workshop on Economic Agents, Models, and Mechanisms*, Seattle, WA, 2001.

[40] M. L. Littman and P. Stone. A polynomial-time Nash equilibrium algorithm for repeated games. *Decision Support Systems*, 39:55–66, 2005.

[41] J. R. Stimpson, M. A. Goodrich, and L. C. Walters. Satisficing and learning cooperation in the prisoner's dilemma. In *Proceedings of the 17th National Conference on Artificial Intelligence*, pages 535–544, 2001.

[42] G. Tesauro. Extending Q-learning to general adaptive multi-agent systems. In *NIPS*. MIT Press, 2004.

[43] D. de Farias and N. Megiddo. Exploration–exploitation tradeoffs for expert algorithms in reactive environments. In *Advances in Neural Information Processing Systems 17*, pages 409–416, 2004.

[44] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *20th Inter. Conf. on Machine Learning*, pages 228–236, 2003.

[45] Y. Chang and L. P. Kaelbling. Hedge learning: Regret-minimization with learning experts. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 121–128, 2005.

[46] E. M. de Cote, A. Lazaric, and M. Restelli. Learning to cooperate in multi-agent social dilemmas. In *Proceedings of the 5th International Conference on Autonomous Agents and Multiagent Systems*, pages 783–785, 2006.

[47] E. M. De Cote and M. L. Littman. A polynomial-time Nash equilibrium algorithm for repeated stochastic games. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, pages 419–426, 2008.

[48] S. Abdallah and V. Lesser. A multi-agent learning algorithm with non-linear dynamics. *Journal of Artificial Intelligence Research*, 33:521–549, 2008.

[49] G. Chasparis, J. Shamma, and A. Arapostathis. Aspiration learning in coordination games. In *Proceedings of the 49th IEEE Conference on Decision and Control*, pages 5756–5761, 2010.

[50] B. Bouzy and M. Metivier. Multi-agent learning experiments in repeated matrix games. In *Proceedings of the 27th International Conference on Machine Learning*, pages 119–126, 2010.

[51] M. Knobbout and G. A.W. Vreeswijk. Sequential targeted optimality as a new criterion for teaching and following in repeated games. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*, pages 517–524, 2011.

[52] S. Ganzfried and T. Sandholm. Game theory-based opponent modeling in large imperfect-information games. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*, pages 533–540, 2011.

[53] N. Cesa-Bianchi, O. Dekel, and O. Shamir. Online learning with switching costs and other adaptive adversaries. In *Advances in Neural Information Processing Systems 26*, pages 1160–1168, 2013.

[54] M. Elidrisi, N. Johnson, M. Gini, and J. W. Crandall. Fast adaptive learning in repeated stochastic games by game abstraction. In *AAMAS*, 2014.

[55] S. V. Albrecht and S. Ramamoorthy. On convergence and optimality of best-response learning with policy types in multiagent systems. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence (UAI-14)*, Quebec City, Canada, July 2014.

[56] Y. Gal, S. Kraus, M. Gelfan, H. Khashan, and E. Salmon. Negotiating with people across cultures using an adaptive agent. *ACM Transactions on Intelligent Systems and Technology*, 3(1), 2011.

[57] W. H. Press and F. J. Dyson. Iterated prisoner?s dilemma contains strategies that dominate any evolutionary opponent. *Proceedings of the National Academy of Sciences*, 109(26):10409–10413, 2012.

[58] A. Rapoport and M. Guyer. *A Taxonomy of 2 X 2 Games*. Bobbs-Merrill, 1967.

[59] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[60] D. Banerjee and S. Sen. Reaching pareto-optimality in prisoner's dilemma using conditional joint action learning. *Autonomous Agents and Multi-Agent Systems*, 15:91–108, 2007.

[61] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the 2nd European Conference on Computational Learning Theory*, pages 23–37, 1995.

[62] H. A. Simon. Rational choice and the structure of the environment. *Psychological Review*, 63(2):129–138, 1956.

[63] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks*, 1993.

[64] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[65] J J. O'Neill, B. Pleydell-Bouverie, D. Dupret, and J. Csicsvari. Play it again: reactivation of waking experience and memory. *Trends in Neuroscience*, 33:220–229, 2010.

[66] P. D. Taylor and L. Jonker. Evolutionarily stable strategies and game dynamics. *Mathematical Biosciences*, 40:145–156, 1978.

[67] B. Bouzy, M. Metivier, and D. Pellier. Hedging algorithms and repeated matrix games. In *ECML Workshop on Machine Learning and Data Mining in and Around Games*, Athens, Greece, 2011.

[68] J. R. Stimpson and M. A. Goodrich. Learning to cooperate in a social dilemma: A satisficing approach to bargaining. In *Proceedings of the 20th International Conference on Machine Learning*, pages 728–735, 2003.

[69] M. Smith and G. R Price. The logic of animal conflict. *Nature*, 246:15–18, 1973.

[70] L. Shapley. Some topics in two-person games. *Advances in Game Theory*, 1964.

[71] M. Oudah, V. Babushkin, T. Chenlinangjia, and J. W. Crandall. Learning to interact with a human partner. In *Proceedings of the 10th International Conference on Human-Robot Interaction*, 2015.

[72] M. Zinkevich, M. Johanson, M. H. Bowling, and C. Piccione. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems 20*, 2007.

[73] M. Johanson, N. Bard, M. Lanctot, R. Gibson, and M. Bowling. Efficient Nash equilibrium approximation through Monte Carlo counterfactual regret minimization. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pages 837–846, 2012.

[74] V. P. Crawford and J. Sobel. Strategic information transmission. *Econometrica*, 50 (6):1431–1451, 1982.

[75] R. J. Aumann and S. Hart. Long cheap talk. *Econometrica*, 71 (6):1619–1660, 2003.

[76] J. Farrell and M. Rabin. Cheap talk. *Journal of Economic Perspectives*, 10 (3):103–118, 1996.

[77] J. Y. Kim. Cheap talk and reputation in repeated pretrial negotiation. *RAND Journal of Economics*, 27 (4):787–802, 1996.

[78] O. Bonroy, A. Garapin, and D. Llerena. Repeated cheap talk, imperfect monitoring and punishment behavior: An experimental analysis. *Working Paper GAEL: 2011-2012*, 2012.

[79] A. L. Thomaz and C. Breazeal. Teachable robots: Understanding human teaching behavior to build more effective robot learners. *Artificial Intelligence*, 172:716–737, 2008.

[80] Robert Axelrod. *The Evolution of Cooperation*. Basic Books, 1984.

[81] A. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.

[82] A. Pentland. To signal is human. *American Scientist*, 98:204–211, 2010.