Generating Believable Virtual Characters Using Behavior Capture and Hidden Markov Models

Richard Zhao[1], Duane Szafron[1]

[1] Department of Computing Science, University of Alberta
Edmonton, Alberta, Canada T6G 2E8
{rxzhao, dszafron} @ualberta.ca

Abstract. We propose a method of generating natural-looking behaviors for virtual characters using a data-driven method called *behavior capture*. We describe the techniques for capturing trainer-generated traces, for generalizing these traces and for using the traces to generate behaviors during game-play. Hidden Markov Models (HMMs) are used as one of the generalization techniques for behavior generation. We compared our proposed method to other existing methods by creating a scene with a set of six variations in a computer game, each using a different method for behavior generation, including our proposed method. We conducted a study in which participants watched the variations and ranked them according to a set of criteria for evaluating behaviors. The study showed that behavior capture is a viable alternative to existing manual scripting methods and that HMMs produced the most highly ranked variation with respect to overall believability.

Keywords: behavior, virtual characters, behavior capture, Hidden Markov Model, HMM

1 Introduction

A story-based computer game contains virtual characters. Most of them are AI-controlled non-player characters (NPCs), who interact with the player character (PC), other NPCs, and the environment. Although games display increasingly realistic graphics and physics, NPC behaviors have improved slowly. Players are demanding more realistic behaviors for the NPCs. Rather than standing or wandering aimlessly, NPCs should converse with other NPCs and interact with game objects in realistic ways. They should also react to events such as explosions or unusual events. Creating natural-looking behaviors for NPCs is not inexpensive. In a typical commercial story-based game, there are hundreds or sometimes over a thousand NPCs. Since manually scripting each NPC individually requires extensive resources, most NPCs in most commercial games have simple and repetitive behaviors.

We propose a new method of creating NPC behaviors called *behavior capture*, based on the concept of motion capture. With motion capture [6], sensors are attached to the bodies of actors, and as the actors move their bodies, the spatial locations of their body parts are recorded. The data is used to animate virtual characters to move in the same way. Our system of behavior capture is based on a similar idea of using captured traces to guide NPC behaviors, but behavior capture is not a generalization of motion capture. Our behavior traces represent high-level intentions as opposed to motion trajectories in space. For example, the NPC may use an entirely different path to approach another NPC during game-play, since the relative positions of the two NPCs may be different than they were during training.

Behavior capture enables a game designer to take control of a particular NPC during training and perform *exemplar* behaviors. It captures traces of the exemplar behaviors and generalizes them. Generalization is necessary to generate natural behaviors with short training times. First, interactions should be generalized. If a particular NPC should talk to a particular set of NPCs (such as rich NPCs only) in a tavern, then the trainer should be able to train this NPC by talking to only one rich NPC. If another NPC should have the same behavior, the trainer should not have to train the second NPC separately. Second, during game play, the NPC should not repeatedly follow the exact training trace. We present several trace generalization mechanisms, including a technique that learns a Hidden-Markov Model (HMM). Our generalization and learning do not require scripting.

The term *behavior capture* has been used in commercial software to describe the LiveAI technique introduced by AiLive [1], and by TruSoft [13]. Unfortunately, there are no publications describing what technique is used to generalize behaviors after training and no indication of the level of behaviors that can be learned, although there is a video that highlights some behavior in a showcase combat scenario [2].

To verify the utility of behavior capture, we created a tavern scene in a commercial video game. We generated a set of scene variants, using a collection of behavior generation techniques, including manual scripting and several forms of behavior capture. We conducted a study in which participants *played* each scene variant and ranked them by: most active characters, most unpredictable characters, most plausible action sequences, most diverse character actions. Participants also ranked overall believability and rated overall believability of each variant.

2 Related Work

There are many methods of creating behaviors for virtual characters. Traditionally, programmers had to script individual behaviors for each character. Orkin and Roy devised a data-driven approach to generating behaviors, using unsupervised learning of behavior and dialogue in a game that simulates a restaurant [10]. They take advantage of the massive online-gaming community and use it to collect data as training examples. A character in the game has a set of goals and corresponding priorities, used to guide interactions. After goal selection, the character retrieves candidate plans and sends them to the critics system, which uses criteria to reject plans. Experiments compare the ability of the plan network and humans to differentiate between typical and atypical restaurant behavior. Thurau et al. [12] describes methods of inferring goals from replays of human games for a virtual character in a First-Person Shooter game. Ontanon et al. [9] proposes a planning system for a Real-Time Strategy game that can be learned from human demonstration.

These two approaches assume there are specific goals (such as killing an enemy) for the virtual characters to achieve, and different methods are used to find ways to satisfy such goals. Our research targets day-to-day behaviors of virtual characters in a non-combat virtual environment, where NPCs often behave without clear-cut goals.

MacNamee [7] proposed a technique called role-passing, which allows an NPC to exhibit behavior variations by assuming roles in particular situations. Such NPCs can be implemented using *level-of-detail*, which means that their behaviors can be modeled abstractly when the player is not looking, and in more detail when the player is focused on them. These *proactive persistent agents* provide a realistic experience. Individual behaviors are driven by an artificial neural network, which is trained using about 300 hand-crafted examples by the author. While these examples contained a set of interesting behaviors, their flexibility to adapt to other situations has not been demonstrated. It is not clear how a game designer can add additional behaviors not included in the pre-designed training examples. Research has been done in the past to analyze player statistics extracted from in-game traces or character attributes [8] [11]. We want to provide a better gaming experience for players, specifically by enabling game designers to create more interesting NPC behaviors.

3 Capturing Behaviors

Behavior capture is a data-driven approach to generating virtual character behaviors. Instead of a programmer specifying how each character should move, speak, and interact with the environment, a game designer takes control of the character and performs the actions that the designer would like to see this character perform. This is done in a game session called *training mode*. In training mode, the designer-controlled character actions are recorded with a behavior capture system. The system remembers what each character did and uses this data to generate new behaviors during game play. There is no need to write programming scripts.

3.1 Training in Neverwinter Nights

We constructed a prototype of our behavior capture system in BioWare Corp.'s Neverwinter Nights (NWN). NWN includes a simple-to-use Toolset that allows designers to create new stories using a C-like scripting language. Since NWN is a medieval fantasy game, tavern scenes are common. A tavern typically has patrons, a bartender, and sometimes entertaining bards. We created a tavern scene and used it as a test-bed for our behavior capture system (Figure 1).

In training mode, the characters start with no behaviors. A game designer takes control of a *trainee* character. Figure 2 shows a trainee in the centre of the screen. At the bottom of the screen, there are buttons representing the actions a trainee can perform (pressing modifier keys display other sets of actions). If the designer clicks on the *Face* button and then clicks on another character, the trainee will turn and face the clicked character, and the *Face* action will be recorded as an action in a sequence of actions the trainee should perform. The game designer can switch trainees at any time, by clicking on the *Become* button and clicking on another character. If a trainer makes an error, the offending trace can be deleted from the training record. To avoid pressure on the designer during training, designer pauses are ignored in training mode and if the trainer wants the trainee to pause, an explicit wait action is selected.

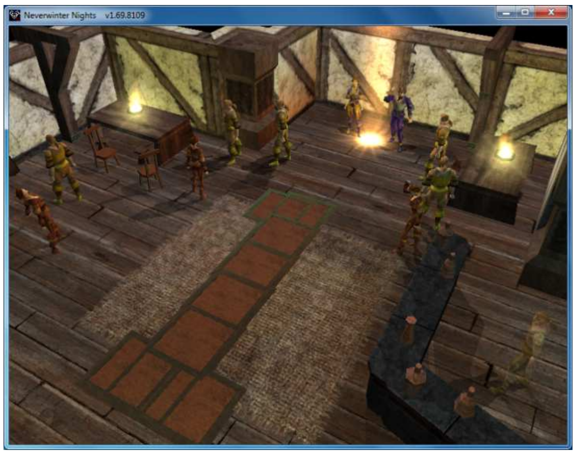

Fig. 1. The tavern scene in Neverwinter Nights.

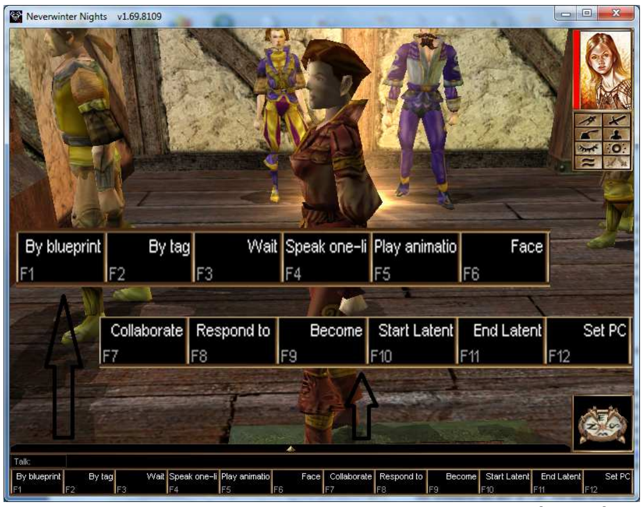

Fig. 2. Training a character. The action bar is enlarged in the figure for clarity.

3.2 Behavior Types

Cutumisu [4] introduced the behavior ontology shown in Figure 3. Behaviors are categorized as independent or collaborative. Independent behaviors are performed by one NPC, while collaborative behaviors are performed with a partner. An NPC may have an independent behavior to *sit on a chair* and a collaborative behavior to *talk to another NPC*. Behaviors can also be classified as proactive, reactive (reacting to a partner’s initiative), or latent.

A new proactive behavior is initiated when no other behavior is active. A latent behavior is triggered by a specific event and has a higher priority than any proactive behavior so it can interrupt. For example, an NPC may perform a proactive behavior to *sit* on a chair. When the NPC is done, a new proactive behavior is selected. The NPC may *wait* or *talk* to another NPC. However, if an explosion occurs, a latent behavior to flee from the explosion can be triggered and the *flee* behavior will interrupt the current proactive behavior.

Our behavior capture system supports this behavior ontology during training mode. For example, to train a collaborative behavior, a designer clicks on the *Collaborate* button and clicks on another character so the system recognizes the collaboration partner. The designer trains one character first, and then switches to the other character and trains a corresponding reactive behavior, clicking the *Collaborate* button again to end the collaborative behavior trace.

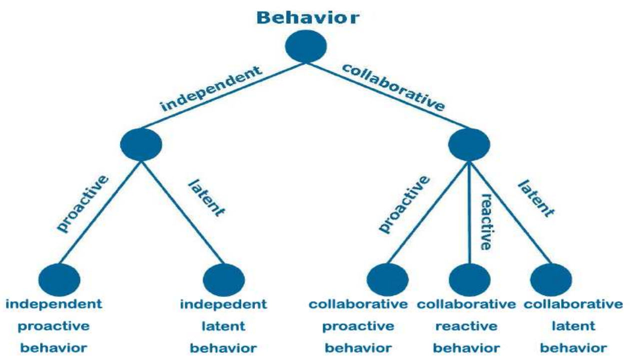
Behavior
independent
collaborative
proactive
latent
proactive
reactive
latent
independent proactive behavior
indepedent latent behavior
collaborative proactive behavior
collaborative reactive behavior
collaborative latent behavior

Fig. 3. Behavior architecture, adapted from Cutumisu [4].

A latent behavior is based on a game event. To train a latent behavior, the trainer first clicks on the *Start Latent* button. The designer next triggers the appropriate event. In our tavern scene, the designer may want some tavern patrons to cheer when the bard finishes performing and *exits* the stage. Since *exits a trigger* (an area on the ground) is an event in the game, performing this action in *Latent* mode enables the behavior capture system to record this event. The trainer clicks on the bard and then moves the bard out of the trigger area. Once the event is recorded, the designer trains an NPC to react to this event by clicking on the appropriate NPC and selecting behaviors, such as a *face* the bard behavior and a *cheer* animation behavior.

4 Generating Behaviors

After data is gathered using training mode, the behavior capture system produces the NPC behaviors. Currently, the behavior capture system supports three types of generalizations. First, there can be hundreds of NPCs, and the designer may want the training of one NPC to apply to multiple NPCs – *character generalization*. Second, the designer may want a trained NPC interaction with a specific object to generalize to an interaction with any one of a group of objects – *object generalization*. Third, the designer may not want the NPC to perform the training actions strictly in the training sequence order during game play and then repeat them in the same order once the sequence is complete, so we perform *sequence generalization*.

4.1 Character and Object Generalization

To support character and object generalization, the behavior capture system uses *categories* of objects and characters. For example, if a designer trains a character to *sit on a chair*, the action is not to *sit on that specific chair*, but to *sit on any chair in a category*. During game play, the character would sit on one of many chairs. To force a character to sit on a specific chair, the designer places this chair in its own category. Object categorization mechanisms are game-specific. For example, in NWN, the designer can use two different categorization mechanisms – tags and blueprints. The designer assigns a tag to each game object. The same tag can be assigned to different kinds of objects. During training, any interaction with an object can be generalized to an interaction with a random object with the same tag. For example, the trainer can train an NPC to converse with any tavern patron whose tag is *conversable* by conversing with any one of them. Alternatively, in NWN, the trainer can choose to generalize by blueprint – the template used to create an object. In this case, sitting on a chair would train an NPC to sit on any object created using the chair blueprint, regardless of tags. In our NWN trainer, the designer can toggle between using tags or blueprints to categorize objects. To support character generalization, when a designer trains an NPC for one blueprint, all NPCs with the same blueprint receive this training. It is easy to make custom blueprints from existing blueprints so creating categories that correspond to groups with common behaviors is straightforward.

4.2 Sequence Generalization

A behavior capture system needs an algorithm to order behaviors based on the training traces. A simple approach, *no sequence generalization*, generates a sequence of actions that exactly matches the recorded sequence and then repeats this sequence. However, a player may regard this repetition as unnatural. Alternative approaches could select actions from the set of training actions in a non-deterministic manner. For example, the system could sample uniformly from the set of all trained actions using *random action sequence generalization*. However, in many situations the order is important. To provide some designer control over the non-determinism, we divide the training actions for a single NPC into a set of traces of actions. A designer starts a trace, performs a sequence of actions and ends the trace. The designer usually performs many traces, each of which contains a short sequence of actions that form a cohesive sequence. For example, one trace may consist of three actions: 1) *wait a few seconds*, 2) *say "I'm thirsty"* and 3) *walk to the bar*. Another trace may consist of three actions: 1) *wait a few seconds*, 2) *say "I'm tired"* and 3) *walk to a chair*. The random action technique could result in unrealistic action sequences, such as 1) *say "I'm tired"*, 2) *walk to the bar* or 1) *say "I'm thirsty"*, 2) *walk to a chair* or even 1) *wait a few seconds*, 2) *wait a few seconds*, 3) *wait a few seconds,* etc.

The *random trace sequence generalization* technique uniformly select traces instead of actions. It tries to maintain the plausibility of action sequences created by the designer. However, over longer periods of time (many traces), this technique can produce behaviors that players view as repetitious. Therefore, we created a third sequence generalization that maintains traces to some extent, while producing emergent sequences that may reduce repeatability. We introduce *HMM sequence generalization* that uses a Hidden Markov Model (HMM) to select actions that have a bias towards selecting actions in the order specified by the designer.

A Markov Model is a statistical model with states, transitions and outputs. One state is a special state called the start state. Each state is connected to a set of other states by probabilistic transitions. In addition, each state has an output probability of outputting a set of outputs. An HMM is a Markov Model whose states are unobserved (hidden). In our application, the output is one of the behavior actions that the designer used in a trace. Hidden states are hidden to a game designer. The number of hidden states is a parameter of the HMM sequence generalization technique.

One HMM generates the proactive behaviors for each character. The Baum–Welch Algorithm [3], a generalized expectation-maximization algorithm, uses the training traces to teach the HMM. The HMM adjusts its transition and output probabilities to fit the trace sequences. If the trainer follows the *say "I'm thirsty"* action with the *walk to the bar* action, the HMM will favor this action order. If the trainer trains the NPC to converse three times as often as ordering a drink, the HMM will generate a similar 3 to 1 ratio. The behaviors generated by the HMM are stochastic, but are somewhat consistent with the training traces. The number of hidden states parameter controls the consistency, with higher consistency achieved by more hidden states.

5 User-Study and Evaluation

5.1 User-Study

We conducted a user study to evaluate the utility of behavior capture. Participants were enrolled in a first-year university psychology class. They did not necessarily have video game experience. Our goal was to show that behavior capture is a viable alternative to typical commercial game NPC behaviors created by manual scripting. We created six variants of a tavern scene in NWN, which are identical in all aspects except in the way that the NPC behaviors were generated. The six scene variations were constructed using the techniques listed in Table 1. The mapping between Scene Variation number and technique was generated randomly to avoid any bias.

Study participants were asked to watch the six variations, and to rank them according to the criteria listed as the first column of Table 2. We also asked the participants to rate the overall believability of each variation on a scale of 1-4.

Technique T1 is a baseline, with all characters exhibiting only stock idling animations provided by the NWN game engine, such as stretching their arms. Technique T2 is hand-scripted – characters behave according to a representative commercial role-playing game, Dragon Age: Origins. The variation was scripted to combine the behaviors in two taverns, *Lothering* (bards entertaining) and *Redcliffe* (a server who walks around). If we only used one of the taverns the behaviors would have been very simple and we believe the evaluations of this variation would have resulted in a worse ranking. The other four techniques used behavior capture.

Table 1. Behavior generation techniques. Scene Variation numbers (shown to participants instead of technique numbers) were randomly assigned to techniques to avoid bias.

Technique	**Scene Variation Number**	**Behavior Generation**
T1	4	No behaviors added (idle)
T2	6	Behaviors hand-scripted by a programmer
T3	5	Behavior capture with no sequence generalization
T4	1	Behavior capture with random action sequence generalization
T5	3	Behavior capture with random trace sequence generalization
T6	2	Behavior capture with HMM sequence generalization (using 8 hidden states)

For behavior capture, fourteen different proactive actions (behaviors) were used:

a1) Wait 5 seconds
a2) Collaborate with another patron by conversing
a3) Wave at another patron
a4) Face another patron
a5) Walk to a chair
a6) Walk to the bar
a7) Speak "I'm thirsty"
a8) Speak "I'm tired"
a9) Speak "I'm lonely"
a10) Speak "See you later"
a11) Speak "I'm bored"
a12) Speak "The inn-keeper is busy"
a13) Speak "I see a friend"
a14) Speak "What was that noise?"

The collaborative *converse* behavior included 4 tasks: *face the collaborator*, *walk to the collaborator*, *speak* and then *listen*. The training set contained 10 traces of three actions each: [a5, a7, a6], [a1, a4, a3], [a6, a8, a5], [a9, a2, a10], [a4, a1, a4], [a5, a11, a6], [a6, a12, a5], [a13, a2, a10], [a14, a4, a4], [a4, a3, a1].

Technique T3 is behavior capture with no sequence generalization. The next action is picked directly from the trainer's traces, combining traces into a single long trace to eliminate randomness in picking actions. The list of actions in order was [a5, a7, a6, a1, a4, a3, a6, a8, a5, …, a4, a3, a1] and this list was repeated forever.

Technique T4 is behavior capture with random action sequence generalization. An action is selected randomly, ignoring training traces. The next action to perform is: `random{a1, a2, … , a14}`.

Technique T5 is behavior capture with random trace sequence generalization. It picks a random trace from available training traces, and performs the actions in the chosen trace order. Therefore the next three actions in order are: `random{[a5, a7, a6],[a1, a4, a3],[a6, a8, a5],…,[a4, a3, a1]}`. Technique T5 is just one of the parameterized HMM generalizations. Since an HMM remembers its previous state information through the transitional probabilities between pairs of hidden states, with enough hidden states, the HMM can remember the exact order of the training traces and reproduce actions in the same order as the training traces.

Technique T6 is behavior capture with HMM sequence generalization and it uses an HMM with eight hidden states.

In addition to the proactive behaviors, each patron was trained to perform one reactive behavior: face the collaborator, walk to the collaborator, listen and then speak. This behavior is used to respond to the proactive collaborative converse behavior. Patron training also included one latent behavior: *when the bard exits the stage, face the bard and cheer*. A multi-queue behavior architecture was used so that the latent behavior would interrupt any proactive behavior and the interrupted behavior would be resumed after the latent behavior was completed [5]. The bard was trained with two proactive behaviors. The first was: *sing*, *wait* and *leave the stage*. The second was: *return to the stage*.

We had two main hypotheses. First, that behavior capture would rank higher than the *no behavior* and *manually scripted behavior* techniques. Second, that sequence generalization would be a factor in the perception of believable characters. Specifically, that the order of behavior capture rankings would be: HMM, random traces, random actions and no sequence generalization.

5.2 Preliminary User-Study

We conducted a preliminary user study to evaluate the effects of the number of training traces. We wanted to determine how the number of traces would influence user perceptions. The goal was to determine if users could distinguish between a very small number of training traces (4) and a larger number (9). The user study was constructed similarly to the main user study that is described in the previous section, except that scene variations with 4 training traces were compared to scene variations with 9 training traces. The results show that with 95% confidence the variations with higher numbers of traces produced more believable scenes. This is an expected trade-off between quality and workload. Based on this result, we set the number of training traces in our main user-study closer to the higher number (we selected 10 traces).

5.3 Results

In the main user study, there were 27 participants. Unfortunately, a few participants did not answer carefully enough for their responses to be considered valid, with some participants not answering some questions and some participants providing what seemed to be random answers (e.g. ranking 1,2,3,4,5,6 for all criteria). Therefore, the results of each questionnaire were validated for self-consistency. One question asked the respondent to rank the six variations according to overall believability, while another question asked the respondent to rate the six variations individually on a scale of 1 to 4 according to overall believability. To ensure that the participants answered the questions carefully, we removed a questionnaire if the rankings and ratings contained more than one inconsistency between the rating and ranking questions. After this consistency check, a total of 21 valid questionnaires remained.

Table 2 shows the average technique rankings for the 6 techniques. Each number represents the average ranking for the particular technique for the particular criteria over the 21 responses. For example, technique T6 is ranked 4.29 on average (where 6 is the highest ranking) among the 6 techniques, according to the criteria *active characters*. The results show that in general, technique T6 is ranked highly for all the criteria except *unpredictable characters*. Note that high rankings are better than low ones for all criteria except *unpredictable characters* and that the *overall believability* is not an average of the other criteria. It was a separate question on the survey.

Table 2. Average Technique Ranking (6 is Highest, 1 is Lowest) and Average Overall Rating (out of 4). Higher numbers are better in all criteria except *unpredictable characters*. Standard deviations are shown in parentheses.

Criteria	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
active characters (ranking)	1.19 (0.60)	3.38 (1.72)	3.90 (1.67)	4.29 (1.23)	3.95 (1.53)	4.29 (1.06)
unpredictable characters (ranking)	3.00 (2.24)	3.29 (1.65)	4.48 (1.60)	3.52 (1.50)	3.62 (1.60)	3.10 (1.34)
plausible sequences (ranking)	2.00 (1.45)	2.38 (1.63)	3.57 (1.57)	4.38 (1.40)	4.29 (1.35)	4.38 (1.20)
diverse actions (ranking)	1.24 (0.54)	3.29 (1.68)	4.10 (1.67)	3.86 (0.96)	3.81 (1.66)	4.71 (1.10)
overall believability (ranking)	1.33 (0.80)	3.05 (1.69)	3.67 (1.28)	4.00 (1.26)	4.10 (1.61)	4.86 (1.15)
overall believability (rating)	1.20 (0.41)	2.05 (1.16)	2.19 (0.87)	2.71 (0.90)	2.57 (0.87)	2.85 (0.59)

We used a Friedman statistical test compared each row of Table 2 to avoid the alpha-inflation effect. It indicated that there are significant differences in the average rankings of the six techniques for each criterion at the 95% confidence level. We used a Friedman test instead of ANOVA because of the ranked data. Based on the positive result of the Friedman test, T-tests were used to compare pairs of rankings.

Table 3 shows the p-values of subsequent T-tests between the average rankings of technique T6 versus each of the other techniques. The most obvious result is that T6 was ranked significantly higher than T1 and T2 in all aspects except unpredictability. This study indicates that hand-scripted characters are perceived as less diverse, less plausible and have less active characters than character behaviors generated using behavior capture with HMM sequence generalization. The study also shows that T6 ranked significantly higher than all other tested techniques for overall believability.

It is perhaps surprising that T6 was perceived as significantly better for overall believability compared to T3, T4, and T5, even though T6 was not perceived as significantly better on some of the four component criteria. This indicates that either there is a missing criterion that is necessary for overall believability or that believability cannot simply be decomposed into parts based on criteria. This is a crucial question in trying to measure believability, as pointed out by MacNamee [7], who endorsed the evaluation of aspects of believability rather than overall believability to reduce subjectivity. To evaluate the importance of our criteria, we asked participants to rate the importance of each of the four criteria. Table 4 show the average importance computed from the responses of the study participants. As expected, unpredictability is the least important in the eyes of the participants.

Table 3. p-values (to two decimals) from T-tests on Technique T6 versus each other technique for each criterion. Entries with a dark background are significant at the 95% confidence level.

Criteria	T1 vs. T6	T2 vs. T6	T3 vs. T6	T4 vs. T6	T5 vs. T6
Active	**0.00** ■	**0.05** ■	0.22	0.50	0.20
Unpredictable	0.44	0.37	**0.01** ■	0.09	0.14
Plausible	**0.00** ■	**0.00** ■	**0.04** ■	0.50	0.40
Diverse	**0.00** ■	**0.00** ■	0.13	**0.00** ■	**0.02** ■
Overall	**0.00** ■	**0.00** ■	**0.00** ■	**0.03** ■	**0.03** ■

Table 4. The average importance of the four criteria. Participants rated each criteria on a scale of -3 to 3. A positive number means important in contributing positively to overall believability. A negative number means important in contributing negatively. The larger the absolute value the more important it is.

Criteria	Average Importance
active	1.76
unpredictable	0.71
plausible	2.19
diverse	1.52

We will conduct studies on different variations and a larger participant pool to increase confidence that behavior capture generates significantly better behaviors.

6 Conclusion

The video game industry continues to grow and game designers are becoming more specialized in their own areas. In addition, recent story-based video games have started providing tools so that non-professionals can design their own stories. However, in order to successfully use these tools, a designer needs to know programming, since characters in game are controlled by programmed scripts.

In this paper we propose a new tool for game designers that allow them to create behaviors for virtual characters, without having to learn complicated programming.

Using an analogy to motion capture, we propose a data-driven method of creating behaviors for NPCs by behavior capture. Game designers take control of a particular NPC and perform the desired behaviors for this NPC. Using category-based generalization for characters and objects, and Hidden-Markov Models for sequence generalization, our behavior capture technique produces a variety of behaviors learned from the training traces. We performed a user study to confirm that behavior capture produces behaviors that are perceived as significantly superior with regards to character activity level, unpredictability and behavior diversity compared to the scripted behaviors seen in a typical commercial story-based game. This study also showed that using HMMs for sequence generalization, instead of raw behavior traces contributes significantly to perceived overall believability. For future work, larger user studies with more participants and different scenes should be conducted to provide move evidence of the utility of behavior capture and more insight into the best technique for sequence generalization.

References

1. AiLive. http://www.ailive.net/
2. AiLive. AiLive LiveCombat: Artificial intelligence and behavior capture for games. http://www.youtube.com/watch?v=u8oNTLzCFNU
3. Baum, L. E., Petrie, T., Soules G., and Weiss, N.: A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. Ann. Math. Statist., vol. 41, no. 1, pp. 164–171 (1970)
4. Cutumisu, M.: Using Behavior Patterns to Generate Scripts for Computer Role-Playing Games. Ph.D. thesis. University of Alberta (2010)
5. Cutumisu, M. and D. Szafron.: An Architecture for Game Behavior AI: Behavior Multi-Queues. Proceedings of the Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE), Stanford, USA, 20-27 (2009)
6. Gleicher, M.: Animation from observation: Motion capture and motion editing. ACM SIGGRAPH Computer Graphics.33, 4 (2000)
7. MacNamee, B.: Proactive Persistent Agents: Using Situational Intelligence to Create Support characters in Character-Centric Computer Games. PhD Thesis. Trinity College Dublin (2004)
8. Mahlmann, T., Drachen, A., Togelius, J., Canossa, A., Yannakakis, G.N.: Predicting Player Behavior in Tomb Raider: Underworld. In Proceedings of the IEEE Conference on Computational Intelligence and Games (2010)
9. Ontanon, S., Bonnette, K., Mahindrakar, P., Gomez-Martin, M. A., Long, K., Radhakrishman, Shah, J., Ram, R.: A. Learning from Human Demonstrations for Real-Time Case-Based Planning. In the IJCAI-09 Workshop on Learning Structural Knowledge From Observations (2009)
10. Orkin, J. and Roy, D.: Automatic Learning and Generation of Social Behavior from Collective Human Gameplay. International Conference on Autonomous Agents and Multiagent Systems (2009)
11. Thurau, C., Bauckhage, C.: Analyzing the Evolution of Social Groups in World of Warcraft. In Proceedings of the IEEE Conference on Computational Intelligence and Games (2010)
12. Thurau, C., Bauckhage, C., and Sagerer, G.: Imitation learning at all levels of game-AI. In Proceedings of the International Conference on Computer Games, Artificial Intelligence, Design and Education. University of Wolverhampton. 402–408 (2004)
13. TruSoft: Artificial Contender. http://www.trusoft.com/principles.html