Proceedings of the Twenty-Third International Florida Artificial Intelligence Research Society Conference (FLAIRS 2010)

# Designer-Driven Intention Recognition in an Action-Adventure Game Using Fast Forward Bayesian Models

**Kevin Gold**
Wellesley College

### Abstract

A method is described for quickly inferring a player's goals from low-level inputs in an action-adventure game. Fast Forward Bayesian Models (FFBMs) use the very efficient forward algorithm normally used in the Forward-Backward algorithm, but they use transition matrices and observation functions specific to the game's current state. In experiments with both a veteran player and novice players using a Flash action adventure game, the algorithm is faster to correctly infer whether the player is attempting to kill monsters repeatedly, explore, or return to town than a finite state machine (FSM) that must wait for less ambiguous information, yet is also more robust against momentary deviations from goal-relevant behavior. Using a constant transition matrix that does not increase the probability of out-of-state transitions on finishing a goal erases the advantage over the FSM.

## Introduction

Two facts about AI in the game industry must inform any choice of algorithm. First, the algorithm must be fast, because the graphics team will take the lion's share of the cycles. Second, the algorithm must allow game designers, who do not necessarily know AI, strong control over its operation. The rationality of the AI should always be secondary to the player's experience of the game, and this means game designers must ultimately have veto power over the AI programmer in terms of which algorithms get used. This means game designers are likely to reject any algorithm that results in a "black box" AI, in favor of a more predictable system.

These two factors have historically made finite state machines (FSMs) or "behavior trees" (Isla 2005) the algorithms of choice in industry-created AIs. These algorithms are fast because they require no lookahead and no memory beyond simple state information, sometimes augmented by a stack of previous goals. They also make it easy for game designers to state the behaviors of the agents in natural language. This kind of strong control over the AI's operation is commonly overlooked by academic AI researchers, though there has been some research into making subsumption or behavior-based control equally accessible (Heckel, Youngblood, and Hale 2009).

But FSMs and behavior trees deal poorly with chaotic inputs over time. A single action, no matter how inadvertent, can trigger the same string of behaviors as a more consistent pattern over time. This makes picking up on subtle patterns of player actions over time more difficult; an FSM must have some arbitrary threshold – step here, keep acting for X seconds – that players can easily circumvent once they discover the rule. This predictability may sometimes be desirable for gameplay reasons, but not always.

This paper will argue that the game industry should add another fast, designer-tunable AI methodology to its toolbox: probabilistic reasoning over time. Probabilistic reasoning is fast: the "forward algorithm" (Russell and Norvig 2003) has the advantage of incorporating all the evidence observed by the AI since it began operation, yet requires only a constant time update relative to the time it has been observing. It essentially only requires a single matrix operation, a vector multiplication, and a scaling operation; everything else is optional "perception" that is faced by the FSM designer as well. Moreover, it is designer tunable: the designer can state in natural language what states the system should be detecting, how they will appear to the system, and how they transition between one another, and leave them to the AI designer to implement. For example, the designer may state, "The player tends to head back to town when the player's inventory is full," and this translates naturally into a transition matrix that takes this heuristic into account.

The application that will be presented in this particular paper is player intention recognition, specifically for an action-adventure setting which includes both exploration and incentives to kill enemies. A common phenomenon in these settings is that the player is trying to gain levels or treasure in a repetitive fashion by killing monsters in the same area over and over – a behavior known as "leveling" or "grinding." Grinding is undesirable from the designer's point of view because the player is engaging in a task that the player probably finds tedious, yet the player believes that the game's rewards have been structured to make it the most effective way to succeed. If a game could detect "grinding," it could shift the reward structure slightly so that the player is more likely to have a varied, fun experience. On the other end of the spectrum, the game designer may wish to reward the

player for attempting to "explore," or trying to encounter new content, even if the player is not certain how to reach new areas.

Grinding and exploring are examples of behaviors that are defined by the player's actions over time, and not easily amenable to FSM or behavior tree descriptions. Killing a monster does not necessarily signal an intention to grind, since the monster may simply be in the way of exploration. A player going over old ground may actually be exploring, but confused about how to reach a new area. These kinds of issues can be handled in an ad hoc way by adding timers to FSM conditions, but the decision between behaviors is best done by comparing the likelihoods of each known behavior. In this way, even behaviors that do not particularly fit the designer's preconceptions can still be classified, yet the probabilities produced will indicate the system's uncertainty. Very obvious grinding could thereby get a very obvious penalty, but if the system is uncertain, it could reduce or dispense with the penalty.

This paper is the first to suggest the forward algorithm as a useful tool for game industry AI, though player modeling, plan recognition, and Bayesian networks in general have precedents in the literature. Typically, player plan recognition has been applied to planning-like domains in which the actions are discrete, "large" actions that are semantically meaningful, such as pushing levers or sitting down to eat. These have included probabilistic context-free grammars that use game context to disambiguate speech, and thereby infer the player's plan (Gorniak and Roy 2005); a game that used a large corpus of person-to-person interactions in a restaurant scenario to generate a plan network that could be used to guess or produce next behaviors (Orkin and Roy 2007), and a dynamic Bayesian network trained on a corpus of actions in a multi-user dungeon (MUD) game to predict the player's current quest, next location, and next action (Albrecht, Zukerman, and Nicholson 1998). The present work differs from all of these in that it is intended for an action-heavy game. This means the player actions are likely to be low-level and highly ambiguous (moving left or swinging a sword), the player must often deviate from the plan briefly in order to avoid danger (moving away from the goal to get away from the monster), memorizing chains of these small actions is likely to be unreasonable, and processing speed must be very fast. Of these, the MUD dynamic Bayesian network of (Albrecht, Zukerman, and Nicholson 1998) was most similar to the present work, but did not address the issue of changing goals, assuming that the quest was fixed over the length of the network; the experiment below shall demonstrate that there is some subtlety involved in designing the network so that it can rapidly adapt to a new plan. The Forward-Backward algorithm from which the forward part is lifted here is more common in speech recognition (Rabiner 1989) and gesture or action recognition (Lee and Kim 1999), two other domains with a high degree of noise and ambiguity that must nevertheless often function in real time.

The next section will describe the theoretical principles behind the forward algorithm and how it can actually operate on much more complex information than a typical Hidden Markov Model (HMM) when performing player intention recognition in a game, because *all the information is known besides what the player will do next*. In short, this allows very sophisticated Bayesian models to act as HMMs for the purpose of the forward algorithm, because all the transitions and emissions are conditioned on known information. (It will also be explained why only the "forward" part of the more widely used Forward-Backward algorithm is actually necessary for our purposes here.) An experiment will then be presented in which a simple action-adventure Flash game decides between three behaviors – "leveling," "exploring," and "going to town" – and its performance is compared to a finite state machine designed to detect the same things. In the discussion section, further applications of dynamic Bayesian models for games will be discussed, including enemy AI that deals intelligently with missing information, trainable player models, and adjusting to play style.

## Fast Forward Bayesian Models

The goal of a "dynamic Bayesian model" for player behavior is to provide at every moment, for each possible high-level description of player behavior, a probability that the player is currently engaging in that kind of behavior. At each stage, the model used in the experiment will be used as an example.

### The required input from the designer

To design a probabilistic model for player behavior over time, the AI programmer must know four things: What are the different states we think the player might be in? What kinds of actions do we expect in each of these states? How often does the player naturally switch between these states? And, what events might make switching between states more likely?

For our example Flash game, the designer may state that we are interested in deciding between three behaviors: "grinding" (running around killing monsters), "exploring" (moving to new areas of the map), and "heading back to town." When killing monsters, the player is likely to either approach monsters and attack, or wait for monsters to approach and then attack; if there are no monsters nearby, the player will move to find some or wait for some to appear. When exploring, the player is likely to move in the direction of unexplored map squares. When returning to town, the player is likely to move in a known path back to town.

This specification can then be handed to the AI programmer for implementation, freeing the designer to create rules about how to react to the probabilities of behavior that will be produced by the system.

### Translating designer-specified rules into a fast dynamic Bayesian model

The forward algorithm is designed to function on a Hidden Markov Model (HMM), so it is useful to start with the HMM as a base model from which our more complex (but still fast) model can be derived. Informally, an HMM consists of a hidden state at each time step that must be inferred, an observation caused by that state at each time step that gives a

hint as to the underlying state, and rules for how states transition between each other and produce observations. The states here will be player intentions, and the observations will be player inputs.

Formally, a Hidden Markov Model consists of a transition matrix $T$ and an observation matrix $O$, both containing probabilities for each entry. The matrix $T$ specifies the probability at each moment in time $t$ that if the state is $i$, the next state will be $j$: $T_{ij} = P(s_{t+1} = j | s_t = i)$. This matrix will typically have large values along the diagonal entries $T_{ii}$, indicating a high probability of remaining in the same state, and low probabilities elsewhere. Given no other input from the designer, it is reasonable to assume that the total probability of transitioning from state $i$ to another state is $(1/timeInState_i)$, where $timeInState_i$ is an estimate of how long the player tends to stay in state $i$. This makes the diagonal entries $T_{ii} = 1 - (1/timeInState_i)$ and the other entries set to values that cause the rows to sum to 1.

A variation from the Hidden Markov Model that is still fast for player intention recognition is to have different matrices $T$ for different conditions: one matrix for the case of the player being low on health that gives a high probability of transitioning to "return to town," another that gives high probability of ceasing to explore when the player's map is complete, and so on. As long as the relevant information is known to the system at each time step, it will still be fast and efficient to update the system's beliefs about the player's intentions. At the very least, the probability of transition to other states should increase when the player accomplishes a goal, to account for the fact that the player is more likely to switch modes of play once the current goal has been achieved. In the current example, on obtaining a treasure chest, the transition probabilities from "explore" to all 3 states were set to be equal (= 0.33), and while in town, the chance of the goal being "return to town" was set to 0 and transitions to the other two goals were equally likely (=0.5).

A Hidden Markov Model normally also possesses an observation matrix $O$, giving the probability of each behavior given the underlying hidden state: $O_{ij}$ = P(system observes behavior $j$ | player is in state $i$). However, the behaviors need not form a discrete set for the Bayesian model's update to be fast; any function $O(i, j, \vec{x})$ that gives a probability of behavior $j$ in state $i$ will work fine, as long as the vector of other relevant variables $\vec{x}$ is known. In general, however, the observation probabilities for a given state must sum to 1, and the number of states among which the probability is being divided should be the same for each hidden state. Failure to obey either of these principles can result in a model that unfairly favors one state.

For the example game, it was assumed that the player would stay in each state for roughly 100 updates on average, giving $T_{ii} = 0.99$ and $T_{ij} = 0.005$ for $i \neq j$. The observation probability under "grinding" was 0.8 for swinging a sword while still, with the remaining probability mass divided equally among other actions, unless there were no monsters in the room. In the case of exploration, returning to town, and grinding with no monsters present, the observation probabilities were calculated by assigning weights to the 8 directions of movement based on which map edges were acceptable based on the state, with a stronger weight given to the closest acceptable map edge. A map edge was acceptable for "explore" if it led to an unvisited map square, was acceptable to "town" if it moved closer to town, and was always acceptable to "grinding." Weights for all directions began at 1, were increased by 4 for the closest acceptable direction, were increased by 2 for diagonals adjacent to the acceptable direction or other acceptable directions, and were increased by 1 for diagonals adjacent to non-preferred acceptable directions. These were then normalized to produce estimated probabilities.

It is important for these models that the states from time $t$ to time $t + 1$ are relatively independent. If the calculations to be described below are performed every frame, or every few frames, it can make the system jump to conclusions too quickly; if a player is still headed east 200 ms after the last poll, it is unlikely that this event is independent from the last check for the player's action, and so these should not be treated as two separate pieces of evidence in favor of a particular interpretation. Therefore, the discrete times here where measurements are made are either when the player's input changes, or when the situation for the player significantly changes (in this example, on an area change). A flag can be set in the event handler for keyboard or mouse input that detects whether the player's direction of movement or action has changed, or when the player enters a new area, and these are the times $t_i$ that are being discussed here.

The resulting model, with varying transition matrices $T$ and an observation function $O$ that both depend on the circumstances, is not technically a Hidden Markov Model but a dynamic Bayesian model; but it will still function with the very efficient forward algorithm for HMMs as long as all the relevant information is known. For lack of a better term, we can call models that obey this property "fast forward Bayesian models" (FFBMs).

## Updating beliefs in constant time

It is a very large advantage in video games over other real time systems that essentially all information is known besides what the player is going to do next. This allows the system to condition a very complex model on many variables, yet retain the constant time update speed of the forward algorithm.

Figure 1 shows the current model as a graphical model. Both the observation probabilities and transition probabilities can depend on an arbitrarily complex "game conditions" vector $\vec{x}$; so long as all of the information in this state is known, the algorithm is as fast as the standard forward algorithm.

Let the current vector of probabilities for the hidden states be $\vec{S}_t$, and $\vec{x}$ be a state vector representing the current game state. Then on receiving a new input from the player $o_{t+1}$, the algorithm for updating $\vec{S}$ is:

$\vec{S}_{t+1} = T(\vec{x})^T \vec{S}_t$
**for** each hidden state $i$ **do**
  $s_{i,t+1} \leftarrow s_{i,t+1} * P(o_{t+1} | \vec{s}_{i,t+1}, \vec{x})$
**end for**
$\vec{S}_{t+1} \leftarrow$ normalize($\vec{S}_{t+1}$)

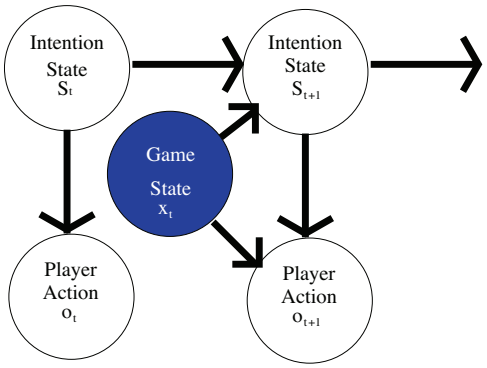


Figure 1: A slice of the graphical model for a fast forward Bayesian model. Even though the transitions and observations are conditionally dependent on the game state, when this information is known, conditioning on it reduces the model to the same calculations as a Hidden Markov Model.

A single matrix multiplication by the current transition matrix $T(\vec{x})$ (dictated by the game state $\vec{x}$) prepares the state probability for the new information in the absence of new input. When a new observation is received, Bayes' rule dictates that $P(\vec{S}_{t+1}|o_{t+1}, \vec{x}) \propto P(o_{t+1}|\vec{S}_{t+1}, \vec{x}) * P(\vec{S}_{t+1})$. The hidden state probabilities then are normalized to sum to 1. Because $\vec{x}$ is known, the observations and transitions can simply be conditioned on $\vec{x}$ (i.e., $T$ and $O$ are chosen appropriately) without further probabilistic computations.

If it matters what the player's intention was at some point in the past $h$, where $1 < h < t$, the Forward-Backward algorithm (Rabiner 1989; Russell and Norvig 2003) should be used to calculate the state probabilities instead, because the observations at times $h \ldots t$ can and should influence the current belief about what the state was at that point in the past. However, the Forward-Backward algorithm is not necessary to calculate the player's most likely *current* state, because the backward part of the Forward-Backward algorithm only affects the inferred hidden state probabilities in the past.

Just as with Hidden Markov Models, the forward algorithm here is calculating the probability of each current state given *all* the evidence seen so far, $P(\vec{S}_{t+1}|o_{1\ldots t+1}, \vec{x}_{1\ldots t+1})$. Even though the model is making only constant time updates, given the Markov assumption of conditional independence, the current beliefs fully take into account all evidence seen so far. This is very different from a finite state machine, which has a state that depends *only* on the current inputs $(o_{t+1}, \vec{x}_{t+1})$.

## Experiment

### Methods

The method was implemented in a Flash game borrowing sprites from *The Legend of Zelda*, running at 30 fps. A screenshot of the game in "debug mode" is shown in Figure 2. The main character could walk in 8 directions, swing a sword to attack, or walk to a map edge without a wall to travel to the next area. The areas were arranged in a 4x5

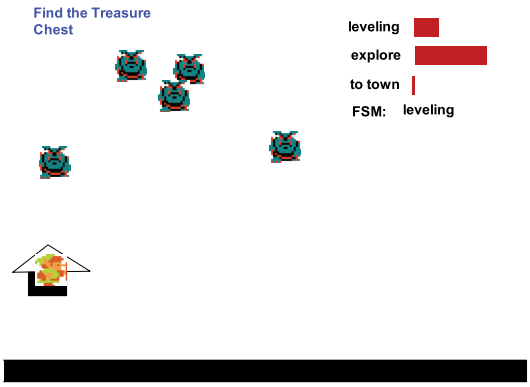


Figure 2: A screenshot of the experiment in "debug" mode, where the FFBM probabilities and FSM prediction are shown. The avatar and town are in the lower left, and a wall blocks progress to the south.

map grid, with the player's starting location, the town, at (2,2). Only 13 of the 20 areas were reachable given the walls, making a small maze. A treasure chest was randomly located at one of (1,4), (4,4), (4,1), or (5,3). Each area contained 5 monsters (for the experienced player) or 3 monsters (for the new players), randomly placed, that would move in non-diagonal lines to approach the player at half the player's speed. These respawned on entering an area, or after a 150 frame (5s) timer.

Each run consisted of 3 phases in which the player was given a specific goal: "Find the treasure chest," "Return to town," or "Kill 10 monsters." This was to establish "ground truth" for the goals – none of the algorithms received information about what the current goal was. The goal became "return to town" on picking up the treasure chest, and "kill 10 monsters" on reaching town after having retrieved the chest. Running into a monster resulted in the player's avatar being returned to town during the first and third phases, and resulted in being returned to the treasure chest location during the "Return to town" phase. Swinging a sword killed a monster in one hit. The player could move fast enough that monsters could usually, but not always, be avoided entirely in moving from one area to the next if the player took a circuitous route. The 10 monster goal was chosen so that the player would need to change map areas at least once to finish the goal quickly, or wait for the monsters to respawn.

The FFBM algorithm described above was compared to two alternatives, which calculated predictions simultaneously with the algorithm. The finite state machine was a simple predictor that obeyed the following rules: start with a random goal prediction; transition to "grinding" when the player attacks a monster; transition to "explore" when moving to a map area that has not previously been explored; transition to "town" when moving to an area that is closer to town than the current square. The unconditional Bayesian predictor was identical to the algorithm described above, but the transition matrix $T$ always used the default probabilities,

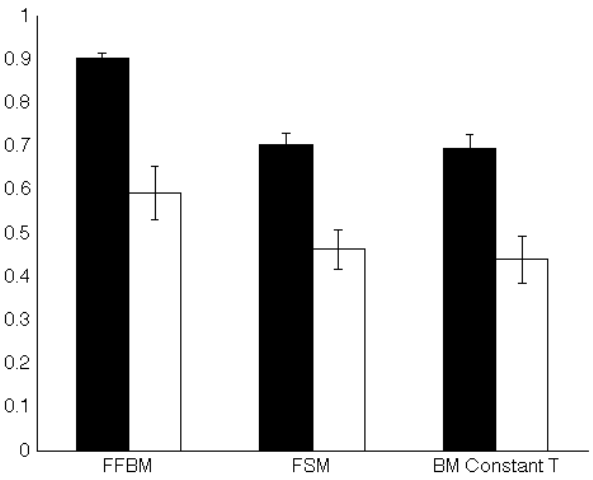


Figure 3: Percent of time slices in which each method correctly inferred the player's current goal for an experienced player over 10 trials (black) and for 10 different first-time players (white).

and did not increase the probability of changing state on returning to town or picking up the treasure, making it more similar to a standard HMM.

The three algorithms' predictions were compared to the ground truth at every update of the FFBM, and accuracy was calculated at the end. The experiment was repeated under 2 conditions: 10 runs with the same experienced player (the experimenter), and 10 times with a different subject for each run who had no experience with the game. No parameters were changed in response to the data and no training took place; the hand-chosen parameters were kept for both conditions and all runs.

## Results

Figure 3 shows the performance of the three algorithms. For the experienced player, at 90% accuracy, the FFBM model performed significantly better than either the finite state machine or the non-conditional Bayesian model ($p < 0.0003$ in each case, two-tailed paired t-test), both of which showed 70% accuracy. The performance of the HMM-like model that used the same transition matrix $T$ across all time was not significantly different from that of the finite state machine ($p = 0.88$). The algorithm was worse overall at predicting the behavior of new players (59% accuracy), but was still significantly better than either the FSM (46%, $p < 0.03$) or the version with constant transition matrix (44%, $p < 0.0002$), which were not significantly different from each other ($p = 0.66$).

Figure 4 helps to explain the importance of using a transition matrix conditioned on the game state by showing the probabilities for the FFBM model and the same-T model over the 20 time steps surrounding the transition from "return to town" to "leveling." Though the Bayesian model with a constant transition matrix has a smoother transition of probabilities, this results in a slower inference of the player's state, where the state is only clear once the player starts attacking. The FFBM, on the other hand, can quickly transition because the transition probabilities no longer favor the current state on reaching town, and thus the "momentum" does not keep the state in "return to town" after the town has been reached. (Recall that the new transition probabilities were *not* chosen to favor the goal that was next in the experiment; they merely ceased to favor the current state.) On the other hand, both Bayesian models resisted the quick FSM state changes produced by an attacking player who was still exploring – without a signal that the state might change, the bump in P(leveling) caused by an exploring player attacking one or two monsters in the way was usually not enough to overcome the momentum achieved by the previous observations.

## Conclusions

Bayesian models are an underutilized method in modern game AI. They are very fast to update and can incorporate more subtle cues than finite state machines, and in a principled way. Very complex reasoning over time can take place very quickly because all of the relevant information is known to the game besides the player's state, allowing the probabilistic calculations to condition on many different factors without slowing down the forward algorithm that makes HMMs so fast. Notice, however, that the models here are slightly more sophisticated than HMMs in that neither the transition matrix $T$ nor the observation model $O$ are constant over time, but can vary depending on the situation. In fact, the experiment above shows that a finite state machine produces comparable performance to a Bayesian model that does not increase the transition probabilities when goals are met, because by the time the HMM-like model overcomes its earlier momentum, the player has probably already produced some unambiguous behavior that renders more complex inference moot.

This paper did not explore training the model based on player actions; the models here were hand-designed, but it should be possible to use data where there exists some ground truth as to player intention in order to tailor the probabilities to the player's actions. It would be worthwhile to run additional experiments to see how sensitive these kinds of models are to their parameters. It is reasonable to hypothesize that such models should be relatively insensitive to their parameters as long as the player goals are relatively distinguishable. The observation probabilities do not particularly have to fit the truth; they merely have to ensure that the player's behavior fits the correct inference better than the incorrect inference.

The use of Bayesian models over time to infer player goals could possibly be extended to decision-making for enemy AI's with limited information, since Bayesian models deal well with inference when variables are missing. The main obstacle would be ensuring that inference remained fast, since the forward algorithm could no longer be used. Inference over polytrees (Pearl 1988) with missing information is therefore one potential future direction for this work.

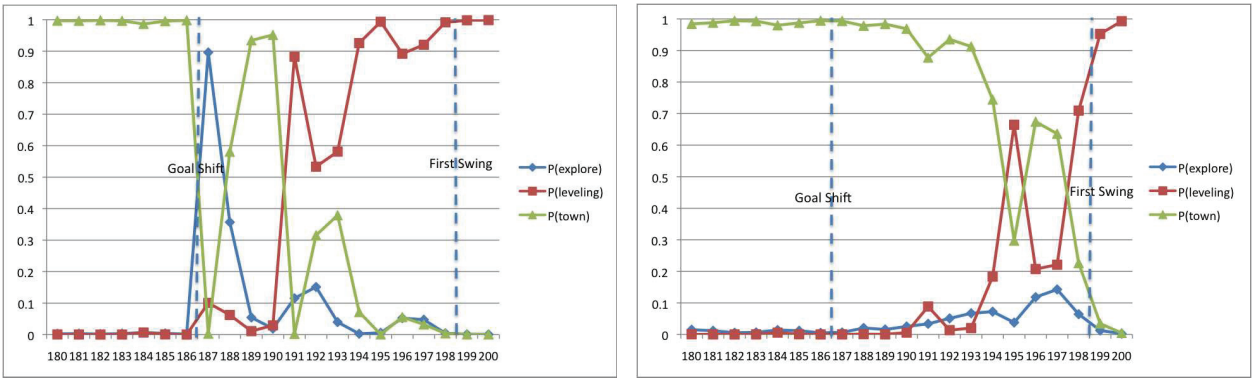


Figure 4: The change in inferred goal probabilities surrounding the goal change on reaching town in a sample run. The time axis is measured in player button presses and releases. Left: The FFBM model quickly realigns itself to the new evidence before the player has even taken the first swing. Right: With constant $T$, the model is smoother but slower to adapt to the player's new behavior, and only barely manages to infer the new state before the FSM switches at the player's first attack.

Another possible extension is creating multiple FFBMs that use rules describing different play styles. Updating the likelihood of each FFBM requires nothing more complicated than the forward model, and therefore beliefs about the player's overall play style could also be updated in constant time. Though others have addressed systems based on assigning points to different play styles based on the player's actions (Thue, Bulitko, and Spetch 2008), dynamic Bayesian models offer both a principled way of choosing these constants – namely, by estimating the probabilities that the player will perform them – and has a much better chance of being moved into continuous, action-oriented domains, since AI designers could choose parameters for continuous probability distributions rather than dealing with a mess of hand-chosen constants.

In short, FFBMs are an exciting potential avenue for growth in game AI. The graphics team should be able to spare the single matrix multiplication and dot product necessary for the update, and game designers should welcome them because describing play styles and strategies based on the current situation is something that they can both easily describe and take an interest in. Moreover, game designers often have a fairly good intuitive grasp of probability from experience with tabletop games, and so the transition and observation models are concepts that should be a relatively easy sell, even if the underlying Bayesian forward model that makes use of the models is unfamiliar. This paper is a first step in what is a potentially very powerful tool for fast, designer-specified game AI.

## Acknowledgments

The author would like to thank Norma Wilentz Hess and Wellesley College for supporting this work.

## 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.

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.

Heckel, F. W. P.; Youngblood, G. M.; and Hale, D. H. 2009. Behaviorshop: An intuitive interface for interactive character design. In *Proceedings of the Fifth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI Press.

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

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.

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.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann.

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.

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.