Modeling Joint Intention and Behavior through Reinforcement Learning

Gaby Ecanow, An Jimenez, and Janice Yang
9.66 Computational Cognitive Science Undergraduate Research Project
Internal TA Advisor: Setayesh Radkani

Abstract

In every interaction involving other individuals, humans decide how they want to act socially. One's choice of social intention impacts the choice of social intentions of those affected by the interaction, which in turn re-informs one's own internal model of their social position and intention of how to act. The goal of this project is to build on earlier research into modeling joint intention and behavior of cooperative versus competitive gameplay (Kleiman-Weiner et al., 2016; Zhang, 2018) to further understand intention, behavior, and social inference in light of a new strategic goal: hindering. Like the earlier research, we use reinforcement learning and Bayesian inference to build models operating at two levels of reasoning in stochastic game play. To validate the models, we ran three experiments that compared the model predictions with human predictions and simulated both low-level and high-level models playing goal-oriented games against itself. While the model outperforms humans in generating inferences for some scenarios, the model lacks the rich prior knowledge of social intention and gameplay of humans, preventing the current model from exceeding human performance across the board.

Keywords: cooperation; competition; reinforcement learning; joint intention; stochastic games; game theory

1. Introduction

Cooperation is central to the success of human societies and solving major global problems like climate change, ecological collapse, overpopulation, and a global pandemic; however, cooperation for certain problems may be individually costly, so it is often a challenge to get large swaths of the population to partake in it. In order to eventually reach this larger goal of understanding cooperation on a global scale, it is important to understand and characterize how people compete and cooperate on the individual level.

Inspired by existing research on modeling joint intention and behavior, our team studied the high-level strategic goals of cooperation, competition, and hindering through reinforcement learning (Kleiman-Weiner et al., 2016; Zhang, 2018). By re-creating a hierarchical model of social agency, we developed both abstract notions of cooperative and competitive behaviors and expanded on the current research by introducing a second environment, a third and fourth low-level agent, and an attribution experiment in which we compare the model's social goal inferences against human interpretations. The conclusions drawn from this project give new insight into the computational mechanisms that perceive and drive cooperative, competitive, and hindering behaviors. In doing so, we also expand our knowledge on creating machines that can cooperate better with their human counterparts.

2. Game

Before creating the computational models, we first decided on the central game used for this study in which players have the opportunity to execute one of the four low-level roles as a cooperative, competitive, hindering, or random agent. In the following section, we describe the games and game rules used to test our high- and low-level models.

Game Rules

In order to explore individual level cooperation and competition, we extended the stochastic goal-oriented player game described in references [1] and [2]. These games are inspired by classical game theory scenarios such as The Prisoner's Dilemma, in which two players must weigh their utility based on playing in coordination with their opponent or in opposition to their opponent.

Prisoner's Dilemma		B	
		Cooperative	**Competitive**
A	**Cooperative**	7, 7	0, 10
	Competitive	10, 0	0, 0

Table 1: Prisoner's Dilemma Payoff

In the case above, if both players play cooperatively, both players win points; however, if one player plays competitively, they win more points than they would playing cooperatively but only if their opponent does not also play competitively. In the case that both players play competitively, both players lose the game.

In this project, we explore paradigms of game play using low-level and high-level agents that model game players. For our experimentation, we use two two-player board games, in which players can take one of the following actions at each time step: UP, DOWN, LEFT, RIGHT, or WAIT. Players gain 10 points for entering a square of their same color (their 'goal') but lose a point for every move made not resulting in hitting their goal. Players also lose a point for engaging in illegal moves, such as trying to enter a square where another player is currently residing, or moving outside the game board. In summary, any move not resulting in a player entering his goal results in a lost point.

A game ends when either one or both of the players enter their goal. If both players enter at the same time, the game ends in a cooperative outcome, whereas if only one player enters their goal on the final timestep, the outcome is considered competitive. Although the game can either end in a competitive or cooperative outcome, the strategies employed by the players may differ from the game outcome. Finally, to mitigate any color bias, the boards are symmetric and both players view themselves as the "Blue" player and their opponent as the "Yellow" player.

Each timestep of our implementation of the game proceeds in a state diagram, as shown below where the green arrows

represent an affirmative response to the actions within the circle:

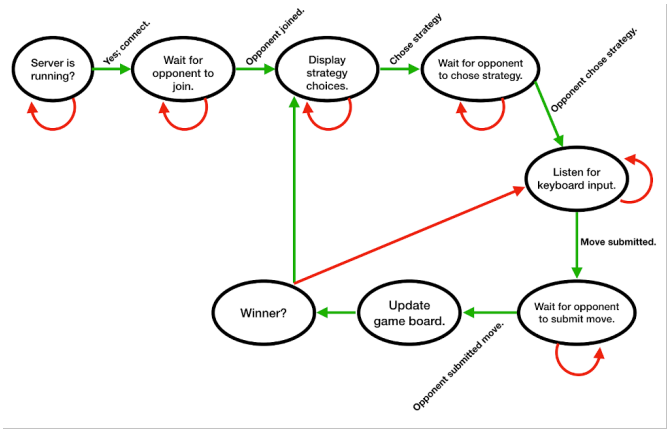


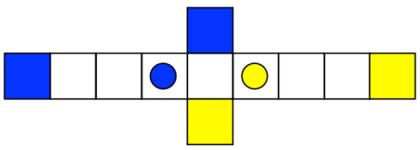
Figure 1: State diagram of game implementation

Before each round, players selected which strategy they plan to use for the coming round. Multiple games can be played in succession between the same two players, and players can choose new strategies at the start of each game. Players do not know which strategies their opponent has chosen, nor what move they have made before making a move themselves. If players attempt to enter the same square on a single time step, a coin flip determines which player "wins" the square. If a player attempts to enter a square that will be occupied by their opponent on the next timestep (i.e. their opponent submitted an illegal or WAIT move), the move fails and the player remains in place. Finally, if two adjacent players attempt to both enter the square of their opponent (i.e. switch squares), both moves fail and the players remain in place.

Game Boards

For this project, we used two game boards in our experiments: Grid 1 from the reference paper [2] and Grid 2 as our own extension of the current research.

As specified above, both game boards are symmetrical, and both players view themselves as "Blue" and their opponent as "Yellow".

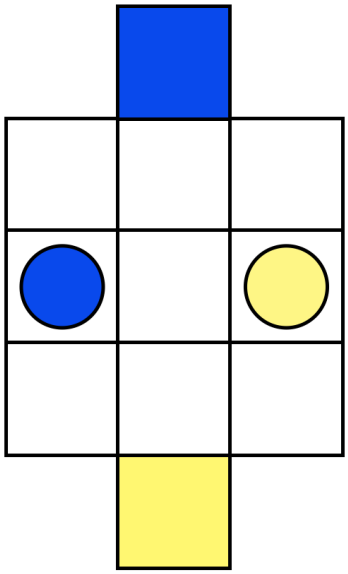


Blue, Yellow	Yellow cooperates	Yellow competes
Blue cooperates	$7*, 7*$	$-2, 8$
Blue competes	$8, -2$	$2, 2$

*assuming no miscoordination

Figure 2: Grid 1 Representation (taken from Reference [2])

Grid 1 This first game board allows us to distinguish between collaborative and competitive strategies. A collaborative player would either move outwards towards the farther goal, or would go in the center but wait; even though both of these sequences is longer and would decrease their final overall utility, they are the only way to ensure that both players get a chance to reach their goal states at the same time; however, if either player is competitive, they would try to go towards the middle, the shorter path towards the goal, even though only one player will ultimately be able to reach it first.



Blue, Yellow	Yellow cooperates	Yellow competes	Yellow hinders
Blue cooperates	7*,7*	7, 7	-t, t
Blue competes	7, 7	7, 7	-t, t
Blue hinders	t, -t	t, -t	t, t

* assuming no miscommunication
t = # timesteps

Figure 3: Grid 2 Representation

Grid 2 This board would be able to distinguish between the competitive and hindering behaviors because a competitive player would head straight to their own goal immediately to increase their own utility; however, a hindering player's utility is based on the other player's negative utility. Thus, the hindering player would be incentivized to head in the other direction to block the other player in front of their goal.

3. Computational Framework

Our model is divided into two categories of models: the low-level and the high-level model. In the following section, we describe the implementation of both categories of models, including the algorithms used.

Low-Level Models

The low-level models generate policies that determine the likelihood of an action given a state: $\pi(s|a)$ where a is an action from the set $\{RIGHT, LEFT, UP, DOWN, WAIT\}$ and s is the state, represented as a tuple $(blue\_cell, yellow\_cell)$ where $blue\_cell$ is the current location of the blue player on the grid and $yellow\_cell$ is the current location of the yellow player on the grid. In this project, we programmed four separate low-level models: CoopAgent, CompeteAgent, HinderAgent, and RandomAgent.

CoopAgent CoopAgent is the low-level model representing cooperative behavior between the two players. CoopAgent uses Q-value iteration over the joint action space to determine the optimal policy given a state. More specifically, CoopAgent takes in as parameters an instance of the grid class grid, a float β for calculating the softmax, and a float w representing the weight of the blue player's utility. The algorithm used to train the model and compute an optimal MDP policy is as follows:

1. For each possible state and for all possible action pairs given a state, calculate the utility and transition probability of the next state given the current state and the action pair.

2. Given the utility and transition probability, calculate the new Q value for the current state and action pairs using a discount factor γ of 0.9 from the referenced papers (Equation 1).

$$Q^G(s,a_1,a_2) = \sum_{s'} P(s'|s,a_1,a_2)(U^G(s',s,a_1,a_2) + \gamma * max_{a'_1,a'_2} Q^G(s',a'_1,a'_2)) \quad (1)$$

3. If the difference between the new Q value and the old Q value is greater than epsilon, terminate training. The epsilon value is set to 0.1 for CoopAgent which is sufficient for Q value convergence during training.

4. After all Q values have been calculated and updated for a given state and all possible action pairs, update the policy for the given state by computing the softmax given β and the Q values (Equation 2). The value β determines how much noise contributes to the transition probabilities. A higher β means there is a higher likelihood of selecting the optimal action, while a lower β means there is more noise. The joint policy is marginalized over each individual player to get each player's cooperative policy separately.

$$P(a_1,a_2|s) = \pi^G(s,a_1,a_2) \propto e^{\beta Q^G(s,a_1,a_2)} \quad (2)$$

After training, one can obtain the optimal action for a player by calling the *get_action* function which takes the current state and the player and samples from the list of possible actions given their likelihood distributions generated from training and returns an action.

CompAgent CompAgent is the low-level model representing competitive behavior, where one player attempts to maximize their own utility assuming that the other player is trying to do the same. We utilize the same K-level value iteration method used in the referenced papers, where the Level-1 Agent responds to the Level-0 agent, and the Level-0 agent is simply the base case of a player maximizing their utility when there are no other players on the board. The algorithm used to train this model is as follows:

1. Train the Level-0 agent for either player i_0 = Blue or i_0 = Yellow, using a similar method of Q value iteration described above for the CoopAgent. We can then calculate the best actions for player i_0 at each given state, taken as sampling from a softmax distribution given the Q values and selected β.

$$P(a_i|s,k=0) = \pi_i^0(s) \propto e^{\beta Q_i^0(s,a_i)} \quad (3)$$

$$Q_i^0(s,a_i) = \sum_{s'} P(s'|s,a_i)(U_i(s,a_i,s') + \gamma * max_{a'_i} Q_i^0(s',a'_i)) \quad (4)$$

2. Train the Level-1 agent for the other player (i_1 = Yellow if i_0 = Blue, and vice versa), also with Q-value iteration, where the dynamics of the state transitions incorporate the Level-0 Agent's actions as a deterministic component as calculated in the previous step. In other words, we marginalize the Level-1 Agent's transition probabilities given knowledge of the Level-0 Agent, represented by this equation (where $-i$ is shorthand to refer to the "other" player):

$$P(s'|s,a_i) = \sum_{a_{-i}} P(s'|s,a_i,a_{-i}) P(a_{-i}|s,k=k-1) \quad (5)$$

3. The optimal policy for the competitive player is then calculated similarly to Level-0, by a sampling from a softmax distribution, given the Q values and the β of the trained Level-1 agent.

$$P(a_i|s,k) = \pi_i^k(s) \propto e^{\beta Q_i^k(s,a_i)} \quad (6)$$

$$Q_i^k(s,a_i) = \sum_{s'} P(s'|s,a_i)(U_i(s,a_i,s') + \gamma * max_{a'_i} Q_i^k(s',a'_i)) \quad (7)$$

HinderAgent HinderAgent is the low-level model representing punishing behavior, where the hindering player gets additional reward when their opponent performs poorly. The training process for the HinderAgent is very similar to that described above for the CompAgent, with the main difference being that instead of measuring each player's utility solely on the points they get or lose stemming from their own actions, there is now an additional parameter h that determines how much they have a hindering characteristic, where the larger the h, the more they benefit from the negative utility of their opponent. The Level 1 hindering player's utility is calculated to be $U = U1 - h * U2$.

RandomAgent RandomAgent is the low-level model representing random actions taken for a given state. The RandomAgent takes in as a parameter an instance of the grid class grid and generates policies based on the equal probability of choosing an action from a set of possible next actions. The algorithm for RandomAgent determines, for each possible state, the set of possible next actions the player can take and assigns a probability of $\frac{1}{n}$ to each action in the set where n is the number of possible next actions given the state. Impossible actions will always be assigned a probability of 0. Calling the *get_action* function returns a random action by sampling the set of possible actions against a uniform distribution.

Low-Level Model Visualizations

In order to better understand the optimal actions for each low-level agent described above, we ran each trained low-level model on each grid in order to visualize their π dictionary likely moves. The heat maps were generated by running each model through 10,000 games to obtain priors, and summing the likelihood of the Blue player moving from the current state to the next state with the actions chosen (i.e. The blue pi dictionary value) times the prior probability of being in the current state. The following two figures display the heat maps for each of the four agents on Grid 1 and Grid 2 respectively. Higher summed posterior values are indicated by a darker color.

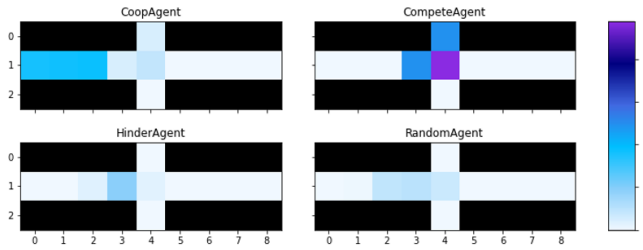


Figure 4: Heat Map Distribution of Low Level Agents on Grid 1

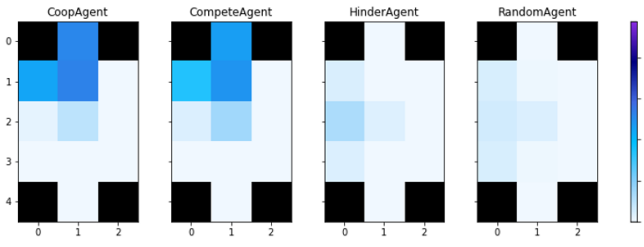


Figure 5: Heat Map Distribution of Low Level Agents on Grid 2

Evidently, on Grid 1, the CoopAgent is most likely to move towards the outer goal, which aligns with the cooperative strategy. Meanwhile, the CompeteAgent is most likely to try for the closer goal at the expense of their opponent, the HinderAgent tends towards the middle of the board, and the RandomAgent is generally anywhere on the left starting side.

On Grid 2, the Coop and Compete agents are likely to move towards their goal with about equal probabilities which makes sense, given these heat maps take into account all possible locations and actions of the opponent. While the CoopAgent and CompeteAgent may differ in how long they wait to enter their goal, their relative paths may look the same. On the other hand, the HinderAgent and RandomAgent equally move about the left side of the board, with the HinderAgent spending more time in its starting position. This may be due to the actions of the other agent, since the HinderAgent may be moving with a strategy in response to the opponent, whereas the RandomAgent is most likely to randomly explore it's nearby cells. Importantly, HinderAgent and RandomAgent games may also last longer than CoopAgent and CompeteAgent games, which might cause their posterior values to drop significantly over the course of a single game given that, well into a game, there's a prior probability of the agent being almost anywhere in the board. We dive into these postulations during our experiments, which we detail later in the paper.

High-Level Models

The goal of the high-level model is twofold: first, to observe sequences of moves made by an opponent and infer their strategy, and second, to choose the best next move in playing a 'tit for tat' (TFT) strategy themselves (i.e. mirroring the strategy of their opponent). In more general terms, the high-level model works as a game player would, attempting to infer the strategy of its opponent and modify its own strategy accordingly. To accomplish these goals, the high-level model first assumes a prior over the possible low level strategies employed by their opponent and uses a Hidden Markov Model (HMM) with Bayesian inference to update its priors over the strategies as more data is observed (see Figure 6).

Why a 'tit for tat' strategy? Reference paper [2] describes how Robert Axelrod, author of The Evolution of Cooperation, held a tournament to identify the best strategy when playing games following the style of the Iterative Prisoner's Dilemma:

> "A strategy called tit for tat (TFT), submitted by Anatol Rapoport [14], won the tournament despite being the simplest algorithm. TFT started in the cooperative mode, and on subsequent rounds it switched to whichever mode the other player used on the previous round. TFT was also able to perform well with itself and generally avoid being exploited, indicating that this type of reciprocity is an important social norm."

The paper also describes a second well-performing strategy, the Pavlovian strategy, which, according to the reference, out-performs tit-for-tat due to its "account of noise" and ability to "correct for mistake":

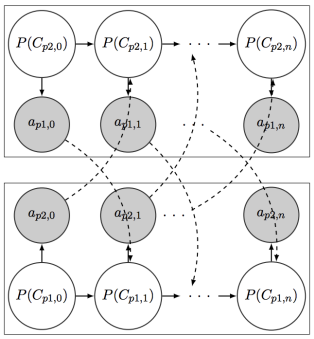


Figure 2-5: The HMM graphical model representing the CCAgent's action observation and belief update in dual-agent interactions. The top nodes represent player $p1$, and the bottom nodes represent player $p2$. On each step i from 0 to n, each player p has a belief $P(C_{p\_other,i})$ of how cooperative the other player is. Using this belief, they generate a policy π_p from which they select their next action $a_{p,i}$. Then, each player observes the other player's a_i and updates their own belief state $C_{p\_other,i+1}$. The shaded nodes represent the observations and the dotted lines represent the belief updates.

Figure 6: A graphical representation and explanation of the HMM taken from reference [2], p.g. 33.

> Nowalk and Sigmund [12] introduced a new strategy called Pavlov, or "win-stay, lose-shift." If a player won a round, they would stay with the same strategy, and if they lost a round, they would shift to the opposite strategy. "Win-stay lose-shift" echoes a Pavlovian effect of responding positively to a reward and negatively otherwise, which is behavior found in many animals [5].

Due to the fact that the reference's High Level Agent follows a variation of the 'tit-for-tat' strategy, we too built our high level models to play a 'tit-for-tat' strategy.

We coded three variations of the high level model: the GeneralAgent, the SuperGeneralAgent, and the ForgetfulAgent. In order to understand the evolution of the high level models, we first present a description of a special case of the GeneralAgent, the CCAgent.

CCAgent The CCAgent, Coop-Compete Agent, assumes their opponent is only playing either a cooperative or competitive strategy. As such, the CCAgent assumes a prior $P(C)$ such that $0 \leq P(C) \leq 1$ which describes their belief degree that their opponent is cooperative. Due to conservation of belief, the prior for the competitive strategy is always $P(C') = 1 - P(C)$. In our model, we assume an initial $P(C) = 0.5$.

We define two hyperparameters: decay d, $0 \leq d \leq 1$ and $0 \leq b \leq 0.5$. A decay closer to 0 will more highly weigh its prior, whereas a decay closer to 1 will more likely switch between strategies. The hyperparameter b bounds the prior for the CoopAgent in the next time step, thereby ensuring that the agent always has some probability of switching.

The model can iteratively observe game data using the observe function. Given a new game state s_i and action pair (a_1, a_2), the CCAgent will update its prior in the following way.

First, the agent weighs its initial priors based on the decay, where

$$T_C = (1-d) * P(C) + d * P(C') \tag{8}$$

$$T'_C = (1-d) * P(C') + d * P(C) \tag{9}$$

Next, the agent uses the trained low level models to determine the likelihood that the observed data occurred given the low level model, $P(s, a_1, a_2|C)$, using the low level *pi* dictionary obtained from training. The model's goal is to update and determine the probability of their opponent using a cooperative strategy given the next state and actions.

$$P(s, a_1, a_2|C) = \text{CoopAgent}.\pi[s][(a_1, a_2)] \tag{10}$$

$$P(s, a_1, a_2|C') = \text{CompeteAgent}.\pi[s][(a_1, a_2)] \tag{11}$$

We can use Bayes' Theorem to determine the posterior:

$$P(C|s, a_1, a_2) = P(s, a_1, a_2|C) * T_C \tag{12}$$

$$P(C'|s, a_1, a_2) = P(s, a_1, a_2|C') * T'_C \tag{13}$$

Finally, the CCAgent reweighs its prior over the cooperative strategy using the hyperparameter b:

$$b_{\text{weight}} = (1 - 2 * b) \tag{14}$$

$$C_{\text{weighted}} = (T_C * b_{\text{weight}} + b) * P(s, a_1, a_2|C) \tag{15}$$

$$C'_{\text{weighted}} = (T'_C * b_{\text{weight}} + b) * P(s, a_1, a_2|C') \tag{16}$$

$$P(C^{i+1}) = C_{\text{weighted}} + C'_{\text{weighted}} \tag{17}$$

Using the low-level model with the higher prior, the CCAgent returns a next action from the low-level model's `get_action` function.

In creating the CCAgent, we assume players are optimal, meaning that although they may not be perfect, if human players assume an opponent is playing a strategy with confidence over another strategy, they will always go with the next move suggested by the strategy with the highest confidence. The uncertainty in game play comes from low-level model uncertainty, including the uncertainty over what the suggested next action to take is, and the uncertainty over what state and actions pairs are most characteristic of which strategy.

GeneralAgent The GeneralAgent is a generalized version of the CCAgent that takes in any two low level strategies and uses the HMM described previously to observe new data and update its prior accordingly.

An instance of the GeneralAgent will take in any two strategies, agent1 and agent2, and will observe in the same way as the CCAgent. Importantly, any one instance of the GeneralAgent will always only consider the two agents it is initialized with. The benefit of the GeneralAgent is being able to focus on any two opposing strategies during game play with an opponent, but the GeneralAgent can be reset to examine two new agents in between opponent swaps or when starting game play on a new board.

The GeneralAgent has a `next_move` function that plays a 'tit-for-tat' strategy, which chooses the move suggested by the low-level with the highest prior probability (i.e. the best guess for which strategy its opponent is using).

SuperGeneralAgent The SuperGeneralAgent is an extension of the GeneralAgent that attempts to simultaneously take into account all n low level model strategies. To do this, the SuperGeneralAgent defines and updates a prior vector, $P(\vec{C})$ with $n-1$ degrees of freedom in its HMM, such that the sum of all priors equals 1. The SuperGeneralAgent only takes in two hyperparameters: d, which acts the same in function to the decay parameter of the other high level models. We found during hyperparameter tuning that a d factor close to zero resulted in the most optimal model predictions. *diff* bounds how much a prior can change in any one observed step. At each observation of new data, the (n,n) matrix T is calculated:

T = matrix of d's with (1-d) along the diagonal multiplied by transposed the prior vector

$$T = \begin{bmatrix} (1-d) & d & d & ... & d \\ d & (1-d) & d & ... & d \\ d & d & (1-d) & ... & d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d & d & d & ... & (1-d) \end{bmatrix} \quad (18)$$

Each prior $P(C_i)_T$ is calculated by multiplying the i^{th} row in T by the current transposed $P(\vec{C})$, effectively multiplying each $P(C_i)$ by $(1-d)$ and adding to it $d*(1-P(C_i))$, given that all the priors sum to 1:

$$P(C_i)_T = T^i * P(\vec{C})^T \quad \forall i \in \{1,...,n\} \quad (19)$$

The likelihood $P(s,a_1,a_2|C_i)$ is calculated using the π dictionaries of the low level models:

$$P(s,a_1,a_2|C_i) = C_i.\pi[s][(a_1,a_2)] \quad \forall i \in \{1,...,n\} \quad (20)$$

Finally, the posterior vector is:

$$P(C_i|s,a_1,a_2) = P(s,a_1,a_2|C_i) \times P(C_i)_T \quad \forall i \in \{1,...,n\} \quad (21)$$

For any posterior values that increase on a single timestep by more than *diff* is subsequently decremented by *diff*, and any posterior value that decreases on a single timestep by more than *diff* is incremented by *diff*. The *diff* parameter was included to ensure that the model does not make large jumps in posterior probability from one time step to the next which can dramatically affect the model predictions by placing a lot of weight on one particular move. As such, the posterior probability will always change by the minimum of *diff* and the calculated new posterior.

Finally, the prior vector $P(\vec{C})$ is updated to reflect the *normalized* posteriors.

Like the GeneralAgent, the SuperGeneralAgent has a `next_move` function that will suggest a next move based on the low level strategy with the highest prior. If multiple low level agents have equal priors, the SuperGeneralAgent will randomly choose from one of them to suggest a next move. The SuperGeneralAgent is an exploratory new high-level model we created and is not from either of the two reference papers. It is inspired by the addition of new low-level models, the HinderAgent and the RandomAgent.

ForgetfulAgent The ForgetfulAgent is a high-level model that makes inferences with no regard to earlier observations. Unlike the other high level models, the ForgetfulAgent does not have an internal HMM, nor does it have initial priors over what strategies it thinks the other player is using. The ForgetfulAgent suggests a next move based solely on the low level model with the highest likelihoods of the last observed state and action pair. We use the ForgetfulAgent as a comparison metric for the two other high level models, the GeneralAgent and the SuperGeneralAgent.

4. Experimental Design

Using the game and models described in the above sections, our team completed three different experiments to gain insight around the accuracy of model predictions versus human predictions. Experiment 1 compares the differences in model predictions and human predictions for agent type given human-generated game data. Experiment 2 compares the differences in model prediction distributions and human prediction distributions across four characteristic, human-generated state and action sequences, one for each type of agent. Experiment 3 simulates the model playing a game against itself for each combination of Low Level agents against each other and against the High Level Agent.

To collect human-generated game data, we recruited a total of four subjects (n = 4) who played a total of 48 games collectively. Figure IV.A shows the distribution of the number of games played for each type of grid (Grid 1 and Grid 2) and for each type of agent (CoopAgent, CompeteAgent, HinderAgent, and RandomAgent). Given the restrictions from the pandemic and the timeline for this project, all subjects were recruited from our close circles of family and friends. Subjects were required to be on the same WiFi network in order to play the game; however, participants did not communicate with one another during each of the trials. Additionally, the human attributions for Experiments 1 and 2 were generated by our own team members (n = 3) answering questions about the human-generated game data. The attribution portion of these experiments were conducted individually as to not influence one another's predictions. In the following section, we describe in detail the experimental design for each experiment, display the results, and highlight any key findings.

For more information on how the human-generated game data was collected, see the Appendix.

Hyperparameter Tuning

Before running the experiments, it was necessary to tune the hyperparameters of the low-level models in order to yield predictions that were in line with our expectations for the model output. To tune the hyperparameters, we generated four characteristic state and action pair sequences, one for each low-level agent, and ran each low-level agent on its respective sequence to tune the hyperparameters such that the model would output the expected prediction with high likelihood. The human-generated sequences are (for player Blue):

	Grid 1
Cooperative	LEFT, LEFT, LEFT
Competitive	RIGHT, UP
Hindering	RIGHT, WAIT, WAIT
Random	LEFT, LEFT, RIGHT, LEFT

	Grid 2
Cooperative	UP, RIGHT, WAIT, WAIT, UP
Competitive	UP, RIGHT, UP
Hindering	DOWN, RIGHT, WAIT, WAIT
Random	DOWN, UP, WAIT, RIGHT, LEFT, DOWN, WAIT, UP, RIGHT

These characteristic sequences were chosen because they distinctly represent the actions that their respective low-level agent would take, according to human definitions of these behaviors. The final hyperparameters for each low-level model was used in each of the following experiments, and the above sequences are also the same ones used for Experiment 2.

Experiment 1

The goal of Experiment 1 is to compare the SuperGeneralAgent model's predictions against the ground truth and compare the model predictions with human predictions for the human-generated game data to determine both whether the model or human surpasses the other in accuracy and across which agents and grids.

Model Predictions To generate the model predictions on the same game data, we used the SuperGeneralAgent model trained on all four agent types with a $diff$ parameter of 0.8. For each game, the model predicted the agent type that had the highest posterior probability at the end of the game. The accuracy score for the model was generated using the following equation: $\frac{k}{n}$ where k is the number of predictions that matched the ground truth out of n total trials.

Human Predictions Each of our team members acquired a text file of human-generated game data which included the game ID, the grid number, the actions taken by each player, and the state transitions for each game played. To improve efficiency and accuracy of the human predictions, a short script created a visualization from the text file that reenacted the actions and state transitions for each game directly onto the grids.

Independently, each team member observed games played, predicted the intended agent type for each player, and recorded the predictions in a spreadsheet. The final human prediction for each game was determined by majority rule of the three individual human predictions from the members of our team. If there is no majority in the predictions, the trial was marked as inconclusive and did not contribute to the overall accuracy score. The accuracy score for the human predictions was calculated in a similar way as the model accuracy score, $\frac{k}{n}$, where k is the number of final predictions that matched the ground truth out of n total trials.

Experiment 2

The purpose of this experiment was to compare human and model posterior probabilities on characteristic sequences for Cooperate, Compete, Hinder, and Random, on both grids. These characteristic sequences describe steps that one single player, the Blue player, can take on each grid. These sequences are characteristic because humans have identified by consensus as belonging predominantly to one player type. The sequences are listed in section Hyperparameter Tuning above, and the same sequences are used here to gain a deeper understanding of model performance, even after parameter optimization. We compared the posterior probabilities for all eight sequences between human posteriors and model posteriors to see whether there were any discrepancies. Model predictions were made using the SGA and equal priors of low-level agents to begin, with sequences fed in as actions the Blue player took (and Yellow always choosing WAIT). For models, we took the generated posterior probabilities from the last timestep of the sequence; for humans, we asked three participants (us) to directly generate posteriors for each sequence, then averaged the samples.

Experiment 3

For the third experiment, we ran a series of tests pitting models against each other, and we scored the high-level model on its ability to converge on the correct guess for its opponent's strategy. We ran the third experiment to validate whether or not the internal HMM performed better than the ForgetfulModel.

Low-level vs. Low-level We first had two low-level models play against each other and against themselves. LowLevelAgent1 (LLAgent1) and LLAgent2 were each drawn from the set of {CoopAgent, CompeteAgent, HinderAgent, and RandomAgent}. Each of the 16 low level model pairs played 1000 games against each other: 500 with LLAgent1 as blue and LLAgent2 as yellow, and 500 as LLAgent1 as yellow and LLAgent2 as blue. For each of the pairings, a high-level SuperGeneralAgent (SGA) observed each game, and we recorded the SGA's confidence in the correct low level models in play over the course of each game. If, after 50 moves, a game had not ended, we ended the game and marked it as inconclusive. In order to run statistics over the 500 games, if a game ended before the 50th move, we padded the confidence array with the last recorded confidence in order to make it of length 50. So, for each pairing, we compiled a (500 x 50)

confidence matrix which we collapsed into a single, averaged confidence over time.

High-level vs. Low-level For the second model versus model experiment, we had a high-level agent play against each of the low-level models. The agent played against each respective low level model 30 times, in order to simulate how long a sequence of games may last between two human players. During a single run of a game, the agent tried to guess the opponent's strategy and played various different strategies, including a forgetful 'tit-for-tat' strategy, an HMM 'tit-for-tat' strategy. The high-level agent was always the blue player. Again, games were cut off after 50 moves and marked 'inconclusive' if unfinished.

5. Results

The following section displays the results of the each experiment and describes the key observations from the data.

Experiment 1

The first graph in Figure 7 displays the accuracy score of the SuperGeneralAgent model's predictions and the human predictions against the subject-chosen ground truth across the four agents. The model surpasses human predictions for the Compete (+10.7%) and Hinder (+5%) agents, but falls far below the human accuracy scores for the Cooperate (-14.3%) and Random (-40%) agents. On average, the human predictions were slightly more accurate (71.9%) than the model's (63.5%). The second graph in Figure 7 displays the accuracy score of the model and human predictions against the ground truth across grid type. While the human and model accuracy scores are almost equal for Grid 1 (+1.7% model), the human accuracy score is much higher (+23.7%) than that of the model for Grid 2.

Figure 8 shows the accuracy scores for model and human predictions for Grid 1 and Grid 2 respectively across all four agents. For Grid 1, the model and human predictions were relatively similar in accuracy score, with the greatest difference in score for the Compete agent (+11.8% model) followed by the Random agent (+9.1% human). For Grid 2, the model and human predictions were more disparate, particularly across the Cooperate (+27.3% human) and Random (+77.8% human) agents. Additionally, there was no difference in accuracy score for the Hinder agent and only a small difference in score for the Compete agent (+9.1% model).

Figure 9 shows the match score of the model and human predictions by grid and agent type. The match score is calculated by comparing the model predictions against the human predictions. Mathematically, the match score is p/n where p is the number of predictions where the model and human make the same prediction and n is the total number of predictions. On average, the match scores for Grid 2 are significantly lower (-26.7%) than that of Grid 1. For both Grid 1 and Grid 2, the match scores for the Random agent were the lowest across all agents at 72.3% and 11.1% respectively. The highest match score for Grid 1 was for the Compete agent at

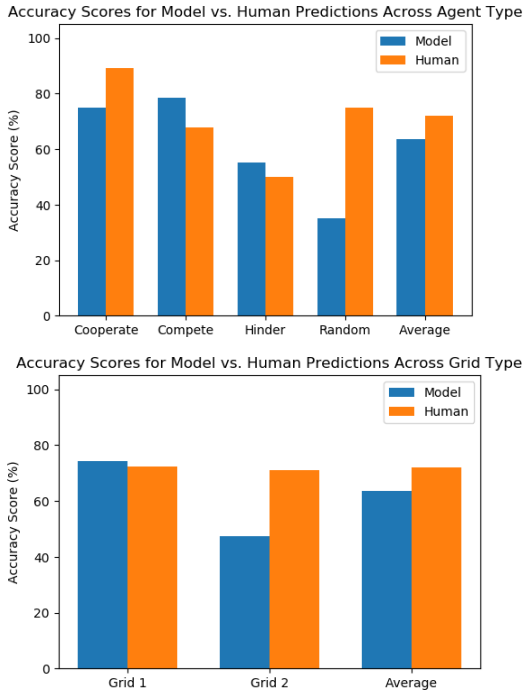


Figure 7: Accuracy scores for model and human predictions across agent type and grid type.

94.1% and for Grid 2, the Cooperate agent at 81.8%.

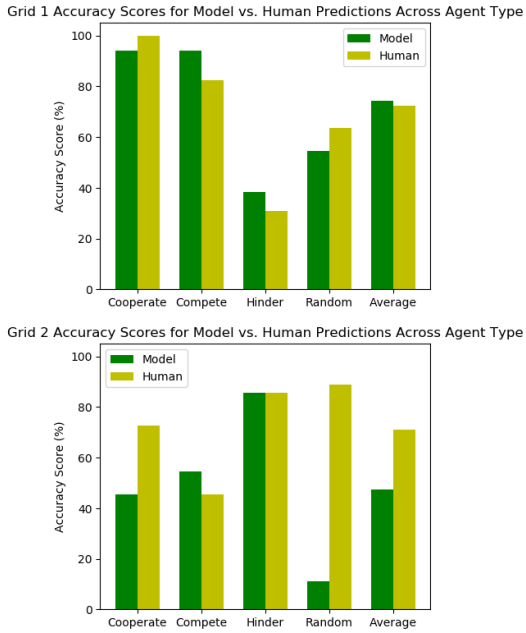


Figure 8: Accuracy scores for model and human predictions across each of the four low-level agents for Grids 1 and 2.

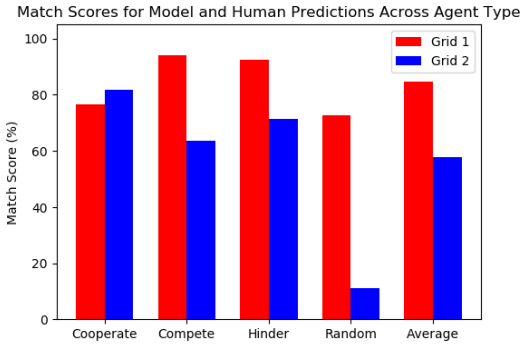
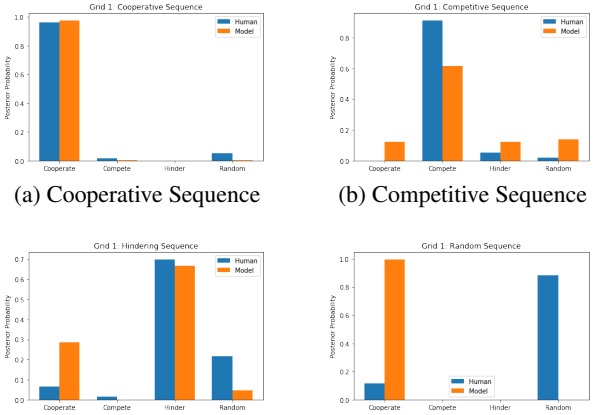


Figure 9: Match scores for model and human predictions across each of the four low-level agents for Grids 1 and 2.

Experiment 2

Overall, the model had quite similar predictions compared to the humans for cooperative, competitive, and hindering sequences for Grid 1. It was just as good at predicting cooperative and hindering sequences for Grid 2; however, it could not identify the Competitive sequence in Grid 2, nor the Random sequences in either grid.

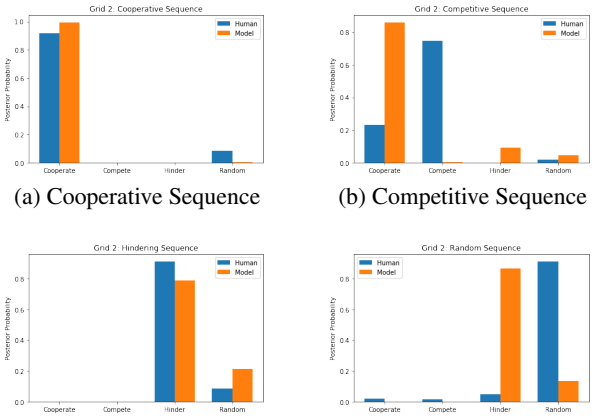
For both Cooperate sequences, the model was able to identify them with very high posteriors: 0.977 and 0.996 for Grid 1 and 2 respectively, similar to humans at 0.963 and 0.917. The model was also able to identify the Competitive and Hindering sequences on Grid 1, at 0.615 and 0.667, which was a little lower than human estimations of 0.913 and 0.7. However, the model was completely off the mark when identifying the Competitive sequence of Grid 2, attributing it to a Cooperative sequence with 0.861 confidence; likewise, it failed to identify both Random sequences, attributing them to Cooperate (0.996) and Hinder (0.865) respectively. More on this will be elaborated in the discussion section.



(a) Cooperative Sequence (b) Competitive Sequence

(c) Hindering Sequence (d) Random Sequence

Figure 10: Grid 1 Characteristic Sequence Attribution (Posteriors from Left to Right: Cooperate, Compete, Hinder, Random)



(a) Cooperative Sequence (b) Competitive Sequence

(c) Hindering Sequence (d) Random Sequence

Figure 11: Grid 2 Characteristic Sequence Attribution (Posteriors from Left to Right: Cooperate, Compete, Hinder, Random)

Experiment 3

Experiment 3.1: Low-Level v. Low-Level Recall, for Experiment 3.1, we had each low-level model play every other low-level model 1000 times, and averaged the SuperGeneralAgent's confidence. (i.e. prior) in the correct low-level model. players over time.

Many of the confidence values range from above 0.99 to 1.0, and the graphs are zoomed in to show the minor differences between SGA's confidence in the Blue versus Yellow player. For example, it seems that in the CoopAgent versus CompeteAgent games, the SGA does a slightly better job at identifying the CompeteAgent (confidence 0.9999 versus 0.9990) whether the CompeteAgent is playing as Blue or as Yellow.

The only matchups for which the SGA performs objectively poorly is CompeteAgent vs. HinderAgent (always having a correct high confidence in the HinderAgent but a low [0.0] confidence in the CompeteAgent), as well as any matchup with a RandomAgent, for which the SGA does a good job of identifying the RandomAgent's opponent (whether Blue or Yellow), but a poor job at identifying the RandomAgent (with a steady confidence of 0.0 throughout the course of every game). These matchup statistics are true for both grids, which provides evidence that the SGA itself is lacking in these areas.

The biggest difference between the two grids is that in the second grid, the SGA does a poor job of identifying both HinderAgents in the Hinder v. Hinder game, and only does a poor job at identifying the CompeteAgent in the Hinder vs. Compete game when the CompeteAgent is playing as yellow. This is possibly because, on the second grid, the competitive sequence may be mistaken for a cooperative sequence, and because the Hinder sequence may seem random when both agents are doing so. These theories are an interesting avenue for future study.

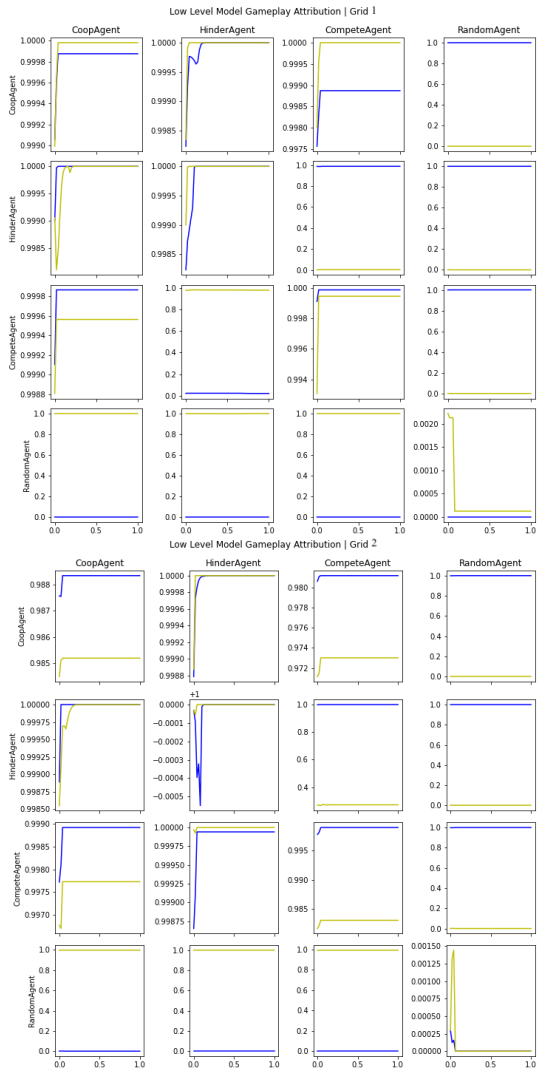


Figure 12: Confidence over time in the correct low-level blue and yellow player strategy. Top: Grid 1, Bottom: Grid 2

For comparison, we ran both the SGA and the ForgetfulModel simultaneously as the high level inference agents over the same distribution of played games. The ForgetfulModel's agent1 (Blue) confidence scores was plotted as a green line, while the ForgetfulModel's agent2 (Yellow) confidence scores was plotted as a red line and yielded the following confidence graphs.

In Figure 13, we see that the confidence values now range from far lower values (between 0.2-0.5) to 1.0, demonstrating that SGA far outperforms the ForgetfulAgent in general, both in raw confidence values, as well as in how quickly the agent converges on the correct low level model. It's possible that, as time goes on, the strength in the confidence values levels off due to the model's tendency to converge on the same or similar prior per agent per timestep in each played game, or due to the quick nature of the game play (in some cases, only lasting 2-4 moves per game). The only matches in which the ForgetfulModel outperforms the SGA is in identifying the RandomAgent (on average, confidence 0.2 versus 0.0), and in identifying the CompeteAgent in Hinder versus Compete games. These phenomena suggest that perhaps the SGA is underfitting against the RandomAgent.

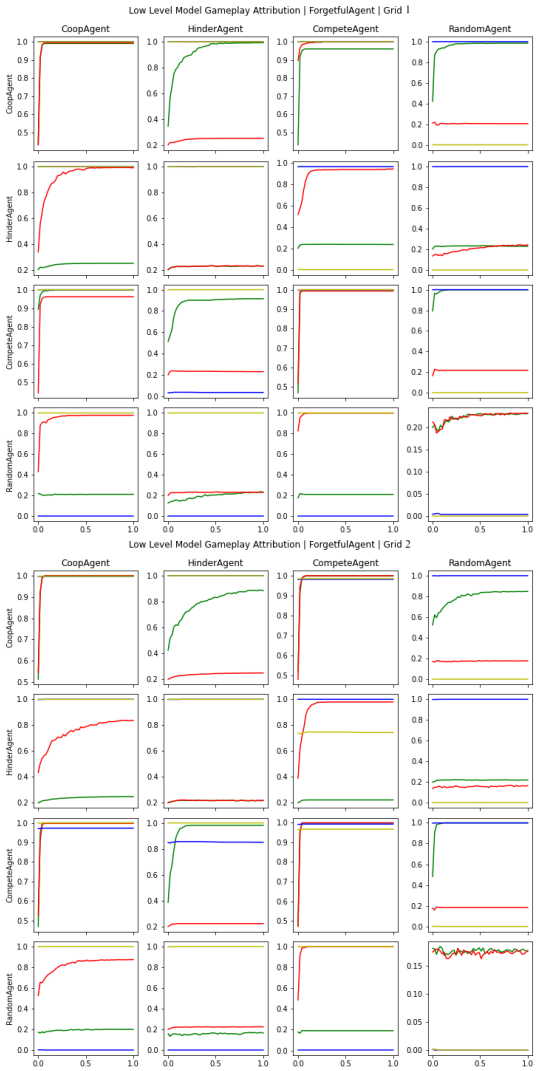


Figure 13: ForgetfulAgent versus SGA confidence over time in the correct low-level blue and yellow player strategy. Top: Grid 1, Bottom: Grid 2

Experiment 3.2: High-Level v. Low-Level Recall, in Experiment 3.2, high-level models were pitted against low-level models, during which we recorded the high-level's model prior vector (confidence in all four low-level strategies) over time. Figures 14 15 detail the outcomes and game statistics for each high level model.

Both agents, overall, do not do as well as we anticipated. First, both agents perform poorly against the low level compete agent, losing either all of the games or close to all of the games. We theorize that this could be due to the layout of the game boards used as well as how fast each game proceeds. The average length of a competitive game lasts around 2-4

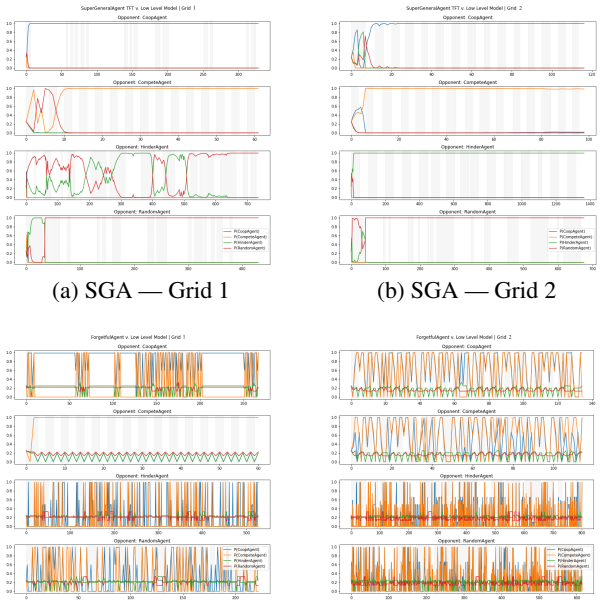
moves, and if the first move taken by the SGA is not competitive in nature, combined with the low strength of confidence in the competitive prior after just 1-2 observed moves, the other player immediately has a high likelihood of winning.

ForgetfulAgent – TFT									
		Grid 0				Grid 1			
	Opponent:	Coop	Compete	Hinder	Random	Coop	Compete	Hinder	Random
Outcome Breakdown	% wins	0	0	**0.9**	**0.6**	0	0	**0.733**	**0.567**
	% losses	0.2	**1**	0	0.267	**0.967**	**0.933**	0	0.3
	% ties	**0.733**	0	0	0.133	0.033	0.067	0	0.1
	% inconclusive	0.066	0	0.1	0	0	0	0.267	0.033
	Opponent:	Coop	Compete	Hinder	Random	Coop	Compete	Hinder	Random
Game Length	Average	8.93	2.03	17.6	7.433	4.5	3.83	26.73	20.5
	S.d.	12.45	0.1795	12.499 6	4.738	1.727	0.734	16.134	14.228

Figure 14: ForgetfulAgent Game Outcome Breakdown

SuperGeneralAgent – TFT									
		Grid 0				Grid 1			
	Opponent:	Coop	Compete	Hinder	Random	Coop	Compete	Hinder	Random
Outcome Breakdown	% wins	0	0	**0.8**	0.3	0	0	0.2	0.1667
	% losses	0.133	**1**	0	**0.6**	**0.967**	**1**	0	**0.633**
	% ties	**0.766**	0	0	0.033	0.033	0	0	0
	% inconclusive	0.1	0	0.2	0.0667	0	0	**0.8**	0.2
	Opponent:	Coop	Compete	Hinder	Random	Coop	Compete	Hinder	Random
Game Length	Average	10.966	2.067	24.533	14.367	3.9	3.3	45.533	22.567
	S.d.	15.617	0.359	15.908	14.226	0.746	0.458	10.71	18.167

Figure 15: SGA Game Outcome Breakdown



(a) SGA — Grid 1

(b) SGA — Grid 2

(c) ForgetfulAgent — Grid 1

(d) ForgetfulAgent — Grid 2

Figure 16: High Level Agent confidence scores in opponent's strategy during gameplay.

Furthermore, on the second grid, both high level players lose to the Cooperative player a majority of the time. The phenomenon is very surprising, and suggests that either the low level Cooperative player must be trained for longer iterations, or the hyperparameters of the high level player must be further tweaked. Either way, this is an important finding that warrants deeper investigation.

Evidently, both high level agents fare relatively similar to each other across the different games, with the ForgetfulAgent beating the SGA agent when against the HinderAgent and RandomAgent. However, based on the following plots, it is clear that the SGA does a far better job at identifying the correct opponent. This suggests that more research and experimentation needs to be done in order to utilize this information to design a best strategy than TFT.

In Figure 16, the alternating gray and white bands represent the successive games played.

6. Discussion

The following section provides explanations for the results displayed in the section above, and for the results that do not have a clear explanation, our team hypothesizes why the data appears as such.

Experiment 1

The results from Experiment 1 show that while in some specific scenarios, the SuperGeneralAgent model outperforms humans in attributing the correct agent types to the players, humans overall perform more accurately than the model in their predictions.

Accuracy Score: Model vs. Human across Agent Type
In general, the results show that model predictions are more accurate than human predictions for the Hinder agent and, more significantly, the Compete agent. Competing requires a player to maximize their own reward and hindering requires a player to minimize the other player's reward; both agents optimize moves by focusing on one player's utility, either their own or their opponent's. As such, the model can more precisely determine a player's optimal moves for Compete and Hinder because the goals of these agents are one-directional, requiring the maximization or minimization of only one score.

On the contrary, the model makes less accurate predictions for the Cooperate agent, and one hypothesis is that human players have different interpretations of how a cooperative player acts. To a human player, cooperation can mean a player reaching their goal at the same time step as their opponent, supporting the opponent with their role's intentions, or simply, not competing against their opponent. The model, on the other hand, defines cooperation as maximizing the joint utility of the two players which does not necessarily cover all the possible human understandings of a cooperative player.

Finally, the model overall performs significantly worse for the Random agent because it is very challenging to define what true randomness is within the context of the games. Even if a player moves in a way that appears to be random because the player switches from goal to goal, there is a chance that the player is, for instance, a Hindering player, switching between goals to throw the opponent off. Perhaps the player is a Cooperative player, appearing to switch between goals to

buy the opponent time to reach their own goal. Maybe instead, the player is a Competitive player but made a mistake on the first move and realized they should move back a step to ultimately reach their own goal with fewer steps. Because randomness spans an infinite space and cannot be easily defined with a handful of characteristic sequences, it is much harder for the model to capture whether a player is Random in comparison to the other three agents, whose optimal actions are more explicitly defined.

Accuracy Scores: Model vs. Human across Grid Type When observing the accuracy scores across grid type, the model performs much worse on Grid 2 than on Grid 1. Simply put, Grid 1 is much more limited because of its cross-shape, and there exists a smaller pool of state transitions and possible action pairs as a result. Grid 2, while containing the same number of cells as Grid 1, is square-shaped which increases the number of state transitions and action pairs that a player can take in a game. Because there are more possibilities for moves in Grid 2, there are naturally more paths to take for cooperating, competing, and hindering a player than there are in Grid 1. As such, it is more challenging for the model to predict accurately the agent type when there is a larger state transition and action space to choose from.

Because the original game data was generated from two humans playing the game against one another, there are certain combinations of agents that prevent the game data from making complete sense to the model. For instance, if two Hinder agents are playing against one another in either grid, both players have the ultimate goal of preventing the other player from reaching their goal. As a result, both players will lurk by the other player's goal, preventing their opponent from reaching their goal; however, these actions will result in an infinite game since both players will never have the intention of reaching their own goal. There are several other combinations of agents that can result in an infinite game, listed below:

1. (Hinder, Hinder)
2. (Hinder, Compete)
3. (Hinder, Cooperate)

If human players were to choose any of the above combinations of agents, at some point, one of the players must decide to abandon their original goal in order to end the game. The switching of goals is rational to the human observer because switching prevents the game from running infinitely, but the model may equate the switching of goals with the switching of agent type instead, contributing to the lower accuracy scores across the board.

Additionally, humans can quickly learn the agent type of their opponent within the first few moves and may choose to shorten play time by abandoning their original goal. Suppose two people are playing a game on Grid 1, and one cooperating and the other random. Within the first few time steps, it will be clear to the cooperative player that their opponent is likely random when they switch from goal to goal. With this knowledge, the cooperative player may choose to end the game early by entering into their goal before the random player since a random player can take up to infinitely many time steps to reach their goal. Other combinations include the following agent pairs:

1. (Random, Cooperate)
2. (Random, Hindering)

Although the following agent combinations do not require an infinite game, a player may choose to abandon their original goals to prevent the game from lasting longer.

Accuracy Scores: Agent and Grid Type Next, we discuss the accuracy scores across each agent for each grid type. In Grid 1, we observed similar accuracy scores across all agents which was in line with our expectations; however, in Grid 2, we observed that the model performed noticeably worse for Cooperate and Random but not for Hinder nor Compete. For Grid 2, the actions and state transitions of two competitive players and two cooperative players can look almost identical because both pairs of players both ultimately take the same path towards their respective goals. As such, the model has a harder time distinguishing between the two.

Match Scores Finally, we discuss the match scores across agent and grid type. For Grid 1, the match scores were highest for Compete which is due in part to the clear characteristic sequence of a competitive move, easily determined by both model and human. The lowest match scores for Grid 1 came from Hinder which, unlike Compete, does not have a clear characteristic sequence of moves, especially for Grid 1 which is stronger in determining a cooperative vs. competitive player. For Grid 2, the match scores were overall lower, with Random achieving the lowest match score. As described above, it is more challenging to truly define the space of random behavior and vice versa: random moves can appear as any one of the other three agent's moves because randomness is not well-defined.

Experiment 2

We had originally hypothesized that the model would have trouble with distinguishing between the competitive and hindering sequences for Grid 1 but also between the cooperative and competitive sequences for Grid 2; however, the model was actually able to identify the hindering sequence quite well – this may perhaps be because we have set the hindering parameter h to be sufficiently high, at $h = 3$, ensuring that only a hindering agent would want to wait in front of their own goal for a long time without entering. As predicted though, the model was not able to distinguish between the Cooperative and Competitive sequences for Grid 2. This may be because in both of these sequences, the first two steps look exactly the same: UP, RIGHT. The difference, which humans are able to detect, is that the Coop sequence waits in front of the goal for a few steps before stepping in; however, Compete

heads straight to the goal. Perhaps mathematically given our utility functions and Bayesian updates, there is no difference to the model whether a straight-shot to the goal is cooperative or competitive. Humans can also see the possibility of this sequence being collaborative (as highlighted in the 25% Coop posterior for humans); however, humans know that it is only Cooperative if the other player is also heading directly to the goal so that both can reach it at the same time; while a straight dive to the goal, regardless of the other player's position (especially given WAIT was the default action by the other player), is more likely to be competitive.

For both grids, we can also see that the model never predicts "random" as the predominant type, even though there are two sequences that humans can identify as predominantly random. One potential hypothesis for this is that humans identify as random behavior that changes back and forth very quickly. For instance, in the characteristic random sequences, there are actions that go DOWN, UP, WAIT, RIGHT, LEFT one after another, or LEFT, RIGHT, LEFT, which indicates to the human that the player continuously negates their prior action, an indication that they may not have a coherent strategy. However, the model, the way it is currently structured, does not have a way of storing previous action sequences to identify this negation. We hypothesize that it instead mainly identifies as "random" the sequences that do not appear with high likelihood for the other agent types, which is a weakness of this current model, and something that can be improved on in future extensions.

Another reason that the model may fare poorly in identifying random sequences is because of another limitation of the game board, similar to the reasoning described in Experiment 1. For Hindering sequences, if the player is successful in Hindering, the current game setup does not end the game in a certain number of steps; thus, if a player simply stands there and keeps blocking the other player, the game will go on indefinitely. The human player learns about these dynamics while playing, and thus during gameplay or attribution, even if it leaves its Hindering post at the end in order to finish the game by entering its own goal, the human recognizes that the agent is still a Hindering agent (and not a random one). To account for this in the model, we updated the priors at each step based on the posteriors of the previous step, so that initial likelihoods would factor more highly into the final prediction; however, this also means that when the first two or three steps look like it follows a certain agent's characteristic sequence, even if the following few steps are completely random (and thus distinguishable by the human), the model may have a harder time unlearning its initial predictions, thus mistaking Random for Coop (Grid 1) or for Hinder (Grid 2) in the characteristic sequences.

Experiment 3

Both experiment 3.1 and 3.2 highlight the SuperGeneralAgent's superior ability to infer a player's strategy over time over the. ForgetfulAgent. However, the SuperGeneralAgent and ForgetfulAgent fare relatively equally in the percentage of games won, lost, and tied against all opponents on both boards. This suggest that the TFT strategy employed by the SuperGeneralAgent is not the best use of its inference over the opponent's strategy.

One avenue for further experimentation would be to experiment with different high level game play strategies and rerun the same experiments, to only allow the High Level agent to employ either the cooperative or competitive strategy (even against opponents playing hinder or random strategies), or to have the high level model choose their next strategy based on the distribution of priors, choosing any low level model strategy with probability equal to its current prior.

7. Conclusions and Future Work

The goal of this project was to model joint intention and behavior for four types of low-level agents - cooperative, competitive, hindering, and random - across two different grid designs, inspired by the previous work from Kleiman-Weiner et al. and Zhang. Our team created four high-level models - CCAgent, GeneralAgent, SuperGeneralAgent, and ForgetfulAgent - using reinforcement learning and Bayesian inferencing that could infer the social intentions of game players.

To test the high-level models in action, our team designed and ran three experiments:

1. **Experiment 1**: To compare the model social intention predictions with human predictions on a set of human-generated game data

2. **Experiment 2**: To compare the differences in model prediction distributions and human prediction distributions across four characteristic, human-generated game sequences, one for each type of agent.

3. **Experiment 3**: To simulate the model playing a game against itself, both low-level versus low-level agent and low-level versus high-level agent.

Our results show that while the high-level model outperforms humans in several parts of the experiments, humans overall still predict more accurately because humans rely on prior knowledge about an agent's social intentions and the gameplay that the model cannot capture.

There are several interesting directions this research:

1. Future extensions of this project can include the addition of new low-level agents such as a Teaching or Sharing agent as well as the additions of new grid designs.

2. New high level strategies could be introduced and tested, such a Pavlovian strategy (win-stay, lose-switch), a Bayesian strategy (low-level strategy sampled with probability equal to the current prior vector) or a new strategy altogether. Then, two high level models employing different next move strategies can play against each other and see which model fares the best.

Appendix

The following section describes the individual contributions made by each team member, a detailed description of the game code, and the instructions followed for data collection.

Contributions

For a project as involved as this one, it was important that each team member took the lead on an equal portion of the project, both to promote efficiency and to ensure each individual had specific responsibilities on the project. Each team member took the lead on the following sections:

1. **Gaby**: Programmed the PyGame Python game for collecting data; programmed the high-level CCAgent in collaboration with Janice; programmed the SuperGeneralAgent and ForgetfulAgent; ran Experiment 3 and analyzed the experimental results.

2. **An**: Programmed low-level CoopAgent and RandomAgent; recruited participants for the human-generated data; designed and executed Experiment 1; analyzed and derived insights from Experiment 1 data.

3. **Janice**: Programmed low-level HinderAgent and CompeteAgent in collaboration with An; structured grid representations and extensions; recruited participants for the human-generated data; executed and analyzed results for Experiment 2.

While each member took the lead on different parts of the project, our team ensured that **every member understood thoroughly both the technical details and big-picture implications of every part of the project.** As such, **every team member is able to explain every part of the project.**

Game Code

The Game Code can be accessed via this Dropbox link: 9.66 Dropbox Folder. In the attached Game code, you will find the following files: `board.py`, `game.py`, `server.py`, `network.py`, `data_collection.txt`, and `local_viz.py`.

`board.py` The `board.py` file contains all the game-UI specific classes: the InfoDump, Player, BoardCell, Board, and Game class (each responsible for what the name implies). The file also has the GameManager class, which contains all the necessary information to coordinate the Game instances running on each individual player's computer.

`game.py` The `game.py`class is run locally by the players. The `game.py` class is responsible for running the state machine described in the Game section, updating the UI using classes from `board.py`, and sending the relevant information from a local player to the server, and as well receiving information from the server.

`server.py` The `server.py` class is responsible for starting the server for use by the players. It holds the information for connecting two players to a single game, and updating the GameManager passed between players.

`network.py` The `network.py` class sends and receives information to and from the server. It works "behind-the-scenes" to connect the `game.py` and `server.py` classes.

Data Collection

The entire system uses Python's built in socket, sys, and network packages for connecting two computers over a local WiFi network. For the game visualizations, we use pygame, a widely used package for game and UI visualizations in Python. If players do not have pygame already installed, they are asked to install it with pip3 install pygame.

As players work through multiple game rounds, we save the data and write it to a .txt file that players send to us following their game play. We also included a text file of instructions for participants to get started. Importantly, the data collected is completely anonymized and securely saved on MIT Dropbox. Due to the stay-at-home order, COVID pandemic, and assignment timeline, participants were recruited from our close circle of family and friends.

The data collected is saved in `data_collection.txt`. The format is as follows:

```
ID: <game id>
Grid:  <board id, in 1, 2>
Moves
p0:[list of blue moves from {LEFT, RIGHT, UP, DOWN, WAIT}]
p1:[list of yellow moves from {LEFT, RIGHT, UP, DOWN, WAIT}]
States
p0:[list of blue states, each in [0, 10])
p1:[list of yellow states, each in [0, 10])
```

You can visualize these games by running local_viz.py.

Google Colab

The code for our project can be accessed through Google Colab at this link: 9.66 Google Colab.

Acknowledgments

Our team would like to thank the 9.66 Course Staff for an enjoyable and valuable semester, despite the challenges of a fully-virtual semester.

References

1. Kleiman-Weiner, Max, et al. "Coordinate to cooperate or compete: abstract goals and joint intentions in social interaction." CogSci. 2016.

2. Zhang, Lily. Cooperation and competition: modeling intention and behavior in dual-agent interactions. Diss. Massachusetts Institute of Technology, 2018.