Gillies, Marco

Learning Finite State Machine Controllers from Motion Capture Data

1

Learning Finite State Machine Controllers from Motion Capture Data

Marco Gillies

July 30, 2009

Abstract

With characters in computer games and interactive media increasingly being based on real actors, the individuality of an actor's performance should not only be reflected in the appearance and animation of the character but also in the Artificial Intelligence that governs the character's behavior and interactions with the environment. Machine learning methods applied to motion capture data provide a way of doing this. This paper presents a method for learning the parameters of a Finite State Machine controller. The method learns both the transition probabilities of the Finite State Machine and also how to select animations based on the current state.

1 Introduction

In computer games and other interactive media it is increasingly common to want to use characters that represent real actors. One reason is simply that many games are now based on films and therefore the main characters in the game should be based on the actors in those films. There is also a more general reason. Actors can bring great psychological insights to their performances, thus ensuring a subtlety and nuance of character that is difficult to produce in interactive media. Therefore the more games and interactive media seek to create complex and rounded characters the more real actors will inevitably be used. As games move from generic types of character actions to more complex and socially meaningful forms of interaction, it is not enough for characters to graphically resemble actors, their behavior must also be the same. This has two components; the character's animation should represent how the actor moves but also the character's Artificial Intelligence. The way that it selects actions should represent how the actor responds to events and chooses how to behave, in other words.

Data driven methods in animation, using data from motion captured human movement to animate characters, are able to reproduce the movements of a particular individual, including mannerisms and style that are vital to expressive characters. Artificial Intelligence (AI) in games, however, has generally been approached procedurally, focusing on algorithms rather than data. This method has produced some excellent results but it is not well suited to recreating the behavior of an actor. AI algorithms typically have a number of parameters that can be tweaked to create different styles of behavior, but there is no principled way of determining which set of parameters will correspond to the behavior of the actor. As this behavior is often highly nuanced, it can be very difficult to determine manually what the key features of the behavior are. Even if it were possible it would require a painstaking analysis of the actor's performance. What is needed is an automated method of extracting the AI parameters directly from the performance.

The aim of this paper is to automatically learn which parameters of a character AI system correspond to an actor's performance. We do this by learning the parameters from motion capture data of a performance. This work integrates state of the art machine learning methods with existing game AI systems. We have chosen Finite State Machines (FSM) as our example of an AI system. Though FSMs are a fairly simple system they are flexible and are very widely used in existing video games. This means that the methods presented here could be easily integrated within existing game engines and content pipelines. The main technical contribution of the paper is a method for learning a probabilistic model that determines the transitions probabilities of the FSM. For this we use a learning method called Input-Ouput Hidden Markov Models [1], a form of Dynamic Bayesian Network, in combination with other learning techniques such as clustering and Principal Component Analysis. Our method also determines how the FSM controls the lower level animation algorithms. This is necessary as the relationship between the FSM states and the character's behaviour is learned automatically, and so a control policy can only be defined during the learning process. For animation we use a method called Motion Graphs [2–5]. By pre-computing the outputs of the Finite State Machine we are able to learn an efficient control strategy for the Motion Graph based on the state of the FSM. For this we use a dynamic programming method based on Reinforcement Learning.

[*]M. Gillies is with the Department of Computing, Goldsmiths, University of London, New Cross, London SE14 6NW, UK e-mail: (m.gillies@gold.ac.uk).

2 Related Work

2.1 Intelligent Behavior

Numerous techniques have been used to simulate the behavior of character for games and interactive media, such as the use of Finite State Machines or of search algorithms such as A* for navigation. Chiu et al [6], like us, used Finite State Machines in combination with Motion Graphs, however, the relationship between states and motion in their system was hand annotated rather than learned. Academic researchers have proposed other methods as diverse as logic based reasoning [7], planning [8], agent based AI [9] and machine learning (discussed in the next section). One particularly active area of research of intelligent behavior has been non-verbal communication and emotional expression [10]. There has also been considerable work on the behavior of large crowds, such as Pelechano *et al.*'s social forces model [11] or Lee *et al.*'s data driven method [12]. Our work, however, focuses on detailed behavior of individuals.

2.2 Machine Learning for Character AI

Recent years have seen an interest in the use of machine learning on motion capture data for the control of characters in games and virtual environments. Examples have included learning styles of motion [13, 14], controlling walking based on a path [15] and simplified user interfaces [4]. Kipp et al. [16] have presented a machine learning method for reproducing the gestural style of particular individuals. They took a somewhat different approach, using manual annotation of video sequences rather than motion capture, to create their data sets.

The work presented in this paper includes two types of machine learning method. The first, Input-Output Hidden Markov Models [1], is an example of Dynamic Bayesian Networks, a model used by Pelachaud and Poggi for controlling expressive characters [17]. However, they used a priori rather than learned probabilities. Shum et al. [18] used an architecture similar to ours in which a finite state machine was used to control a Motion Graph for two character interaction, while Kwon et al. [19] used a Dynamic Bayesian Network, without hidden states, to control a motion graph, again for two character interaction. Both of these example used clustering on the motion data to determine the states of their Finite State Machines. This clustering was similar to the method we use to obtain actions, however, both identified the state of the character entirely with the action. This is reasonable in the case of two character interaction as the actions of both characters, taken together, are sufficient to specify the state of the interaction (though it has no history). However, separating state and action, as we do, creates a more general model that can be used for other applications. The other aspect of our work is the use of dynamic programming techniques from Reinforcement Learning [20]. Blumberg *et al.* [21] used reinforcement learning for an animated dog, the user "trained" the character by rewarding correct behavior and punishing wrong behavior. This is very well suited to their virtual dog character, but not for recreating the behavior of an actor. Lee and Lee [22] used reinforcement learning to train a virtual boxer, giving rewards based on how close a punch is to the desired position. Similarly McCann and Pollard [23] used similar methods for training a character that can produce realistic responses to a navigation control signal from a user, while Treuille, Lee and Popović [24] used reinforcement learning for autonomous navigation behavior. These methods generally used a well defined reward function based on how well a character performed an action. We, on the other hand are dealing with behaviors for which it is difficult to define quantifiable score, rather we are attempting to create behavior similar to an existing performance. The method we present in this paper builds on the work of Gillies et al. [25].

2.3 Data Driven Animation

A feature of this research is that we automatically learn a policy that allows the AI level to control the animation of the character. This combination of learning and animation is based on research in data driven animation. Data driven animation methods consist of taking motion capture data and using it to animate a character. While this can give an exact reproduction of an individual's behavior it can be inflexible. Basic data driven animation can only play back animation that has already been captured and cannot adapt to new situations. For this reason most research in data driven animation has focused on transforming existing motion data in order to adapt it to new circumstances. A wide range of methods has been used, ranging from interpolating multiple motions [26] to using physical simulation to alter the details of a motion [27]. The range of problems that have been tackled is also large, for example, changing the style of a motion [13, 14] or adding positional constraints [28, 29]. In this paper, the approach taken is to generate new animation by sequencing together motion clips from a large data base. The most common method for this, are graph based methods [2–5], and we use a method of this type.

3 Method

In order to illustrate the method we will use a running example throughout the next sections. In the example we are creating a character that plays the part of a sentry. Events such as shots, explosions etc., can occur in the environment either in front or to the left and right of the character. The character will respond to these events in some way.

The control of our characters is based on a probabilistic form of Finite State Machine (FSM). The FSM consists of a set of possible states that the character could be in: $S = \{s_0 \dots s_n\}$. There is also a set of events that the character can receive and respond to: $E = \{e_0 \dots e_m\}$. The events correspond to things happening in the environment and in our examples these include events as diverse as explosions and human speech. On receiving an event the character can transition from its current state, s_i to a new state s_j (where j can be equal to i). Thus in our example, on receiving an event corresponding to a noise to the left, the sentry might enter a new state in which he is more vigilant in that direction. These transitions are probabilistic, and the probability distribution for the new state, $P(s_j|s_i, e)$, depends on the current state, i, and the event, e. This commonly used control architecture allows the character to have complex and long term responses to events, in which an event can alter all future behavior.

Our characters are animated using a Motion Graph [2–5], a data structure that allows new animations to be generated by resequencing a data set of existing animations. It is a graph in which edges correspond to animation clips and nodes are choice points at which a set of different clips can be selected. We present an algorithm in which edges are selected based on the current state of the FSM. Since the structure of the motion graph is known in advance the probability of each edge of the graph, given each FSM state, can be pre-computed, resulting in a very fast run time method for controlling the animation.

The character can respond to events in real time. The real time control of the character consists of two parallel processes. The first process handles events using the FSM. As new events occur the FSM is triggered to transition to a new state. We define a special event e_0 that is triggered in a time step in which no other event occurs. The second process generates animation using the motion graph. Whenever the current clips comes to an end a new clip is selected based on the current state of the FSM. For example, a sound to the left would cause an event, which would be sent to the sentry and might result in a new state, for example being vigilant toward the left. When the current motion clip ends the choice of the next clip would use a different probability distribution dependent on this state. It might, for example, give higher probability to movements that appear to look to the left.

The main contribution of this paper is a method for learning the parameters of our FSM from a set of motion capture data. We use a probabilistic learning model called an Input-Output Hidden Markov Model (I-O HMM) [1] that is equivalent to our FSM model. The learnt I-O HMM is used in two ways, firstly to learn the transition probabilities for the FSM and secondly to learn the function that selects edges in the Motion Graph based on the current FSM state. The next sections will describe the method.

3.1 The Capture Process and Data Handling

The first step in our learning method is to create a data set of an actor responding to a set of events. This data set consists of motion capture data together with some representation of events. We motion capture an actor responding to the events. In all our experiments the events were presented to the actor as audio signals. In our sentry example the actor was given a soundtrack consisting of a number of loud crashes with variable periods of silence in between. In the other experiments the exact nature of these signals varies and will be described in detail for each example in section 4. All that needs to be said here is that the actor could clearly hear the different events and alter his or her behavior accordingly. There is nothing in our method that constrains the events to be auditory, in fact many types of events might be better suited to being presented visually on a screen while the actor performs. The actor was asked to respond to the audio events, for example the actor playing the sentry would respond to the crashes by looking in the direction of the sound. Eventually, if a long enough period of silence occurred the actor would return to scanning the environment in a more general way.

After the motion capture session the resulting data is pre-processed to obtain a combined data set of events and motion data in a form that is suitable for learning. As the sets of events are quite diverse the exact handling of them varied between different examples and will be described individually in section 4. For the sentry example the audio volume on the left and right channels was thresholded to detect events on either the left or right of the actor. As the sound lasts about a second, the result is that the event occurs repeatedly over a number of time steps, rather than a single instantaneous event. This makes it easier to model delayed responses, as described below. The end result in all cases is a sequence of discrete events.

We also discretize the motion data, using a clustering algorithm, so that it consists of a sequence of discrete poses. This discretization step is needed so that our data is in a suitable form for our machine learning method. While it is possible to model continuous and multivariate data using an IO-HMM, these models are generally not suitable for our data. Firstly they usually require more data to learn (having more parameters). More importantly most commonly used continuous models, for example Gaussian models, are poor representations of human motion data. The large space of possible joint configurations is very sparsely occupied, only a small set of poses occur commonly. However, these poses can be very diverse, existing in very distinct areas of joint configuration space. For this reason movement is best represented using a distribution of discrete clusters, rather than a continuous Gaussian distribution over the whole space.

The aim of the clustering algorithm was to reduce the motion data to a representation consisting of a discrete set of possible "actions". In our method it is very important that this discrete representation depends not only on the static pose but also the instantaneous velocities at the pose. This allows us to distinguish different types of movement that may share similar poses. We therefore define actions to consist of a pose with an associated velocity component. The initial representation of our data is the standard animation format: a sequence of frames, each of which consists of quaternion joint rotations. The frame at time t takes the form: $m(t) = \{p_0, q_0, q_1 \dots q_n\}$ where p_0 and q_0 are the position and orientation of the root (the overall position of the character) and the remaining q_i are the rotations of the joints relative to their parents (e.g. the rotation of the elbow relative to the upper arm). The quaternion representation of rotation exists in a spherical space S^3; and so is not suitable for the linear algebra methods we use. We therefore convert to the exponential map representation [30]. This representation consists of projecting quaternions to a linear subspace that is tangent to the spherical quaternion representation. The best results are achieved if we choose the tangent space at the average orientation of each joint (calculated from the data set using Buss and Filmore's algorithm [31]). As the resulting data is fairly high dimensional (with a 25 joint character and 3 degrees of freedom each for both rotation and angular velocity we have 150 dimensions) we perform an optional dimensionality reduction using Principal Component Analysis (PCA). PCA finds an optimal reduced set of dimensions for representing the data that will be more efficient than directly using the joint rotations. For example, in the sentry data we have a dimension that corresponds roughly to looking left (though it should be noted that the dimensions are not generally meaningful). The PCA is performed on a data set consisting of both the values and first derivatives of the exponential map data. This ensures that our method takes velocities into account. When performing the PCA, sufficient components were kept to reconstruct the data with less than 1% error. The PCA was performed separately for each data set in order to ensure an optimal representation for that data set. The result is data in a form suitable for clustering. The purpose of clustering is to group similar frames into a set that represents a discrete class of action, for example standing still or turning left. As velocity is included we can distinguish motion, such as turning left, from static poses, such as looking left. We use the standard k-means algorithm to form the clusters, this is a good general purpose clustering method that works on many types of data. The main parameter for this method is the number of clusters to choose. If too few are chosen then there will not be enough to distinguish important types of behavior. At the absolute minimum there must be at least one cluster per state of the FSM, otherwise the states cannot be distinguished, though this is unlikely to be enough in practice. If too many clusters are used then the effectiveness of learning can be reduced as individual actions may appear infrequently in the data set. As as rule of thumb using between 1 and 2 times as many clusters as states in the FSM seems reasonable. Figure 1 shows examples of actions from different clusters created from the sentry data.

The result of this processing is a sequence of discrete events and a sequence of discrete actions. We synchronize these two data sets and use the result to learn the FSM model. Since human responses are not instantaneous the motion that responds to an event occurs slightly after the start of the event itself. In order to compensate for this we introduce a small offset between the data sets, putting the motion data 0.5 seconds earlier than the event data. This is not absolutely necessary, and the size of the offset does not have to be exact, as the learning algorithm will automatically find the start of the response. However, a small offset does make the task of the learning algorithm easier and improves the quality of the learned model.

3.2 Learning Finite State Machine Models

In order to learn the Finite State Machines we use a probabilistic learning model called an Input-Output Hidden Markov Model (IO-HMM) [1], that is functionally equivalent to a probabilistic FSM. This equivalence makes it an ideal learning method for FSMs. An IO-HMM is a generalization of a Hidden Markov Model that includes an input variable. It is therefore a dynamical system in that it handles time series of data, with the data divided into a number of discrete time steps. This makes it well suited to handling motion capture data. The structure of an IO-HMM is shown in figure 2 (left). It consists of an input, a current state and an output. The input, E_t, corresponds to our events, it is a discrete random variable with one possible value e_i for each event including the special event e_0 that corresponds to no event occurring at a current time step. Thus, in the sentry there are three values e_l, e_m and e_r for a sound to the left, middle and right of the character, plus a value e_0 for silence. The state, S_t is again a discrete random variable with one value for each state of the FSM. The definitions of the states are learned from data so we cannot give them a priori meanings, however, an example might roughly correspond to a response to a sound to the left. The current state, s_t depends on the previous state s_{t-1} as well as the input. The output, M_t, is the motion data. The possible values of the output variable are the actions that were learned using the clustering method described in the previous section.

3.2.1 FSM templates

There are certain problems with using machine learning within a content pipeline for interactive media. Firstly, a large amount of data is often needed to learn effectively. This can be expensive to capture. Secondly, and perhaps most importantly, a simple application of learning will ignore whatever prior knowledge and requirements that the developers have about the behavior. Thus far we have assumed that the behavior of the actor is completely opaque to the developers of the characters. That the developers have no idea what the behavior model should be and it needs to be learned from scratch. This is typically not the case, developers will have a fairly clear qualitative knowledge of the nature of the behavior, but would find it difficult to determine the exact numerical values of the parameters for the behavior. It is therefore important that developers are able to use their qualitative knowledge to guide and constrain the learning process. This knowledge can be used to constrain the FSM model to be learned, which has the side effect of reducing the number of parameters that have to be learned and therefore the amount of data needed to learn them effectively.

We provide this capability through a set of templates which constrain which state transitions are possible upon receiving each event. The developer can select whichever template best matches the behavior. The templates used here make the assumptions that state transitions are triggered by events and that each event results in a state unique to that event (though later we show how the second assumption can be relaxed for suitable data). Templates consist of constraints on which transitions are possible in a given state with a given event as well as a bias on the learnt transitions probabilities of the possible transition. This means that a template will specify that only a certain number of transitions probabilities will be non-zero. The probabilities of those transitions will be learned by our method. The learning process can also be biased somewhat towards providing higher or lower values for certain transitions. In the sentry example, events are sounds and transitions due to a sound will be constrained to either enter a state that corresponds to a response to the sound, or to remain in the current state, with a bias towards responding to the sound. On the other hand if there is no sound the character will be constrained to remain in the current state or to return to a neutral state, with a bias to remaining in the current state in order to encourage long term responses.

Figure 3 shows the three templates used in the examples section. In our example of the sentry the left hand template is used. The character starts in a neutral state N and has a specific response state, E_i, for each event i. Transitions to each response state are only possible on receiving the specific event, while any state can transition to the neutral state. This creates a behavior in which the character remains in the neutral state until an event occurs (e.g. a sound), this causes it to perform another behavior (e.g. looking in the direction of the sound) and then eventually return to the neutral states. None of the non-zero transition probabilities are deterministic. The character has a learned probability of transitioning to the response state on hearing an event. This makes it possible for delays in responding, as the character (and actor) does not automatically respond immediately when the event starts. Similarly the transition probability to the neutral state determines how long it takes the character to return to that state. The middle template is a linear progression, it has a start state S similar to the neutral state and event states E_i. Transitions to the E_i are the same as the previous example, but on leaving these states rather than returning to a neutral state the character enters a terminal state T from which no further transitions occur. The last template is similar to the first, but it has a separate neutral state N_i for each event. On leaving the event state E_i it transitions to the specific state N_i. This allows for longer term responses to an event.

3.2.2 Learning transition probabilities

There are three probability distributions that have to be learned: the probability of the events $P(E_t)$, the conditional probability of the current state S_t given the current event E_t and the previous state S_{t-1}, $P(S_t|E_t, S_{t-1})$ (called the transition probability); and finally the probability of the current action given the current state $P(M_t|S_t)$. The last two probabilities define the states of our FSM. A state is define by what events it responds to and what actions it implies. The transition probabilities (learned with constraints given by the templates) define what a state responds to, whereas the action probability defines how a state corresponds to a character's actions. Learning the definitions of states from data allows us to learn more nuanced behavior models, i.e. complex relationships between states and behaviors. We can learn behaviors that are not entirely deterministic responses to events and actions that are sometimes paradoxical in a given context. These complex factors would be difficult to include in a hand authored state model.

These values are learned from the data set described in the previous section. This consists of sequences of values for the event (E_t) and motion (M_t) variables but the state variable is unobserved. We use the Expectation-Maximization (EM) algorithm, which is the standard method of learning IO-HMMs with hidden states. It alternately estimates the expected value of the unobserved state variable given the current estimates of the probability distributions and then updates the probability distributions based on the current observed and expected values. We use Markov Chain Monte Carlo sampling to estimate the expected value of the state variable. This approximate method achieves efficient performance when learning a large number of hidden states.

The templates described in the previous section are implemented in terms of constraints on the transition probabilities of the FSM. This type of constraint can easily be implemented within the Expectation-Maximization learning framework. Within that framework if any values of the starting estimate of the probability distribution are zero those values will always remain zero throughout the learning process. The constraints can therefore be specified as zero values within the starting distribution. The starting distribution is also initialized so as to give a high probability to transitions into event states E_i (0.9 in our experiments) and low probabilities of transitions out of events states given the 0 event (0.1), this provides a bias that helps guide learning. The final probabilities are learned from the data.

Once the probability distributions are learned we can use the distribution for the current state, $P(S_t|E_t, S_{t-1})$, directly as the transition probabilities of the Finite State Machine. The distribution of the motion given the current state is used to learn a policy for controlling the motion graph as described in section 3.3.1.

3.2.3 Immediate responses

Up to now we have assumed that an event can only influence behavior through changes in state. This can be somewhat inefficient, as quite a lot of behavior can be implemented in an entirely reactive way. In other words the change in behavior can depend only on the current event, without the longer term alteration of behavior implied by a change of state. For example, when someone talks to the character, it should look at the speaker, but this should only happen for the duration of the speech and no longer. It is therefore inefficient to model this behavior as a state change when the state would entirely be determined by the presence or absence of speech. None the less the character will have a state that is determined in response to other events and the reactive behavior should depend on this current state. For example, if the character is not paying attention it will not look at the speaker.

We extend the IO-HMM model as shown on the right side of figure 2. A new class of events are introduced, immediate events (E_t^{im}) that are responded to reactively. These events are handled separately from the state changing events and are a separate element of the data set (again we introduce a special event e_0^{im} representing no event). The probability distribution of the motion data now depends not only on the current state but also on the immediate event: $P(M_t|S_t, E_t^{im})$. Otherwise the model is identical to the one described above and is learned in the same way.

3.2.4 Combining States

In certain situations, characters can produce the same responses to different events (e.g. the second example in section 4.1). In this case it is more efficient to represent the response to the different events as a single state, rather than as a set of identical states, one for each event. The responses given by two states can be defined to be the same if the output probability distributions are similar and therefore equivalent states can be detected. The standard measure of the difference between probability distributions is the Kullback-Leibler divergence:

$$KL(P_1||P_2) = \sum_x P_1(x) \ln \left(\frac{P_2(x)}{P_1(x)} \right)$$

This quantity is not symmetric so in order to measure the difference between the output probabilities of two states we use a symmetric version:

$$D(P_1, P_2) = KL(P_1||P_2) + KL(P_2||P_1)$$

If this measure is low (below 1 in our experiments) then the two states can be considered the same. For each pair of events e_1 and e_2 the measure D is evaluated for the state resulting from those events. If D is less than the threshold for all of the states resulting from e_1 and e_2, the two events can be considered equivalent. The learning process is repeated, treating e_1 and e_2 as the same event.

3.3 Learning to control animation

As the definitions of states are learned we can't have a priori rules to control animation based on the states. We therefore use the probabilities found in the last section to learn a policy to control our animation engine. For animation generation we use Motion Graphs [2–5]. A Motion Graph (figure 4) is a data structure built from a data set of motion. It is a graph structure in which each edge corresponds to a short motion clip. The nodes are choice points in which one of a number of possible outgoing edges (and therefore motion clips) can be chosen. The nodes are chosen in such a way as to ensure smooth transitions between incoming and outgoing edges at a node. New animations are generated by performing a walk of the graph and concatenating together the motion clips corresponding to the chosen edges.

For the application described in here it is important to find a suitable trade off between two factors in the performance of the motion graph: real time response and continuity. Characters' behavior should respond in a timely and appropriate way to events and changes of their own state. While responses do not need to be instant (human responses are not instant) response times do need to be comparable with those of the actor in the data set. As events are received in real time it is not possible to look ahead to achieve this response. Responsiveness should not come at the cost of significant loss of continuity or realism of animation. In this context it is not sufficient to use the mathematical concept of derivative continuity as a C^2 (continuity of second derivative) motion can still contain many changes of direction and velocity that seem unrealistic in an animation. It is important to ensure that gestures and other movements are, as far as possible, not interrupted and there are relatively few changes in direction introduced by the method. We achieve responsiveness by using a motion graph with a hub-like structure and short edges, which ensure that transitions can occur frequently and that there is a large choice of out going edges for each transition. Continuity is maintained by biasing the selection of edges towards continuing to play the current animation clip rather than transitioning to a new clip.

We use a Hub-based Motion Graph similar to those proposed by Gleicher et al [5]. The graph has a small number of nodes corresponding to commonly occurring poses in the data set. This structure has the advantage that each node has a large number of outgoing edges, and it is therefore possible to be flexible in choosing the best motion clip at each node. In our implementation we choose the poses that correspond to the nodes to be the actions selected by the clustering method described in section 3.1. Transition points are chosen by finding frames that are similar to the nodes, using the same distance measure that we used for clustering. As the data we use is of fairly static behavior, transitions were frequent and therefore edges short.

3.3.1 Driving the Motion Graph

This section describes how edges are selected in the motion graph based on the current state of the FSM. This is done so as to ensure rapid responses to changes in state while maintaining the appearance of continuous motion. As described in the previous section, the edges of the motion graph are short, making it possible to have frequent transitions ensuring rapid responses to state changes. However, frequent transitions can make the animation appear discontinuous and fragmented, so measures need to be taken to counteract this tendency. When the state of the FSM changes it is important that the animation reflects this change quickly, whereas when the state stays the same it is more important that the animation is continuous. We therefore use different methods for selecting edges in these two situations.

For the first transition after a state change we choose the next edge solely based on how well it matches the new state. We define a value V_{ms} for each possible combination of edge m and state s. V_{ms} quantifies how well the motion attached to the edge matches the state (see the next section for the exact definition). For example, for a state that responds to an event to the left we would expect V_{ms} to be high for motions that involve looking left and low for those that look right. The value is defined as the expected future match given that we select the edge m, and therefore takes future transitions into account. This is important as it might be that none of the edges at the current node are a good match, but some will lead to better matches. The new edge is selected randomly with probability proportional to V_{ms}. In order to avoid very poor choices edges with low V_{ms} are discarded before this choice we made. If the V_{ms} of an edge divided by the maximum V_{ms} of all the outgoing edges at the current node is less than a threshold then it is discarded. In our experiments we used a threshold value of 0.8, this value discards a large proportion of the outgoing edges, ensuring that the remaining edges are good. As our Motion Graph topology has a very large number of outgoing edges per node, the number of remaining edges is still high.

When the state has not changed we introduce a bias in the selection to make the animation more continuous, i.e. to ensure that the movement is smooth and to minimize sudden changes. Firstly, when calculating the probability of choosing an edge we multiply V_{ms} by a penalty based on the quality of the transition. This ensures smooth transitions without unnatural motion. For this penalty we use the function suggested by Lee et al [4]:

$$\rho = \exp(\frac{-D_{i,j}}{\sigma})$$

where $D_{i,j}$ is a distance measure between the incoming edge, i, and the outgoing one j. We use distance in PCA space as described above. σ is a parameter for controlling the degree to which poor transitions are penalized. In our method this becomes a useful trade off parameter that controls how likely the system is to choose a poor transition that is a good match to the current input features. Values between 0.1 and 0.3 seemed to work well.

The second bias factor is a preference for continuing the current action rather than making a sudden change to a new type of behaviour. For this we make an explicit distinction between two types of edge. Continuation Edges (shown in black in figure 4) represent continuing along the original motion sequence from the current node without making a transition. Transition Edges (shown in grey in figure 4) represent a transition to a new position in the same or a different motion sequence. For a continuous animation we want to reduce the number of transition edges taken, but still take a few to ensure variety. The selection process is therefore strongly biased towards selecting continuation edges. Firstly, as taking a continuation edge results in motion that exists in the original data set, no transition error can be introduced and so the penalty factor ρ is defined to be 1, and thus ignored. If a continuation edge is very poor in terms of V_{ms}, it should not be chosen so we reject continuation edges based on V_{ms} as described above. If it has not been rejected we choose it with a probability P_{cont} and select another edge as normal with probability $(1 - P_{\text{cont}})$. In our experiments P_{cont} was 0.75.

To summarize we can calculate the probability of choosing each edge. First we define a base probability $P = V_{ms}$ if there has been a state change or $P = \rho V_{ms}$ if there has not. If P is much smaller than the maximum P at that node it is set to 0. If there has been a state change, or P has been set to 0 for the continuation edge, the probability of choosing the node is αP, where α is a normalization factor that ensures that the probabilities sum to 1. Otherwise, the continuation edge is chosen with probability P_{cont} and other nodes with probability $\alpha P(1 - P_{\text{cont}})$.

3.3.2 Finding the value of edges

The selection of edges relies on the ability to assign a value V_{ms} to selecting an edge m in a state s. This is based on the output probabilities of the FSM as calculated in section 3.2 and also on the expected values of future edges that are selected after that edge. As the structure of the graph is known in advance and all the state transition probabilities have been estimated in the learning step the value can be entirely pre-computed. This makes run time evaluation very fast, simply looking up a single value. The method described here is adapted from Gillies et al. [25].

The starting point for calculating V_{ms} is the value of playing the motion attached to edge m in state s, we call this V^0_{ms}. This is calculated using the the output probability distribution of the IO-HMM: $P(M_t|S_t)$, i.e. the probability of an action given the current state. This probability is used as a “score” that rates how appropriate the action is given the state (it is not used directly as the probability of taking the action, the probabilities used are defined in the previous section). V^0_{ms} is simply the average of this distribution over the frames of the motion clip attached to m:

$$V^0_{ms} = \frac{1}{|m|} \sum_{i=0}^{|m|} P(m_i|s)$$

Where $|m|$ is the number of frames in the motion clip m and m_i is the i^{th} frame. Values of V^0_{ms} are only assigned to continuation edges, transitions are assumed to have zero value, as they are not in the original data set.

This base value is augmented with the expect value of future edges. This is done using Dynamic Programming based on the Bellman Equation, a common technique in Reinforcement Learning [20]. The Bellman Equation gives the expected value of an action in a dynamic environment. In this case it is how well we expect our future actions to match the behavior in our training data, given that we select a given edge. For our system it gives the following equation:

$$V^*_{ms} = V^0_{ms} + \gamma^{|m|} \sum_{s'} \sum_{m'} P(s'|s)P(m'|s')(V^*_{m's'}) \tag{1}$$

The sum is over all possible next states s' that the FSM could be in when the next edge is chosen, and over all possible edges m' that could be chosen at the next node. The value inside the sum is the probability of selecting the edge m' multiplied by its value (i.e. the expected value of choosing the edge). The probability of choosing an edge is decomposed into two terms $P(s'|s)$, the probability of transitioning to the new state s' and $P(m'|s')$ the probability of selecting the edge m' in state s'. $P(m'|s')$ is calculated as described in the previous section, while $P(s'|s)$ is calculated by marginalizing the state transition probability from the FSM over all possible events:

$$P(s'|s) = \sum_{e} P(e)P(s'|s, e)$$

The term $\gamma^{|m|}$ is a discount term that weights future values lower than immediate ones, $|m|$ is the length of the motion at edge m, thus ensuring that motion lengths are taken into account. γ is another parameter that trades off rapid response against long term quality, a low value is needed to ensure quick responses, 0.1 was used in our experiments.

V^*_{ms} can be calculated by dynamic programming. Equation 1 is iteratively applied, at each step the value of V^*_{ms} for each edge is updated using the current values of the other edges.

4 Results

This section will describe the results of experiments with three sets of data collected as described in section 3.1. The motion capture data was generally of good quality though there are some errors with the data of the hands, and the spine data was occasionally incorrect due to lost markers in sitting and leaning over positions.

4.1 Sentry

The images shown in figure 5 are of the sentry example used in the rest of this paper. The character responds to events such as explosions and other loud noises. Two data sets were collected, one of a brave sentry that looked in the direction of the noise to investigate, and the other of a fearful sentry that ducked down to hide. Both data sets consisted of 3 clips of about 80 seconds each. For the first data set 5 principal components were found while 10 were found in the second. For both data sets 5 actions were used in the clustering method. The first template from section 3.2.1 was used and there were no immediate events. There were three events, for a sound to the left, middle or right. The template gives 3 response states, one for each event, and one neutral state. In the fearful example the responses to all events were similar so the output states were merged as described in section 3.2.4. To test the results a user interface was set up through which it was possible to trigger new events by pressing buttons.

4.2 School Kid

In the second example (figure 6) the actor plays the part of a school pupil. There are two scenarios. In the first the character is doing an examination, he listens to the teacher until the signal is given to start, and then works on the test until he finishes at which point he relaxes (in order to show this in a reasonable time, the length of the test has been made unrealistically short). There were 5 motion capture sequences each lasting about 30 seconds. 12 Principal Components were found and 5 clusters were used. The second template was used without immediate events. There was 1 event, resulting in 1 start state, 1 response state and 1 terminal state. For the demo an interfaces similar to the pervious example was used to trigger the start of the test. The second example is more complex. The character is bored and does not pay attention to the teacher until the teacher shouts at him, then he pays attention when the teacher talks and works when the teacher is quiet, at least for a while. 4 sequences of about 2 minutes each were used. 18 PCs were found and 5 clusters were used. Boredom is modeled as the neutral state from the first template and there is a single event, when the teacher shouts. The result is two states, neutral and response. Whether the teacher is speaking or silent is modeled as an immediate event. The events were extracted by thresholding the audio volume and the results were created by applying the same method to a new piece of audio.

4.3 Sports Fan

The final example (figure 7) is of a sports fan. The actor responded to a fictional soccer commentary. This commentary was manually marked up with events: goals for either side were events and possession of the ball by either side was an immediate event. Three motion capture sequences were used, each of approximately 2 minutes 30 seconds. 18 Principal Components were found and 8 clusters were used. The third template was used so that the neutral state, and therefore the character's response to possession, was different depending on whether the last goal was scored by the character's own team or the opponent. With two events (a goal by either side) this resulted in 4 states, 2 neutral and 2 response. The results were generated using data from a "real" sporting event: a video of a table football match that was manually marked up.

5 Conclusion and Further Work

We have proposed machine learning from motion capture data as an effective way of selecting parameters of an interactive animated character's AI system so that the character's behavior reflects an actor's performance. We believe that this approach has great potential to enhance the individuality and believability of characters in interactive media. This general approach has been demonstrated by a method for learning the parameters of a Finite State Machine controller. As FSMs are widely used in the game industry, this demonstrates the practical potential of the idea. The method is fast at run time, relying largely on look ups of pre-computed values, and runs comfortably at real time rates. This means that it would not provide a great overhead for inclusion in existing systems. Learning time is rather longer, about 20 minutes to learn the FSM and 20 more minutes to create the motion graph and calculate V^*_{ms} on a 2.4 GHz dual core PC (the code was not multi-threaded). It is likely that this can be speeded up with some optimization (it is currently implemented in an interpreted language). The examples given in this paper show a range of possible applications of the method. Future work will involve trying the method in new and more complex applications in order to better understand how it applies and the limits of the method. New, more rigorous, evaluation methods will need to be developed in order to assess the success of this technique on future applications. The major challenge in evaluating this work is that human behavior is not deterministic, and that many different actual behavior patterns may be a reasonable response to a set of events. In fact, the training data used consists of a number of different responses to the same sequence of events. For this reason, we cannot test by a simple comparison to training data, as output of our system may be different but still valid. Proper evaluation is therefore an important open question in this research area.

The proposed method aims at automatically finding parameters of a character, but like most learning methods, it introduces a new set of (meta-) parameters. In the paper we have tried to describe the effect of most of the parameters and tentatively suggested suitable values. Many of the parameters relate to the trade off between response and continuity, which is a difficult problem. In order to increase the usability of the method we need to investigate either automatic methods for setting the parameters or easily understandable ways of presenting them to users.

The most obvious limitation of the method is that it will only work if the state of the character can be inferred easily from the character's movements and the events that have occurred. In some ways this does not seem an important limitation, as there is little point in a character having a state if no one can distinguish the state based on the character's behavior. However, as characters become more complex it is likely that there will be meaningful states that are difficult to infer using the machine learning methods described here. This type of limit will only emerge with new applications, but we believe they can be over come by combining different machine learning methods with new ways of balancing learning with prior knowledge. The use of prior knowledge in this context is an interesting research area; how much should the end result be based on the intuitions of developers and how much on the data? Clearly a balance is needed, and we have provided this using templates for FSM topology. This is only one of many possible methods that need to be evaluated both in terms of their power to aid learning and their usability by developers. Methods are also needed for cases where developers have little prior knowledge. We look forward to investigating these issues in the future.

Acknowledgment

This work has been supported by the UK Engineering and Physical Sciences Research Council grant, "Empathic Avatars". The author would like to thank the member of the UCL Department of Computing Virtual Environments and Computer Graphics group and the UCL Centre for Computational Statistics and Machine Learning for their help, support and suggestions. In particular I would like to thank David Swapp for his help with Motion Capture and Amanda Gower for the use of her character models.

References

[1] Y. Bengio and P. Frasconi, "An input output hmm architecture," in *Advances in Neural Information Processing Systems (NIPS) 7*, G. Tesauro, D. S. Touretzky, and T. K. Leen, Eds. MIT Press, 1994, pp. 427–434.

[2] O. Arikan and D. A. Forsyth, "Interactive motion generation from examples," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 483–490, Jul. 2002.

[3] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 473–482, Jul. 2002.

[4] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard, "Interactive control of avatars animated with human motion data," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 491–500, Jul. 2002.

[5] M. Gleicher, H. J. Shin, L. Kovar, and A. Jepsen, "Snap-together motion: Assembling run-time animation," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 702–702, Jul. 2003.

[6] B. Chiu, V. Zordan, and C.-C. Wu, "State-annotated motion graphs," in *VRST '07: Proceedings of the 2007 ACM symposium on Virtual reality software and technology.* New York, NY, USA: ACM, 2007, pp. 73–76.

[7] J. Funge, X. Tu, and D. Terzopoulos, "Cognitive modeling: Knowledge, reasoning and planning for intelligent characters," *Proceedings of SIGGRAPH 99*, pp. 29–38, August 1999.

[8] R. Aylett, J. Dias, and A. Paiva, "An affectively-driven planner for synthetic characters." in *ICAPS 2006.* AAAI Press, 2006.

[9] C. Brom, J. Gemrot, M. Bída, O. Burkert, S. Partington, and J. Bryson, "Posh tools for game agent development by students and non-programmers." in *9th International Conference on Computer Games: AI, Animation, Mobile, Educational & Serious Games*, November 2006.

[10] V. Vinayagamoorthy, M. Gillies, A. Steed, E. Tanguy, X. Pan, C. Loscos, and M. Slater, "Building expression into virtual characters," in *Eurographics Conference State of the Art Reports*, 2006.

[11] N. Pelechano, J. Allbeck, and N. Badler, "Controlling individual agents in high-density crowd simulation." in *ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2007.

[12] K. H. Lee, M. G. Choi, and J. Lee, "Group behavior from video: A data-driven approach to crowd simulation," in *2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, Jul. 2007.

[13] M. Brand and A. Hertzmann, "Style machines," in *Proceedings of ACM SIGGRAPH 2000*, ser. Computer Graphics Proceedings, Annual Conference Series, Jul. 2000, pp. 183–192.

[14] E. Hsu, K. Pulli, and J. Popović, "Style translation for human motion," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 1082–1089, 2005.

[15] E. Hsu, S. Gentry, and J. Popović, "Example-based control of human motion," in *2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, Jul. 2004, pp. 69–77.

[16] M. Kipp, M. Neff, K. H. Kipp, and I. Albrecht, "Towards natural gesture synthesis: Evaluating gesture units in a data-driven approach to gesture synthesis," in *IVA*, 2007, pp. 15–28.

[17] C. Pelachaud and I. Poggi, "Subtleties of facial expressions in embodied agents." *Journal of Visualization and Computer Animation.*, vol. 13, pp. 287–300, 2002.

[18] H. P. H. Shum, T. Komura, and S. Yamazaki, "Simulating interactions of avatars in high dimensional state space," in *Proceedings of theSymposium on Interactive 3D Graphics, SI3D 2008*, E. Haines and M. McGuire, Eds., 2008, pp. 131–138.

[19] T. Kwon, Y.-S. Cho, S. I. Park, and S. Y. Shin, "Two-character motion analysis and synthesis," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 3, pp. 707–720, 2008.

[20] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction.* Cambridge, MA: MIT Press, 1998.

[21] B. Blumberg, M. Downie, Y. Ivanov, M. Berlin, M. P. Johnson, and B. Tomlinson, "Integrated learning for interactive synthetic characters," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 417–426, Jul. 2002.

[22] J. Lee and K. H. Lee, "Precomputing avatar behavior from human motion data," in *2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, Jul. 2004, pp. 79–87.

[23] J. McCann and N. Pollard, "Responsive characters from motion fragments," *ACM Transactions on Graphics*, vol. 26, no. 3, Aug. 2007.

[24] A. Treuille, Y. Lee, and Z. Popović, "Near-optimal character animation with continuous control," *ACM Trans. Graph.*, vol. 26, no. 3, p. 7, 2007.

[25] M. Gillies, X. Pan, M. Slater, and J. Shawe-Taylor, "Responsive listening behavior," *Computer Animation and Virtual Worlds*, vol. 19, pp. 1–11, 2008.

[26] T. Mukai and S. Kuriyama, "Geostatistical motion interpolation," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 1062–1070, Aug. 2005.

[27] C. K. Liu, A. Hertzmann, and Z. Popović, "Learning physics-based motion style with nonlinear inverse optimization," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 1071–1081, Aug. 2005.

[28] M. Gleicher, "Motion editing with space time constraints," in *symposium on interactive 3D graphics*, 1997, pp. 139–148.

[29] J. Lee and S. Y. Shin, "A hierarchical approach to interactive motion editing for human-like figures," in *Proceedings of SIGGRAPH 99*, ser. Computer Graphics Proceedings, Annual Conference Series, Aug. 1999, pp. 39–48.

[30] F. S. Grassia, "Practical parameterization of rotations using the exponential map," *Journal of Graphics Tools*, vol. 3, no. 3, pp. 29–48, 1998.

[31] S. R. Buss and J. P. Fillmore, "Spherical averages and applications to spherical splines and interpolation," *ACM Transactions on Graphics*, vol. 20, no. 2, April 2001.

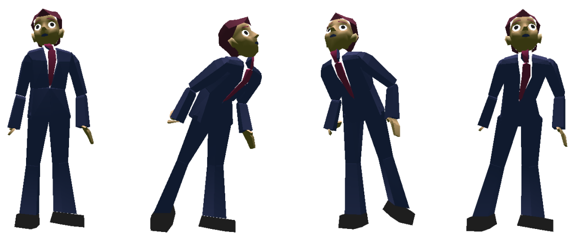

Figure 1: Examples of of the different "actions" found by our clustering method. These are example frames from our first sentry data set, each of these frames was classified as a different action by our method.

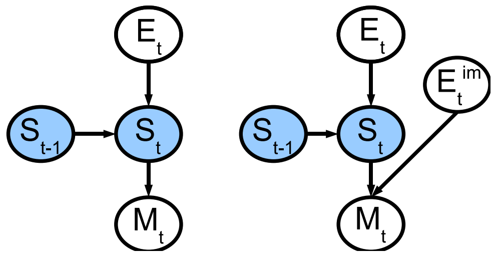

Figure 2: An Input-Output HMM (left) and a modified version to include immediate responses (right)

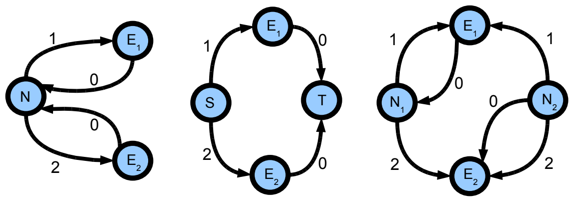

Figure 3: Finite States machine templates. Left: alway returns to a neutral state, middle: a linear progression and right: multiple neutral states. Transitions between event states E_i are not show for clarity but they are possible in the first and last templates. Each template is shown with two events, but they can accommodate any number, with an equivalent increase in the number of states.

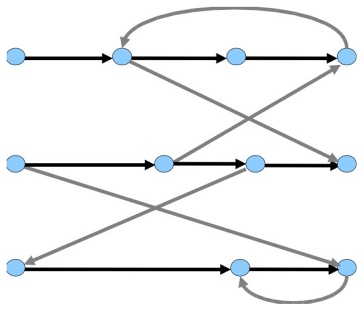

Figure 4: Motion graphs. Motion graphs consists of a number of motion sequences (shown horizontally from left to right) and transitions between them. Continuation edges are shown in black and transition edges in grey.

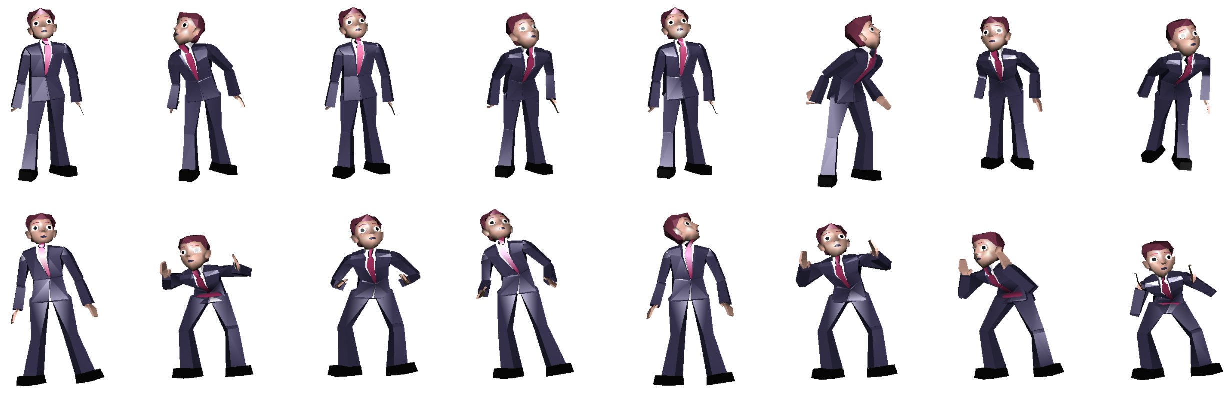

Figure 5: The sentry top: brave, bottom: fearful. The first frame of both is the character the initial neutral state. Then a sound occurs to the characters right and he responds (2) and then gradually return to a neutral state (3). An event then happens to the right (4), followed again by a return to neutral (5). In the last 3 frames a sequence of rapid events occurs one after the other.

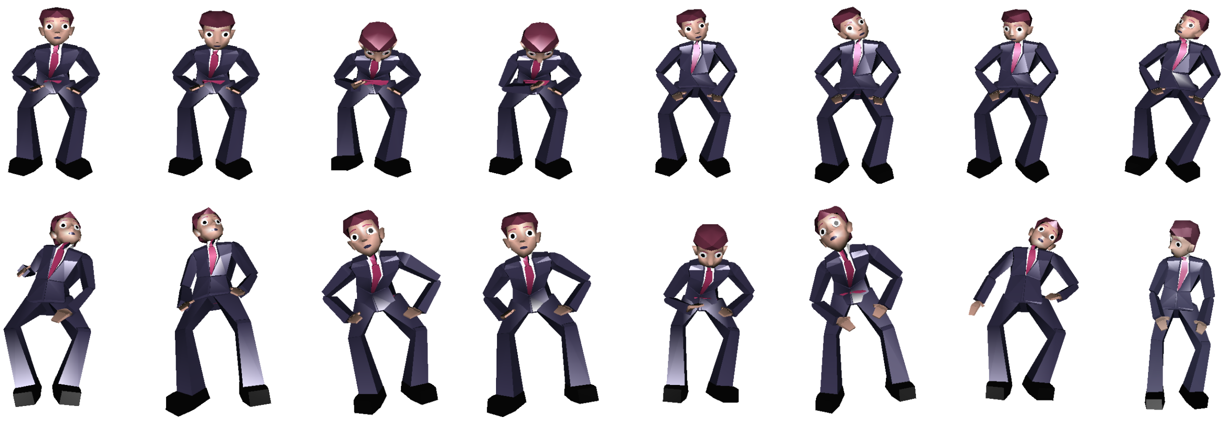

Figure 6: The school kid character. The exam example is show on the top and the listening example on the bottom. The exam example starts in a neutral waiting state (frame 1) the exam then begins, resulting in a new "working" state (2-4) that eventually returns to a final "resting" state (5-8). In the listening example the character starts in an initial state corresponding to not paying attention (frames 1-2), an event (being shouted at) results in a new state in which the character is more attentive when spoken to (3-4) and works when not (5). Eventually the character returns to the initial state, in which it make little difference if he is spoken to or not (6-8).

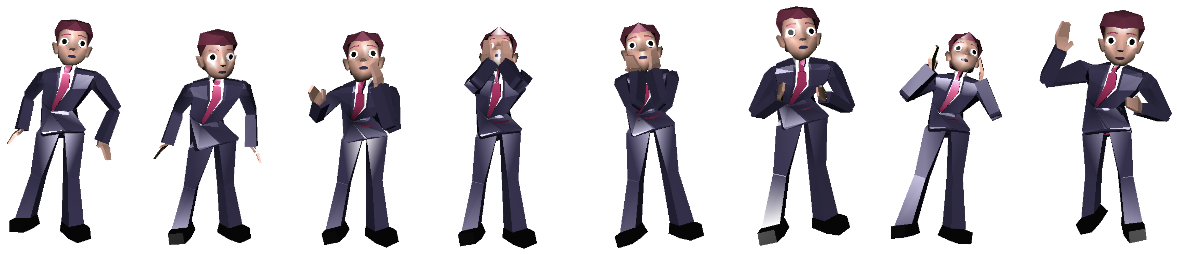

Figure 7: The sport fan. The character starts in a neutral watching state, but responds differently to ball possetion by his own team (frames 1-2) or the opposition (3). A goal by the opposition results in a reaction state (4-5), and the character returns to a different watching state with more nervous behaviour (6-7), a goal by his own team results in a different reaction state (8).