Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment

Learning Policies for First Person Shooter Games Using Inverse Reinforcement Learning

Bulent Tastan, and **Gita Sukthankar**
Department of EECS
University of Central Florida
Orlando, FL 32816, U.S.A
{*bulent@cs, gitars@eecs*} *.ucf.edu*

Abstract

The creation of effective autonomous agents (bots) for combat scenarios has long been a goal of the gaming industry. However, a secondary consideration is whether the autonomous bots behave like human players; this is especially important for simulation/training applications which aim to instruct participants in real-world tasks. Bots often compensate for a lack of combat acumen with advantages such as accurate targeting, predefined navigational networks, and perfect world knowledge, which makes them challenging but often predictable opponents. In this paper, we examine the problem of teaching a bot to play like a human in first-person shooter game combat scenarios. Our bot learns attack, exploration and targeting policies from data collected from expert human player demonstrations in Unreal Tournament.

We hypothesize that one key difference between human players and autonomous bots lies in the relative valuation of game states. To capture the internal model used by expert human players to evaluate the benefits of different actions, we use inverse reinforcement learning to learn rewards for different game states. We report the results of a human subjects' study evaluating the performance of bot policies learned from human demonstration against a set of standard bot policies. Our study reveals that human players found our bots to be significantly more human-like than the standard bots during play. Our technique represents a promising stepping-stone toward addressing challenges such as the Bot Turing Test (the CIG Bot 2K Competition).

Introduction

Although there are many factors that contribute to the immersiveness of a gaming experience, the thrill of combating an exciting adversary can lift the gaming experience from ordinary to memorable. Yet the question remains— what is the key element underlying interesting adversarial behavior? Creating human-like adversarial intelligence poses significantly different challenges from the related problems associated with creating non-player characters with realistic dialog, emotion, and facial animations (Cassell et al. 2000).

Variability in execution has been listed as an important desiderata for adversaries (Wray and Laird 2003); the ideal opponent should not be repetitive and predictable in its action selections (Schaeffer, Bulitko, and Buro 2008). However, over-reliance on randomized action selection can sabotage effective gameplay by making autonomous opponents seem like distracted amnesiacs (Hingston 2010b). Also, opponent modeling has been highlighted as an important technology that can be coupled with the bot's decision-making to create adaptive gameplay in more structured domains such as football (Laviers et al. 2009).

More fundamentally, we hypothesize that the problem with intelligent bots is that their valuation of actions is significantly different from a human player's. For instance, a human player will recklessly chase an opponent even when their health status is dangerously low just to experience the thrill of the chase. Many commonly-used game algorithms such as A* path planning optimize subtly different metrics than human players; for instance, A* implementations often use an exclusively distance-based heuristic that does not express player movement preferences to avoid certain types of terrain.

In this paper we propose an approach to resolve this problem of mismatched valuations by having our bot directly learn reward functions from expert human player demonstrations. Our experimental testbed, constructed using the Pogamut toolkit (Gemrot et al. 2009), can be used to acquire data from expert human Unreal Tournament players playing deathmatch games. We attempt to learn policies for **attack**, **exploration**, and **targeting** modes from human data. Rewards for attack and exploration actions are learned using inverse reinforcement learning (Ng and Russell 2000).

IRL attempts to solve the opposite problem as reinforcement learning; instead of learning policies from rewards, IRL recovers rewards from policies by computing a candidate reward function that could have resulted in the demonstrated policy. During the demonstrations, the most frequent actions executed by human players in a set of designated game states are recorded. A reward model is learned offline using the CPLEX solver based on a set of constraints extracted from the expert demonstration. Using value iteration (Sutton and Barto 1998), a policy is created from the reward vector and then executed by our bot. For the targeting mode, we simply fit a parameterized model to the expert data. Rather than having the bot rely on the automated targeting provided by the game, our bot samples from the learned distribution. Our bot switches between these three modes (attack, exploration, and targeting) based on

game cues such as opponent proximity and availability of resources.

We evaluate the overall performance of our bot in multiple ways. First, we compare the exploration policy learned by the bot against a set of standard Pogamut (Gemrot et al. 2009) bots on multiple maps. The results of these experiments indicate that our bot is significantly better at foraging for resources such as health pellets and weapons than a set of standard UT bots. The combat performance (attack and targeting) of our bot was evaluated by having a set of human subjects compare their play experiences with our bot vs. a standard bot. Players had the opportunity to play both bots directly in deathmatch competition multiple times, as well as to view movies of the two bots. Our findings for this experiment indicate that human players rate our bot as more human-like in gameplay.

Related Work

The 2k BotPrize competition (Hingston 2010a) has stimulated much of the related work in this area. Known as the "Bot Turing Test", the BotPrize competition pits autonomous bots vs. a panel of expert human judges drawn from the gaming industry. The agents fight a modified death match game in Unreal Tournament against a human player and a judge; to win the major prize, the bot must convince four out of five judges of its humanness.

In the 2010 competition no bot was able to win the major prize and as in previous competitions the most human bot did not manage to outscore the least "human" human player. There are, however, promising strategies that have emerged for topics such as item acquisition, steering, and firing techniques. Ideas such as circular strafing, imprecise aiming, and more complex item seeking strategies have been demonstrated by various bots (e.g, (Hirono and Thawonmas 2009)).

The top-scoring bot in 2010 was Conscious-Robots from Carlos III University in Madrid. In many of the submissions, human-like characteristics were implemented for navigation. Relational databases have been used to provide the bot with long term memory of hotspots where actions commonly occurred in the past, instead of going to the last place an enemy was seen (Hirono and Thawonmas 2009). In contrast, our bot uses IRL and expert human demonstrations to learn key locations for resource acquisition.

The bots, like humans, have different objectives depending on their state. The ICE 2008/2008 bot had two behavior states: item collection and combat (Hirono and Thawonmas 2009). The 2008 FALCON bot implemented four states allowing for more variability in action: running around, collecting items, escape, and engage (Wang et al. 2009).

(Hingston 2010b) notes that two of the most historically common failures among competition bots have been overly accurate shooting and lack of aggression compared to human players; our approach was specifically designed to address these shortcomings. Note that due to the rules of the competition, it would not be possible to use inverse reinforcement learning to learn an exploration policy directly on the competition map, but it remains a feasible approach for attack policies which are not map-specific.

Inverse Reinforcement Learning

To learn attack and exploration policies from expert player demonstrations, we first use inverse reinforcement learning to learn a reward model from a set of player demonstrations. First the expert's demonstration is converted into a set of linear programming constraints over states, actions, and rewards. The goal is to find a reward model that effectively explains the player's action choices. This is done by assuming that the observed policy maximizes the selected reward model, and searching for reward values that minimize the difference between an optimal policy (under that reward model) and the observed set of demonstrations. To solve the resulting set of equations, we use the CPLEX solver offline.

We use the IRL formulation defined by (Ng and Russell 2000) and maximize:

$$\sum_{i=1}^{N} \min_{a \in \mathcal{A}} \{(\mathbf{P}_{a_1}(i) - \mathbf{P}_a(i))(\mathbf{I} - \gamma \mathbf{P}_{a_1})^{-1}\mathbf{r}\} - \lambda ||\mathbf{r}||_1$$

$$s.t. \quad (\mathbf{P}_{a_1} - \mathbf{P}_a)(\mathbf{I} - \gamma \mathbf{P}_{a_1})^{-1}\mathbf{r} \succeq 0 \quad \forall a \in \mathcal{A} \setminus a_1$$
$$|r_i| \leq r_{\max} \quad i = 1, \ldots, N$$

where a_1 is the action index of the policy for current state, $\mathbf{P}$ is the state transition matrix, γ is a discount factor, and λ is the penalty coefficient. $\mathbf{r}$ is the reward vector and $r_{\max}$ is set to 1. The max-min formulation can be converted to a problem of only maximization by adding new variables, y_i, to the problem as follows:

$$y_i \leq \{(\mathbf{P}_{a_1} - \mathbf{P}_a)(\mathbf{I} - \gamma \mathbf{P}_{a_1})^{-1}\mathbf{r}\}$$

Maximizing y_i is equivalent to minimizing the right-hand side part of the objective function. By adding the sum over ρ_i to convert the first-norm of rewards into linear form, the overall linear programming formulation becomes:

$$\text{maximize} \sum_{i=1}^{N} (y_i - \lambda \rho_i)$$

$$\begin{aligned} s.t. \quad y_i &\leq \{(\mathbf{P}_{a_1} - \mathbf{P}_a)(\mathbf{I} - \gamma \mathbf{P}_{a_1})^{-1}\mathbf{r}\} \quad \forall a \in \mathcal{A} \setminus a_1 \\ r_i &\leq \rho_i \\ -r_i &\leq \rho_i \\ r_i &\leq r_{\max} \end{aligned}$$

If we define N as the number of states (128 for exploration and 12 for attack mode) and A as the number of possible actions (4 for both exploration and attack) we have $3N$ variables and $(A-1)N + 3N = (A+2)N$ constraints. Hence, we have 384 variables, 768 constraints for exploration mode, and 36 variables, 72 constraints for attack mode; due to our simple state space representation, CPLEX can calculate values for all the reward variables within a few seconds.

By using IRL, we avoid having to hand-code rewards for particular states based on domain knowledge and instead can directly leverage the experience of expert players. After IRL returns the reward model, we run value iteration offline to learn a policy. During gameplay, the bot simply performs a policy lookup based on the current game state which can be executed very rapidly.

Bot Design

Our system was trained to play two-player deathmatch mode in Unreal Tournament; the winning objective is simply to score the highest number of kills during a set period of time. We created our game bot and our data collection bot using the Pogamut (Gemrot et al. 2009) platform.

Pogamut was developed to allow programmers to connect to the Unreal Tournament (UT) server using Java. It includes three types of connections: server, observation and bot. The server connection allows the developer to access player and map information, add a built-in UT bot or remove a bot from game. The observation connection opens a link to a player and gives information about the world from the perspective of that player. This information not only includes the player's location, shooting and orientation data but also provides information about other items and players in sight. The bot connection creates a programmable bot that can move, jump, shoot, detect visible players, and spot items.

Our bot uses a finite state machine to switch between three different modes; we learn the policies for each mode separately as described in the following sections. **Exploration** mode is designed to gather resources (ammo and health packs) from the environment when the user is not fighting opponents. As soon as an enemy is visible, the bot switches into **attack** mode. The **targeting** mode is triggered during attack mode when firing at the opponent.

Exploration Mode

To learn a policy for the exploration mode, we divide the current map is into an occupancy grid. Since certain cells are hazardous, only regions of the map that have been explored safely are included in the occupancy grid. Therefore we initialize the map by having the user perform a short exploration and then using the initial exploration data to generate a safe occupancy grid. Using the grid map, we collect a series of short (15 second) segments from the user's explorations for processing. During a single segment, the user typically explores one region in the map.

This data demonstrates how a user explores various map areas and which spots are preferentially visited. We then convert the user's movements to a reduced set of actions (up, down, left, and right). State transitions are recorded, and from that data, actions are extracted. We predefine a simple transition matrix between states in which many states are linked by deterministic transitions. First, the path demonstrated by the expert is cleaned by by filling the gaps and diagonal jumps between cells to align with the four-way connectivity of our action space. A reduced state space is generated using only the cells covered by the path.

The demonstration is converted into actions which are used to learn the reward model and policy for that section of the state space using the following procedure:

1. Our IRL script is used to automatically generate constraints based on the expert demonstration.
2. A consistent reward vector for the observed states is computed using CPLEX.
3. Rewards for unobserved states are set to 0.

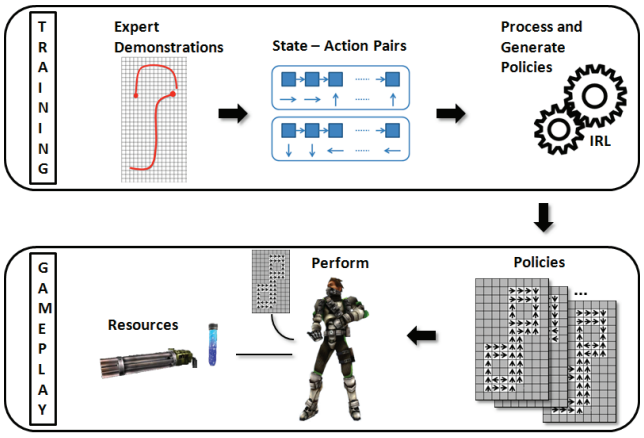

Figure 1: Architectural diagram showing training and gameplay in **exploration** mode.

4. Value iteration is executed until the error threshold reaches 10^{-4}.
5. The policy, the start, and end cells for the demonstration are saved.

The process is repeated for each demonstration to create multiple policy maps for each area of the map. The training process is depicted in the top field of the Figure 1. At the end of training process, a set of policies are generated which are then used by the bot during game play. During the game, the bot either selects the policy closest to its current position or chooses to explore a completely new region (30%). To move to a new region, it randomly selects a new policy and then uses A* search to move between its current position and the starting location of the next policy. The bot collects any resources (weapons or health packs) it detects while in exploration mode before continuing to follow the learned policy.

Attack Mode

To train the bot for attack mode, we collected 45 minutes of log files at 20 frames/sec of an expert player fighting an opponent. The log file contains the following fields: 1) timestamp 2) location (x,y,z) 3) health level 4) shooting status 5) enemy visibility 6) jumping status 7) angle to the enemy in XY plane 8) currently used weapon. After data acquisition the log file is filtered to create state-action pairs for the parts of the session that the opponent is visible to the player.

In attack mode, the state representation is based on a combination of the health and distance status of the player, yielding twelve total states. The state representation is shown in Table 1. The four possible actions are: 1) shooting and moving 2) shooting only 3) moving only 4) standing still. Note that even when the bot is in attack mode, it is not constantly firing; it can opt to hide or disengage by selecting movement actions and going to the appropriate distance.

The distribution of these actions across states represents the aggressiveness of the player. To identify the action being executed, we considered windows of 250 ms and eval-

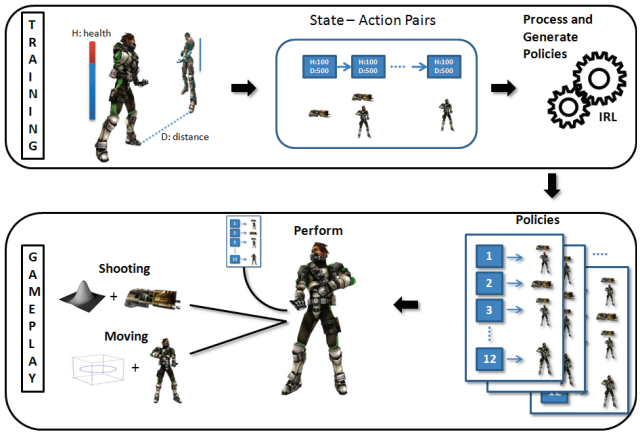

Figure 2: Architectural diagram showing training and gameplay in **attack** mode.

uated the percentage of shooting and movement the player performed during that time period to assign each window a state and action value.

Distance (UT units)	Health 30	60	90	
less than 500	1	4	7	10
between 500, 1000	2	5	8	11
greater than 1000	3	6	9	12

Table 1: State representation for **attack** mode.

After the state-action pairs from the expert's demonstration are extracted, a frequency count is performed over the entire dataset to create a state transition probability matrix. The transition probability matrix and policy data is then used as the input to CPLEX to learn a mapping between state and rewards. Using these rewards, we execute value iteration to learn the optimal policy that the bot should perform. This policy is used to guide the bot's movement and shooting during attack mode; note that the learned policy only specifies the general action, not actual direction of movement or bot orientation which need to be calculated at run-time. Figure 2 shows the operation of our attack method.

Targeting Mode

To create a more human-like shooting pattern, we collected data on the expert players' firing patterns and fitted it to a separate Gaussian distribution based on the distance between the player and the enemy. The learned parameters are shown in Table 2. Interestingly, the angular variance of the player's shooting angle is actually reduced when shooting at farther ranges; this reflects the tendency of most players to spend a longer time aiming at distant targets.

Whenever the bot can see the enemy or detects that an enemy is in the vicinity, it switches from **exploration** to **attack** mode. The bot is alerted to the presence of the enemy by detecting damage, seeing a bullet shot by enemy, or hearing noise from the enemy. When the bot switches to **attack** mode, it calculates the current state using the relative health level and distance to the enemy. The bot selects an action (attack, move, or both) based on the learned policy. If the bot is going to attack, it changes its weapon to most destructive weapon that it carries. After generating an angle to shoot by drawing from the applicable Gaussian distribution, the bot calculates the location to shoot. If the bot opts to move, it runs and jumps in a circle to maintain the distance recommended by the learned policy. The center of the circle is mapped to a navigation point within a known safe region to protect the bot from moving to unknown hazardous locations.

Evaluation

We evaluated both the exploration and combat performance of our bot. For exploration, we seek to create a bot that can effectively forage for health packs and weapons, based on a policy learned from expert players on the same map; essentially the bot is learning the areas that produce the richest yield of resources. The metric used is simply the amount of health packs the bots were able to collect within a five minute interval. We compared our bot to two standard bots included within the Pogamut distribution: Navigation and Rayscan.

The Navigation bot simply moves from one navigation point (navpoint) to another and periodically uses A* to move to a new randomly selected navpoint. This bot is good at covering large sections of the map very rapidly and picking up resources along the way. The Rayscan bot scans the game world with using ray-tracing and performs obstacle avoidance to avoid hazards; the advantage of this policy is it enables exploration and foraging in areas not covered by the navigation grid. We performed five runs of each bot over the same map. As Figure 3 shows, our bot, using the exploration policy learned by IRL, outperforms the other two fixed policy bots at this task and a t-test indicates that the differences are statistically significant ($\alpha = 0.01$).

User Study

To evaluate the attack mode we performed a user study to evaluate the "humanness" of the bots' combat performance. The goal of the user study was to determine whether the users' found our bot to be more human-like than a UT bot without learning. Against autonomous UT bots, our bot played competitively at level 3 out of 7; we did not attempt to optimize our bot further since we felt that it was desirable

	Distance (UT cms) 0 to 400	400 to 700	above 700
μ	-0.4°	0.5°	0.1°
σ	9.4°	5.4°	3.8°

Table 2: Learned Gaussian parameters for different shooting distances.

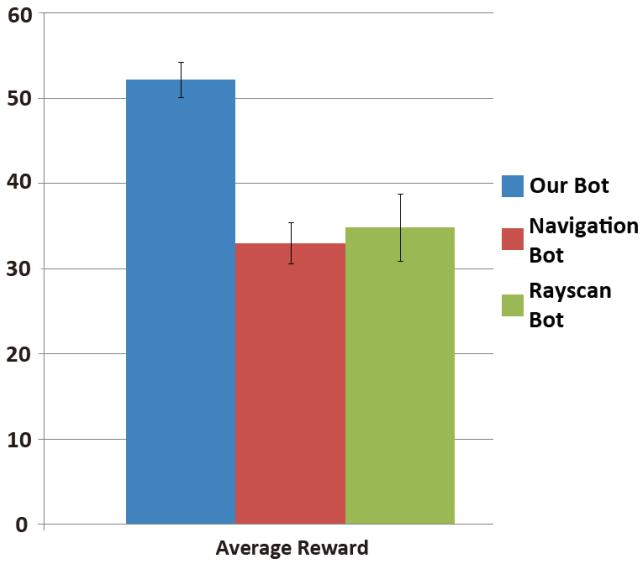

Figure 3: Evaluation of **exploration** mode. Our bot outperforms the other non-learned Pogamut bots, Navigation and Rayscan, at foraging.

for all players to occasionally beat the bot and for intermediate players to be able to beat the bot half the time. The experience levels of the thirteen participants, based on self-reported hours of play, ranged from beginner to expert; all participants were male between the ages of 18-28.

After a short training period to familiarize the participants with Unreal Tournament, we explained to the participants that they were going to play two separate games: one against a bot and the other against a human player. In reality, both tests were executed against bots, our bot and a UT bot using a non-learned policy. Subjects were asked to look for clues about the humanness of the bot based on shooting, movement and overall abilities. Each session consisted of the subject playing four combats, viewing a movie of the bots, and responding to questionnaires. After each pair of combats, players were asked to respond to a questionnaire based on the previous combats. Participants were asked to rate the bots' shooting, movement, and behavior on a 1 to 7 scale, where 1 indicated more bot-like and 7 indicated more human-like. Participants also were asked how fun each opponent was to play. At the end, we asked the participants to watch the two "players" in combat against one another and evaluate the players' shooting, movement and overall behavior.

From the questionnaires of the 13 participants, we divided the question responses into four parts: shooting, movement, overall performance, and fun. We plotted the number of human-like and bot-like answers for the UT bot and our bot, then performed Fisher's exact test which is based on two categorical datasets of small sample size. For the results we used left tail (negative correlation), which shows that the observations tend to lie in lower left and upper right of the table. In our case, having observations in lower left and upper right indicates that the participants rated our bot as more human-like. As we can see from the results for shooting, movement and also overall behaviors, our bot appears to engender statistically significantly differences in participants' perception of the bots' human-like behavior ($p < 0.05$).

Attacking	Human	Bot
UT Bot	5	8
Our Bot	11	2
Left p value = **0.02**		

Table 3: Questionnaire responses relating to attacking (statistically significant)

Movement	Human	Bot
UT Bot	5	8
Our Bot	10	3
Left p value = **0.05**		

Table 4: Questionnaire responses relating to movement (statistically significant)

Overall	Human	Bot
UT Bot	5	8
Our Bot	11	2
Left p value = **0.02**		

Table 5: Questionnaire responses relating to overall performance (statistically significant)

Fun	Fun	Boring
UT Bot	12	1
Our Bot	11	2
Left p value = 0.522		

Table 6: Questionnaire responses relating to fun experienced

Overall	Human	Bot
UT Bot	7	6
Our Bot	9	4
Left p value = 0.344		

Table 7: Questionnaire responses relating to spectator mode

Interestingly, our bot was more successful at fooling human players during gameplay (which is traditionally thought to be a harder test) than during spectator movie mode. When spectating both the bots, the players are able to slow the movie down and notice repeated patterns and occasional movement glitches exhibited by both bots. The difference between our bot and the UT bot is not statistically significant in movie mode. Also the reported fun between play

experiences with the two bots was not statistically significant.

Conclusion

In this paper, we demonstrate an inverse reinforcement learning approach for teaching first-person shooter bots to play using expert human demonstrations. Our bots outperform bots using fixed policies at map exploration, and our user study evaluation reveals statistically significant improvements in the "humanness" of our bot. However, IRL is essentially an underconstrained problem; there are many possible reward models to fit a single observation sequence. Hence we feel it will be valuable to consider including additional constraints, beyond the ones required to create a valid reward model. In future work, we plan to work on the issues of generalization; instead of learning demonstrations from a single map, we will use demonstrations across different maps to learn a more general model of player combat and exploration preferences.

Acknowledgments

This research was supported in part by DARPA award N10AP20027.

References

Cassell, J.; Sullivan, J.; Provost, S.; and Churchill, E., eds. 2000. *Embodied Conversational Agents*. MIT Press.

Gemrot, J.; Kadlec, R.; Bida, M.; Burkert, O.; Pibil, R.; Havlcek, J.; Zemck, L.; Lovic, J.; Vansa, R.; Tolba, M.; Plch, T.; and Brom, C. 2009. Pogamut 3 can assist developers in building AI (not only) for their videogame agents. In Dignum, F.; Bradshaw, J.; Silverman, B.; and van Doesburg, W., eds., *Agents for Games and Simulations*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg.

Hingston, P. 2010a. Botprize 2010: Human-like bot competition. Website. `http://www.botprize.org`.

Hingston, P. 2010b. A new design for a turing test for bots. In *IEEE Computational Intelligence and Games (CIG)*.

Hirono, D., and Thawonmas, R. 2009. Implementation of a Human-Like Bot in a First Person Shooter: Second Place Bot at BotPrize 2008. In *Proceedings of Asia Simulation Conference 2009 (JSST 2009)*.

Laviers, K.; Sukthankar, G.; Molineaux, M.; and Aha, D. 2009. Improving offensive performance through opponent modeling. In *Proceedings of Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE)*.

Ng, A. Y., and Russell, S. 2000. Algorithms for inverse reinforcement learning. In *Proceedings of 17th International Conference on Machine Learning*, 663–670.

Schaeffer, J.; Bulitko, V.; and Buro, M. 2008. Bots get smart. *IEEE Spectrum*.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press.

Wang, D.; Subagdja, B.; Tan, A.-H.; and Ng, G.-W. 2009. Creating Human-like Autonomous Players in Real-time First Person Shooter Computer Games. In *Twenty-First Annual Conference on Innovative Applications of Artificial Intelligence (IAAI'09)*.

Wray, R., and Laird, J. 2003. Variability in human behavior modeling for military simulations. In *Proceedings of Behavior Representation in Modeling and Simulation Conference (BRIMS)*.