# Training Goal Recognition Online from Low-Level Inputs in an Action-Adventure Game

**Kevin Gold**
Rochester Institute of Technology

## Abstract

A method is presented for training an Input-Output Hidden Markov Model (IOHMM) to identify a player's current goal in an action-adventure game. The goals were Explore, Fight, or Return to Town, which served as the hidden states of the IOHMM. The observation model was trained by directing the player to achieve particular goals and counting actions. When trained on first-time players, training to the specific players did not appear to provide any benefits over a model trained to the experimenter. However, models trained on these players' subsequent trials were significantly better than the models trained to the specific players the first time, and also outperformed the model trained to the experimenter. This suggests that game goal recognition systems are best trained after the players have some time to develop a style of play. Systems for probabilistic reasoning over time could help game designers make games more responsive to players' individual styles and approaches.

## Introduction

In games that do not obey a linear plot, it is a common occurrence for a player to return to a game after a long hiatus, only to have no idea what she was trying to accomplish last. For example, a player logging in to *World of Warcraft* (Blizzard, 2004) for the first time in months will have a quest log full of goals, but may not have been actively pursuing any of them. Instead, the player may have been headed back to town to sell items, headed to a new zone, or grinding (repeatedly killing monsters) to reach the next level. The player may not have known at the time of logout just how long it would be before playing next, and therefore may not have made any note of what she was doing. Goals such as exploration, heading to a destination, or grinding may not be obvious from any particular action, but may instead become apparent through a pattern of behavior over time. If the game had some way of recognizing such patterns, it could store its guess about what the player's goal was, and remind the player what it was when the player returns to the game.

The present paper is about a system that can be trained to recognize some common video game goals from a player's

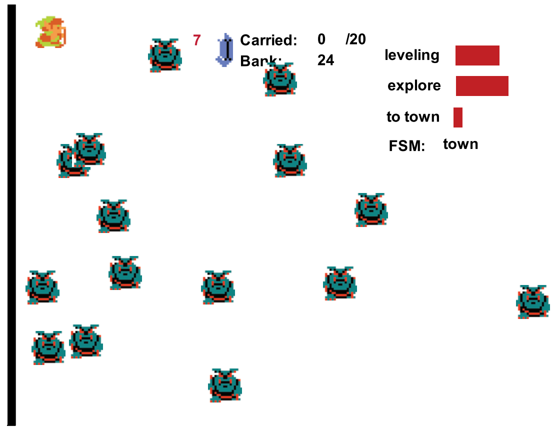

Figure 1: A screenshot of the game providing realtime feedback about its beliefs about the player's intentions. Lengths of the bars in the upper right correspond to goal probabilities.

pattern of behavior over time. The method can produce realtime feedback about the game's degree of belief that the player is pursuing each goal (Figure 1). Unlike similar previous work, the system is designed to recognize actions from relatively low-level inputs – the player's moment-to-moment movement and sword-swinging. The method can be trained and applied online, using very fast operations. The training requires only counting actions when the player is given known goals; given known goals, learning can occur online, without churning through the player's action history. Applying the algorithm then only requires a few matrix multiplications with each new observation of the player's actions.

Probabilistic reasoning over time is arguably the best strategy for making this kind of decision online in a game. Low-level actions can be ambiguous; is the player walking south to approach the monster, or the area behind the monster? Rule-driven strategies popular in the game industry, such as finite state machines and behavior trees (Isla 2005), may have a difficult time distinguishing between player motives, because they often fail to take into account the rest of the player's history. Other methods may be slow and re-

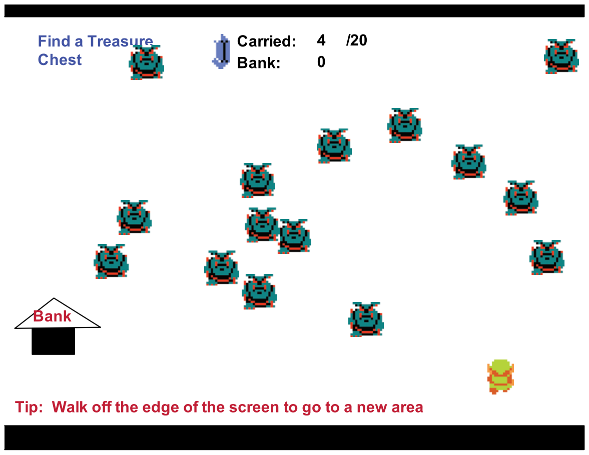

Figure 2: A screenshot of the game, under the training condition that gives explicit goals. Feedback about beliefs is not provided during training and testing, to avoid bias.

quire significant computation to update in response to the player's actions. By contrast, the model to be presented can update its beliefs in constant time, with just a single matrix multiplication and some scaling operations. It also has the advantage of being relatively transparent to inspection, with conditional probabilities that are readily interpretable. Speed is essential for an algorithm's viability the game industry, where AI is not allowed much processing power, while transparency is important to the game designers, who have the final say over agent behavior.

The present work is meant to answer three questions. First, can a trained probabilistic model provide better online predictions for player behavior than a hand-designed finite state machine? Second, does it make sense to apply the same trained model to multiple players, or should these models be trained to each individual player? And third, is it possible to train such models during the "tutorial" while the player is still learning the game, or is this training data not as good as later measurements in predicting the player's goals?

Machine learning techniques that attempt to identify goals can take advantage of the fact that the game can temporarily control the player's goals, in the form of specific quests or instructions. This is useful for establishing ground truth for training. In fact, the current structure of modern video games dovetails perfectly with the machine learning researcher's aims, because games typically begin with tutorials in which the player's goals are directed. Before the player is allowed to choose goals, the player may be given a series of directed goals to ensure that the player knows the possibilities that the game offers. This offers the tantalizing possibility that as the human player learns the game, the game can "learn the player." This model could then be updated whenever the player's goals are known, as is often the case for the obligatory plot-critical segments of a game.

The target application in the present paper is a simple action-adventure game with an interface similar to *The Legend of Zelda* (Figure 2). Players can accumulate money by either slaying monsters or finding treasure chests throughout the game world. The game learns conditional probabilities for various actions under three possible hidden states: Explore, in which the player is searching for new treasure chests; Grind/Level, in which the player is killing monsters repeatedly; and Return, in which a player is attempting to return the money to a bank before it is lost to attacking monsters. These general goals – reaching a known location, fighting enemies, and attempting to find new locations – apply to several popular game genres, including MMORPGs such as *World of Warcraft*, action-adventure games such as the *Legend of Zelda* series, and sandbox-style action games such as *Grand Theft Auto IV*. In all such cases, the interpretation of the player's movement patterns is likely to be critical in identifying the current goal.

The paper will not deal with the issue of learning how often particular players indulge in particular goals; i.e., this is not about learning players' preferred "play style" at a high level (Thue, Bulitko, and Spetch 2008). Rather, the paper will deal with learning the players' low-level behavior under particular goals, with the intention of recognizing when those particular goals are being pursued. Thus, the transition models, which govern how often and under what conditions players switch goals, will be fixed for the experiments to be described; this is a paper purely about learning the observation model, so that player goals can be recognized in the future. Moreover, it is assumed that computation is at a premium in these games, so the methods presented will sacrifice some accuracy and power in exchange for being trainable and applicable entirely online, without making multiple passes over the player's action history.

Plan recognition using Bayesian methods was first used in the domain of natural language story understanding (Charniak and Goldman 1993). Previous goal recognition work in the context of video games has typically been in the context of non-action-oriented games in which inputs are either less ambiguous or involve some verbal component. Typical examples include *The Restaurant Game*, which built trees of action sequences based on observed player actions to recognize sequences similar to those previously observed (Orkin and Roy 2007); an action recognition system in the RPG *Neverwinter Nights* that used actions and an Early parser to disambiguate speech (Gorniak and Roy 2005); the academic/art game *Façade*, which used pattern-matching on typed text to guess player attitudes towards its characters (Mateas and Stern 2007); and systems that used Bayesian reasoning to guess player goals in an online MUD (Albrecht, Zukerman, and Nicholson 1998) and in an interactive storytelling environment (Mott, Lee, and Lester 2006). Of these, the last two are most similar to the present work, but occurred in much less ambiguous settings, that of a text-based online adventure game and a first-person educational "science mystery" game, respectively. In an action game, player inputs can be much more ambiguous as players engage in complicated maneuvers to avoid enemy attacks; "moving east" could be highly ambiguous if a town, an unexplored map edge, and a monster are all in that direction, while the player is also being chased by a different monster from the west.

Application of keyhole plan recognition to action games has included using plan recognition networks for the networked space combat game Netrek (Huber and Hadley 1997), support vector machines to identify plays in the football video game RUSH 2008 as they are being executed (Laviers et al. 2009), and HMM model comparison for recognizing military operations played out in Unreal Tournament in the absence of opponents (Sukthankar and Sycara 2005). The present approach has the advantage over these of being able to model goal shifts; while plan recognition networks, SVMs, and HMM model comparison are all designed to grant a single classification to a whole time series, hidden state inference can deal with situations in which a player abandons a goal in favor of a new one. The current work also has the advantage of comparing probabilistic goal recognition to the simpler industry approach of using a finite state machine, to give industry professionals a better idea of the return on the added time investment of programming and training such a model. It does reside within a much larger research context of action recognition from continuous inputs using Hidden Markov Models (Lee and Kim 1999; Starner and Pentland 1997), but reviewing this background is beyond the scope of this paper.

Previous work has studied a similar model with hand-chosen probabilities, which was shown to be more effective than an FSM as long as the extra IOHMM goal-switching structure was used (Gold 2010). The current paper is the first to address training such a model to user data. In particular, it addresses whether it is more worthwhile to train to individual players, or player behavior in the aggregate; and whether players need experience with the game before the models are useful.

## Methods

### Game

The game was a top-down action-adventure game running at 30 fps. The player's available actions were to move in eight cardinal directions, or jab a sword up, down, left, or right. Each screen contained 15 monsters, randomly placed, which would charge the player at 4 pixels/frame (120 pixels/second) if the player approached within 150 pixels (the "aggro radius"). Each edge of the screen was either blocked by a wall or led to another screen, with screens arranged in a $5 \times 4$ map. The player started at a "Town" sprite near the middle of the world map. Six locations on this map, each at least 2 screens away from start, had a 33% chance of containing a treasure chest.

If a monster hit a player, the player restarted either from the bank or the location of the last treasure chest pickup, wherever the player visited last. On being attacked with a sword, a monster died and was replaced with a gem. A monster respawned randomly on the screen 5 seconds after being killed, and all monsters respawned if a player left a screen and came back to it.

During training and evaluation, in the upper-left corner, the game instructed the player what to do next: "Kill 20 monsters," "Find a treasure chest," or "Return to the bank." Each goal was given twice: once to train the underlying model, and a second time to establish ground truth for the test condition. (Ostensibly, the game could be played without such instructions, and this is the point; but the instructions are necessary to establish ground truth for evaluation.)

Instructions at the bottom of the screen appeared three times: at the beginning of the experiment, to tell the player how to move and attack; on beginning the "Find a treasure chest" goal, to inform the player that it was possible to move off the edge of the screen to reach a new area; and on the player's first death during the "Return to town" goal, to state that the player had been returned to the location of the last treasure chest.

### Finite State Machine

A finite state machine (FSM) provided a baseline for predictions about player goals. When the player reached a previously unexplored screen, the finite state machine predicted that the player's current goal was Explore. When the player entered a square closer to the bank than the current square, the FSM changed its prediction to Return. When a player killed a monster, the Finite State Machine set its prediction to Level.

### Probabilistic Model

The model is an Input-Output Hidden Markov Model (Bengio and Frasconi 1995) in which the input consists of whether the player has just achieved a goal, the hidden state represents the player's current goal, and the output is the low-level input from the keyboard, interpreted in the context of nearby landmarks.

IOHMMs have a similar structure to Hidden Markov Models (HMM; (Rabiner 1989)), but with additional context information that can change the transition matrix and observation probabilities. When there is additional context information that can cause a state to change, or can cause the behavior within a state to change, IOHMMs best represent the causal structure of the problem for the purposes of Bayesian reasoning (Figure 3). When this additional context information is known, IOHMMs are just as efficient to update their hidden state beliefs as Hidden Markov Models. They are ideal for use in games because games often have access to a wealth of contextual information that can help identify the player's current goal, beyond the player's actions.

Here, the additional context information is used primarily to change the HMM transition matrix when the player achieves a goal. There are two transition matrices in use: one with probabilities that tend to perpetuate the current goal, and another that suggests the player is likely to switch goals, which is swapped in when the player has just achieved a goal. This is an effect that is awkward to achieve with a normal HMM; including "goal achieved" as another observation would double the number of observation categories, increasing the time necessary to train, and would unnecessarily mix observables (whether a goal is achieved) with unknowns (the player's current goal) in the hidden state variable, since there are different transition probabilities associated with each. In general, Bayesian models can be more

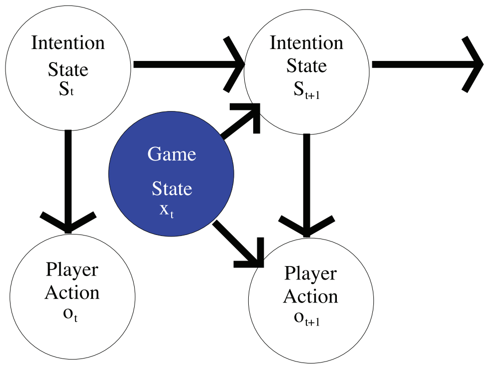

Figure 3: A slice of an Input-Output Hidden Markov Model (IOHMM) for games. The game state is a known variable that can affect the conditional probability of the player's switching goals, and the player's actions given those goals.

concise when they reflect the causality of the situation (Russell and Norvig 2003). Furthermore, elsewhere it has been shown that without this extra structure, HMMs do not outperform the FSM described earlier on the goal prediction task to be described (Gold 2010).

The three hidden states $G$ corresponded to the three goal types, Explore, Level, and Return. The possible observations $A$ were Swing, Stop, Approach Unexplored and Monster, Approach Unexplored Without Monster, Approach Town And Monster, Approach Town Without Monster, Approach Monster, Approach Gem, Other Movement. A movement counted as an approach to unexplored territory or the town if it included at least one correct basic cardinal direction, so that if the town was to the upper-left, movement up, left, or diagonally up and to the left counted as an approach. Since monsters could not be attacked diagonally (an artifact of the attack sprites used), a diagonal move toward a monster did not count as approaching it, since this generally did not signal an attack; however, a non-diagonal move toward the circle defined by a monster's "aggro radius" in which it would attack did count, since this was a common way to bait monsters. The probabilistic model consisted of the conditional probabilities $p(A_t|G_t)$ for each action $A$ and goal $G$, and the probabilities $p(G_{t+1}|G_t, G')$ for each goal $G$ and a boolean variable $G'$ that indicated whether a goal has just been achieved.

The transition matrix $T_{G'}$ thus varied depending on whether a goal had been recently achieved. One matrix $T_0$ with 0.9 along the diagonal was used for continuing any goal in progress, and another with higher out-of-state transitions was used for each of the possible goals $G'$ that could be completed in the last time step.

The time $t$ was indexed by actions instead of time. Every time the player's keyboard input changed, this was used as an observation, and the model updated. The observations were also updated every 500ms as the player moved, even if input did not change, since the context of an action could change over the course of a move.

In updating the beliefs at each time step, the vector of probabilities was updated using the Forward algorithm (Russell and Norvig 2003). Let the current vector of probabilities for the hidden goals be $\vec{G}_t$, and $P(a|g)$ be the probability of the observed action under a given goal $g$. Then on receiving a new input from the player $o_{t+1}$, the algorithm for updating the probabilities of each hidden state is:

```
$\vec{G}_{t+1} = T_{G'}^T \vec{G}_t$
for each hidden state $g$ do
    $s_{g,t+1} \leftarrow s_{g,t+1} * P(a|g)$
end for
$\vec{G}_{t+1} \leftarrow$ normalize($\vec{G}_{t+1}$)
```

This provided the probability $p(G_t|G_{1...t-1})$ at each time step, mimicking the information the game would have available for determining the player's current goal at each time step.

For interpreting the player's goal at the current time step, no further computation is necessary; the "backward message" of the Forward-Backward algorithm only affects beliefs about hidden states at the previous time steps (Russell and Norvig 2003). It would, however, be trivial to add some lag $\ell$ to the model and apply the Forward-Backward algorithm instead, so that it could use the most current evidence to re-interpret the beliefs it had about the player's goals $\ell$ actions ago. The current work forgoes the extra lag to achieve maximum swiftness of response to goal changes, at the potential expense of accuracy.

The Forward algorithm is also not to be confused with the Viterbi algorithm, which finds the most likely sequence of actions up to the present time step. The Forward algorithm returns the *total* probability of each hidden state, regardless of the history of hidden states that led to it, which may be different from the Viterbi prediction (Russell and Norvig 2003). For developers attempting to build an algorithm that builds a single narrative for the player's actions, as opposed to simply making a best guess about the current state, the Viterbi algorithm would be more appropriate.

## Training the Observation Model

For computational efficiency and simplicity, priors on the model probabilities were taken into account by using pseudocounts of one observation for each goal-action pair – an old strategy known as "Laplace's law" that is equivalent to a Bayesian estimator with a uniform prior on events (Manning and Schütze 1999). An observation probability could then be computed later by dividing the sum of the action counts and pseudocounts by the total number of actions made in that state. All three training conditions – "Kill 20 monsters," "Find a Treasure Chest," and "Return to Town" – could often be completed in under 5 minutes.

Given the sparsity of data for transitions, transition models were not trained, but set to be plausible transition models for the genre of game. This consisted of out-of-state transition probabilities of 0.05 when no goal had recently been achieved; probabilities of 0.5 of transition to the other two states when "Return" had recently been achieved; and a probability of 0.95 of transition to Return if a treasure chest had just been found, since in the final game, the player would not be able to pick up more treasure until the treasure chest had been brought to town. No particularly different transition model was used for the "Level" goal, since this goal in practice can be pursued indefinitely.

## Experiment 1: Player variability and first-time players

This experiment was designed to assess two questions: first, whether training the model was better than a finite state machine across all players; and second, how well a model trained on one player could be applied to novice players.

21 players were recruited online for the study, of whom 19 successfully finished both the training and testing rounds. The players' experience with games ranged from complete novices to experienced. Each run of the study consisted of training the observation model to the players' actions during the training phase, then running the three goals again with each of three models making online predictions of the player's current goal. The three models were the trained probabilistic model, the Finite State Machine describe in the Methods section, and a model previously trained on the experimenter, which itself predicted the experimenter's goals with an average of 87% accuracy over six test trials. All three algorithms were always run in parallel on the same time series, justifying a paired $t$-test in the analysis.

### Results

The models trained to each individual player performed significantly better (66% accuracy) than the finite state machine (54% accuracy, $p < 0.01$), but their improvement over the model trained to the experimenter only approached significance ($p = 0.18$).

## Experiment 2: Effects of experience

Informal reports from subjects who continued to play the game after Experiment 1 suggested that the method's performance improved as subjects became more experienced with the game. To test this hypothesis, all subjects from Experiment 2 were invited to repeat the experiment. Five subjects responded to the invitation. Of these, two had played the game on their own in the interim, resulting in three players that had done one full run and two that had done 3 full runs previously.

Subjects' training data from Experiment 1 was not used at all, to avoid confounding the effect of more training data with the effect of experience.

### Results

All five players produced trained models with higher accuracies than their last runs, significantly increasing the average accuracy among this subgroup from 60% to 70% (two-tailed paired t-test $p = 0.01$). In addition, all models trained on this run – without using any data from the prior run – were now significantly better at predicting their goals during the test phase than the models trained to a different player (74% versus 63%, $p = 0.03$).

Figure 4 summarizes the results of Experiment 2, comparing the five repeat players' results to their results from Experiment 1.

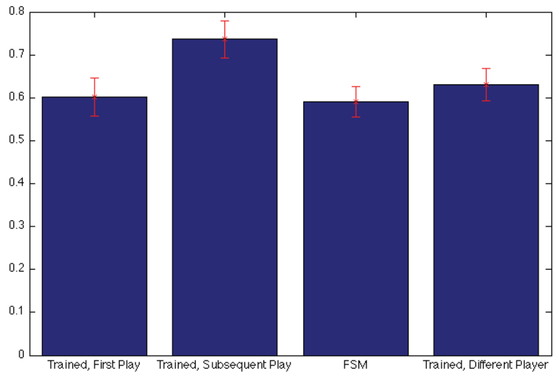

Figure 4: Average accuracies for the models trained on a subsequent playthrough, compared to models trained on a first playthrough, an FSM, and a model trained to a different player, for the five repeat players of Experiment 2. Bars are standard error. Chance performance is 0.33.

## Conclusions

The most interesting result here for the researcher interested in goal recognition is that training probabilistic models on players experienced with the game appears to have been much more useful than training the models on the subjects while they were still learning to play. This is true despite the fact that there was far more training data during the players' first run, which took them longer than the second run due to their lack of experience. If the training data had been kept from one run to the next and included in the second run, the machine learning researcher's most natural conclusion would have been that the increased amount of data was what led to the model's improved performance. Instead, data collected after the player has become acquainted with the game seems to be simply more reliable than the earlier data. This is interesting particularly because it was not even the players' high-level strategies and transitions between them that changed, but the low-level actions that they tended to take to achieve those same goals.

What this suggests for those creating intention recognition systems for video games – or indeed, machine learning researchers interested in goal recognition in general – is that it can be useful to give users some experience with novel tasks and systems before collecting training data. Or, if data is being collected passively in the aggregate, it may be useful to separate out the records of users who are new. Thus, even though it is appealing to imagine a game which learns the player even as the player learns the game, models may be more effective if they are trained on player missions that occur after the player has had a chance to perform similar tasks once or twice. It appears that at this point, the player's pattern of play becomes more understandable to the probabilistic model, and also more distinguishable from the patterns of an arbitrary experienced player. In short, it takes experience to develop a style.

There are many possibilities for future work with these

IOHMMs. A natural extension of the present work would be to learn the transition probabilities as well as the observation probabilities for a player model, using Expectation Maximization (Bishop 2006). It is an open question as to whether the hidden states learned from EM would still correspond to player motivations. It is entirely possible with EM that the hidden states will come to have an entirely different semantics. This sometimes occurs in speech recognition; though the textbook coverage of speech recognition often treats the hidden states as the phonemes to be recognized, in practice, EM can cause the hidden states to become uninterpretable by inspection. Controlling player goals in order to train the observation models may therefore still be useful as a component of a two-step process, where the second step trains the transition models either with EM or simple observation counts.

More broadly, other techniques for player intention recognition can fall out naturally from applying different algorithms to the IOHMMs. Categorizing players as one type or another, as in (Thue, Bulitko, and Spetch 2008), might be performed using model likelihood comparison. Constructing a single narrative that explains the player's behavior might be performed with the Viterbi algorithm. Artificial agents might register surprise when the prediction given by the Forward algorithm is later invalidated by the Forward-Backward algorithm, as they come to reinterpret the player's actions in light of new evidence. The IOHMMs themselves could grow more sophisticated as they condition on more game state evidence; introducing such evidence to the model is only as computationally expensive as procuring the evidence in the first place, since conditioning on known inputs does not increase the computational complexity of the Forward algorithm. Ideally, probabilistic methods could be used not just to help detect and gently guide players away from repetitive strategies, but to shape high-level stories from low-level actions.

## Acknowledgments

This work was supported by the Norma Wilentz Hess fellowship of Wellesley College.

## References

Albrecht, D.; Zukerman, I.; and Nicholson, A. E. 1998. Bayesian models for keyhole plan recognition in an adventure game. *User Modeling and User-Adapted Interaction* 8(1–2):5–47.

Bengio, Y., and Frasconi, P. 1995. An input output HMM architecture. In Tesauro, G.; Touretzky, D. S.; and Leen, T. K., eds., *Advances in Neural Information Processing Systems*, volume 7. MIT Press. 427–434.

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning*. New York: Springer.

Charniak, E., and Goldman, R. 1993. A Bayesian model of plan recognition. *Artificial Intelligence* 64(1).

Gold, K. 2010. Designer-driven intention recognition in an action-adventure game using Fast Forward Bayesian Models. In *FLAIRS 2010*. AAAI Press.

Gorniak, P., and Roy, D. 2005. Probabilistic grounding of situated speech using plan recognition and reference resolution. In *Proceedings of the Seventh International Conference on Multimodal Interfaces*. New York, NY: ACM.

Huber, M., and Hadley, T. 1997. Multiple roles, multiple teams, dynamic environment. In *Proceedings of the first international conference on autonomous agents*, 332–339.

Isla, D. 2005. Handling complexity in the Halo 2 AI. In *Proceedings of the Game Developers Conference*. Gamasutra.

Laviers, K.; Sukthankar, G.; Molineaux, M.; and Aha, D. W. 2009. Improving offensive performance through opponent modeling. In *AIIDE 2009*. AAAI Press.

Lee, H.-K., and Kim, J. H. 1999. An HMM-based threshold model approach for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21(10):961–973.

Manning, C. D., and Schütze, H. 1999. *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press.

Mateas, M., and Stern, A. 2007. Writing façade: A case study in procedural authorship. In Harrigan, P., and Wardrip-Fruin, N., eds., *Second Person: Role-Playing and Story in Games and Playable Media*. Cambridge, MA: MIT Press. 183–208.

Mott, B.; Lee, S.; and Lester, J. 2006. Probabilistic goal recognition in interactive narrative environments. In *Proceedings of AAAI-06*. Menlo Park, CA: AAAI Press.

Orkin, J., and Roy, D. 2007. The Restaurant Game: Learning social behavior and language from thousands of players online. *Journal of Game Development* 3(1):39–60.

Rabiner, L. R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* 77(2):257–296.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, New Jersey: Prentice Hall, 2nd edition.

Starner, T., and Pentland, A. 1997. RealTime American Sign Language Recognition from Video Using Hidden Markov Models. Technical Report 375, M.I.T Media Laboratory Perceptual Computing Section.

Sukthankar, G., and Sycara, K. 2005. Automatic recognition of human team behaviors. In *Proceedings of the IJCAI Workshop on Modeling Others from Observations*.

Thue, D.; Bulitko, V.; and Spetch, M. 2008. Player modeling for interactive storytelling: a practical approach. In Rabin, S., ed., *AI Game Programming Wisdom 4*. Boston, MA: Course Technology. 633–646.