Toward Gesture-Based Behavior Authoring

Edward Yu-Te Shen* Bing-Yu Chen†

National Taiwan University

ABSTRACT

Creating lifelike, autonomous, and interactive virtual behaviors is important in generating character animation, such as animal crowds, pedestrians, battle scenes, etc. Unfortunately, such task has long been limited to skilled users, since the authoring tools, including script languages and other commercial programs, mostly require lengthy prelearning process or are difficult to use. A novel approach, gesture-based behavior authoring, is proposed to open the interesting experience of creating autonomous animated characters to novice users. The technique enables users to efficiently prototype behaviors of a character, with the potential for further refinements. With several testees, our gesture-based authoring manner has been verified to be beneficial to the addressed problem , and, grounded on the HCI (Human-Computer Interaction) literature, the authoring process is direct, easy, and enjoyable.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; H.5.2 [Information Interfaces and Presentation]: User Interfaces—Graphical user interfaces (GUI), Interaction styles, Prototyping

Keywords: behavior authoring, mouse gesture, end user programming, sketch interface, autonomous characters, crowd animation

1 INTRODUCTION

Modeling the behaviors for virtual characters has played an important role in the production pipelines of modern graphics-related industries: from crowd scenes in animated films to individual AI warriors in computer games. Unfortunately, behavior modeling has long been a difficult and cumbersome task. To enable non-programmers to create such "behavioral animation" [3, 27, 32] more easily, a large amount of work have been proposed in the research community [14, 25], as well as in the commercial domain[1]. These techniques, although attempted to be user-friendly and powerful, still ask for "literate users" who are familiar with finite state machines. Some of them even require lengthy tutorial processes. Novice users, including artists, storytellers, and others with potential for building such behavioral animation, are still kept from the developing processes.

To enable more users to participate such a task, we propose a gesture-based authoring technique for creating virtual characters' behaviors. The authoring process functions as users draw series of gestures directly on the characters in the simulation window, as if they were uttering sentences to the characters by combining vocabularies. Because of the sequential way of using the gestures and their high flexibility in shape, mouse gesture is an expressive input means. As a result, the proposed approach is capable of expressing conditional properties (in respect of the relationships between actions and perceptions) as well as the characters' inner, outer, and mutual characteristics, which have been major challenges for building a graphical behavior authoring tool.

Our design philosophy follows the ideas of easy, rapid construction of approximate models as proposed in Teddy [16]. Basically, our technique is for prototyping fairly acceptable behaviors as opposed to careful, precise specification. We regard it worthwhile to sacrifice a small degree of users' control over the characters for the strong accessibility the sketch-based interface brings. Furthermore, to ensure the potential of the prototyped behaviors, our system is capable of automatically rendering the results into XML or other formats, as shown in Figure 5, for further refinements by skilled users or programmers.

The main contribution of this paper is the gesture-based authoring approach that allows users to manipulate behaviors directly on virtual characters without requiring any additional behavioral representations (e.g. script code, graphs, etc.). We name this approach as "What You Do Is What You Get" (WYDIWYG), and its groundings include Csikszentmihalyi's "flow" [2, 9] and Shneiderman's "direct manipulation" [28]. Our approach alleviates users from the difficulties of converting between those representations and the characters' actions, as well as the need for comprehending the computational logic. The elimination of inter-window navigation also concentrates users on the virtual characters, and the tedious authoring task has become an easy, immersive, and interactive experience. To evaluate our method, a usability test is also included in this paper.

2 RELATED WORK

A notable investigation of generating behavioral animation is Reynolds' superb work [27], in which flocking behaviors of birds and fishes is simulated with a graceful approach. Following Reynolds, many works were proposed on crowd animation or the navigating behaviors [30, 24]. Works focus-ing on locomotion have also been presented, including the animation of worms' and snakes' motion [23], birds' flight [6], general articulated 3D virtual creatures [29], and fishes [32]. Brooks [4], Meyer [22], Bryson and Stein [5], and other researchers have made great effort on behavioral structure design too. As for tools that help creating behavioral animation, they are mostly languages and scripts, such as Improv [25], Cognitive Modeling Language (CML) [14], and many others [1, 17]. Basically, all these divisions benefit programmers with ways of designing systems. In contrast, our goal is to empower end-users to modify the behaviors easily.

Incorporating graphical representation for finite state machines is also a popular solution, and can be found in [13, 29]. These systems have made the interface more user-friendly by visualizing the edited behaviors with linked graphs. However, they still require users' comprehension of logical programming, which is an important reason that they are still difficult to many people, even though they were meant to benefit non-programmers.

Machine learning is another way to simplify the task of generating behavioral animation. Through reinforcement learning, the virtual dog in [3] gradually learns to perform new behaviors. Many other works also proposed similar ways to animate virtual charac-

*e-mail: edwards@cmlab.csie.ntu.edu.tw

†e-mail:robin@ntu.edu.tw

[1]Massive Software - http://www.massivesoftware.com/

ters [12, 32, 6]. Despite reducing users' loads of detailed specifications, meanwhile these systems have eliminated users' control over the behaviors. Our approach, in contrast, reserves both users' capabilities of authoring the behaviors and the ease of use.

With respect to the interface, mouse gestures, or sketches, have been applied to various areas because of their expressiveness and user-friendliness, including web navigation[2], editing operations [18], math calculation [19], shaping, modeling, animating in 2D [15, 31], and in 3D [7, 11, 16]. We also take advantage of this superior interface, while the major difference of our approach is that we take gestures as a set of narrating tool. That is, complex meanings are depicted by gestures with temporal orders.

A great deal of effort has been made toward opening the interesting experience of creating behavioral animation to novice users by researchers in the end-user-programming area. Graphical Rewrite Rules (GRR) is one of the methodologies adopted by this field [10, 26]. Although GRR enables users to edit the "before state" and the "after state" for each conditional statement in a fully graphical environment, the lack of well-designed computational structures prevents one from inputting complete, sequential behaviors, even though "it is generally easier to describe the desired behavior in terms of sequences of events" [5]. The unacceptability of generalized rules also made users input similar rules repeatedly. Our method shares a common goal with the end-user-programming area, while avoiding the above shortcomings through applying sequential gestures and an AI behavioral model.

Other systems such as Alice [8] bridge across different categories and deliver multiple benefits. Particularly, Alice developers tried to achieve the simplicity of use without losing the functional complexity by letting users to create textual programs in a drag-n-drop fashion. Although Alice has been considered as a practice of "direct manipulation," unfortunately, its intrinsic nature, a scripting language, still asks the users to *indirectly code* the behaviors in their minds during the creation of the animation, *rather than to manipulate the characters directly*. Additionally, Alice is used to create procedural animation, which differs from our goal of behavior authoring.

3 Authoring Behaviors with Mouse Gestures

Figure 1 shows the authoring and simulating environment for virtual characters' behaviors. The authoring process begins when the users draw gestures on a certain type of character, and lasts until it is completed or interrupted by pressing the DELETE key. For each type of character, animation clips created by animators in advance are listed on the right for users to select, and the "behavior icons", user-created symbolizations of the behaviors for later access, are listed on the left.

The authoring and simulating environment is comprised of four main elements: 1) the mouse gestures, 2) the behavioral structure, 3) the authoring interface, and 4) the simulation engine. Briefly speaking, the authoring interface manipulates the characters' behavioral structures according to users' mouse gestures, and the behaviors will be simulated by the simulation engine according to these behavioral structures.

3.1 The Mouse Gestures

We define a mouse gesture as a mouse action which has a predefined meaning, such as single-clicks, double-clicks, dragging an animation clip, and drawing a gesture. The designed gestures are listed in Table 1. The gesture shapes and names are in the first two columns, and the meanings are in the third one.

[2]Mozilla - http://www.mozilla.org/

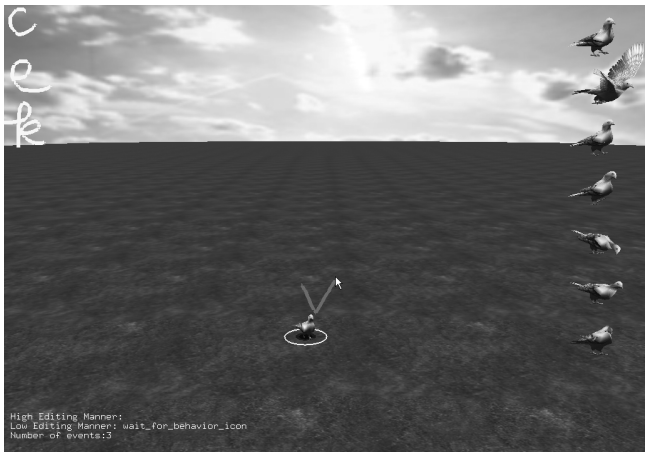

Figure 1: Our gesture-based behavior authoring system.

Gestures are input sequentially. A gesture in a sequence can be regarded as a word in a sentence in our daily conversation. For proper interpretation of the gestural sequences, five "grammars" are defined, as listed in Figure 2. These grammars depict users' intentions throughout gesture sequences, including, in a top-down order, building a new behavior, deleting a behavior, editing an existing behavior, building a new behavior which is related to a target, and building a new behavior for each of the grouped character in terms of interaction. For the "Build Action" nodes, Figure 3 illustrates the possible combinations of different gestures. All input sequences must match one of the predefined grammars, otherwise will be ignored.

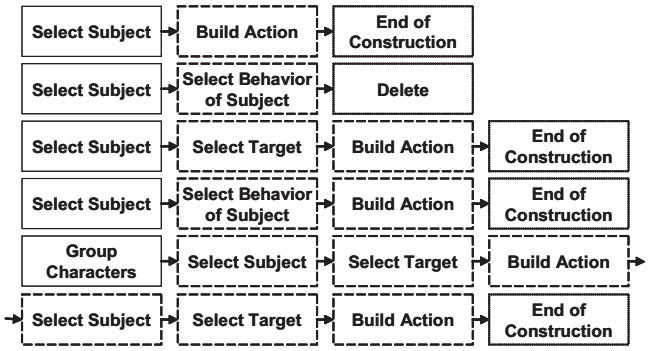

Figure 2: The Grammars.

3.2 The Behavioral Structure

Figure 4 illustrates our behavior structure, exampled by a virtual pigeon's forage behavior. Following the ideas suggested by previous research on behavioral structures [3, 4, 5, 22], our behavior is defined as a process toward the satisfaction of an intention (the solid-line region). The process consists of several stages (the dashed-line regions), and each of the stages contains its respective actions, perceptions, and, optionally, influences of the actions (e.g. destructions of other characters). The perceptions of a stage function as the preconditions of this stage. A behavior also has its own set of preconditions, namely, the "ground perceptions" (the green region), usually identical with those in the first stage. In most cases, there is one action in a stage, though this is unnecessary. The virtual pigeon on the ground may eat feeds in three different directions - front,

Table 1: The Gestures.

Gesture Trajectory	Gesture Name	Gesture Meaning
N/A	single click	To select a target if a subject character is selected.
N/A	double click	To select a subject character.
N/A	drag navigating animation	To specify the animation for a navigating action.
N/A	drag action animation	To build an action with this animation.
	X	To destroy something.
	arrow	To indicate a direction toward or away from something.
	check	To finish the construction of subject character's new behavior.
	circle	To group characters.
	wandering sign	To indicate the subject character to wander through its world with no specific destination.

right, and left. Thus, three different actions with their respective preconditions and animation clips may concurrently exist in this stage.

The behavioral structure is vital to the gesture-based authoring approach. This is because, after interpreting the individual as well as the combined meanings of the input gestures, *only according to this computational structure can the system transform these interpretations into the behavior's finite state machines*.

Details about actions and perceptions in our behavior authoring fashion are detailed below:

Actions: An action consists of a steering style, an animation clip, and a means of navigation. The steering styles, originally proposed by Reynolds [27], include wandering (moving aimlessly), pursuit (chasing some target), fleeing (escape some target), etc. The steering style of an action is determined according to the input gestures, and an action without a steering style simply means that this action is performed without navigation. The animation clip is the necessary and sufficient condition of the construction of an action. All animation clips need to be provided prior to the author-ing process with their navigation means. The means of navigation includes flying, walking or running, and swimming.

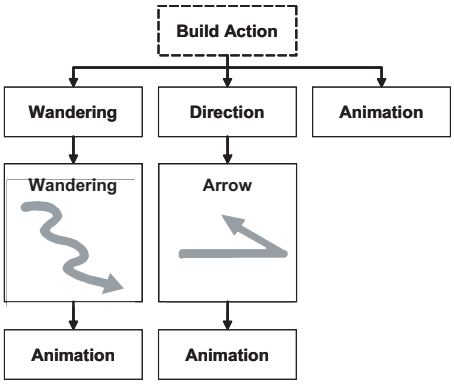

Figure 3: The possible combinations of different gestures.

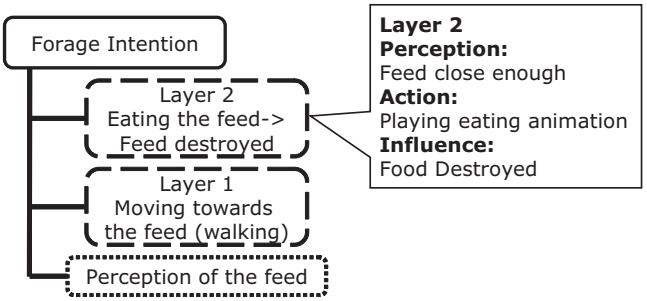

Figure 4: The Behavioral Structure.

Perceptions: A perception contains a perception type, a target type, a radius, an axis angle, and a range angle. Perception types include 0) none, 1) terrain, and 2) target, which are used to denote the object this perception is concerned with. Target types indicate the types of the character, such as a feed, a pigeon, a dog, etc. The remaining components are used to describe the range of this perception. The radius means the farthest distance this perception reaches. The axis angle, which suggests the direction, is defined as the angular difference between the character's front direction and the direction of this percept. The range angle, finally, indicates the angular range of the fan-shaped area spread from the axis. All components of a perception are specified while the action in the same stage is built during the authoring process.

3.3 The Authoring Interface

We incorporated the toolkit built by Boukreev[3] in our system as the gesture recognizer. When the user inputs a mouse gesture, the gesture will be first processed by the recognition system, stored into a history table, and then applied to modify the character's behaviors. Here we introduce four specific tasks: 1) validating each input gesture, 2) determining and executing each gesture's function, 3) properly displaying the gestures' trajectories, and 4) rendering the behaviors into a textual presentation.

First, a new gesture is verified by both its screen location and its order in the sequence. Each gesture must be close to a character, an animation, or a behavior icon, and all input sequences must match one of the five grammars in Figure 2. An invalid gesture would be discarded.

[3]http://www.codeproject.com/cpp/gestureapp.asp

Second, the function of a gesture is determined by both its own meaning and how it is used. For example, drawing an 'x' sign on a character destroys the character, while draw-ing it on a behavior icon means to delete an existing behavior. Whether an arrow sign is drawn from or toward an item also carries different meanings.

The execution of an identified gesture function includes two steps. The first one is the immediate visual response. To create a pigeon's "forage" behavior in our video, for example, the pigeon starts walking toward a target right after the walking animation is dragged onto an arrow. Such response is used only to clarify the gestures' effects interactively, whereas the modification on the behavioral structures is not performed until later. The second step, background modification, is performed when the whole input sequence is confirmed as valid. In this case, the "walking-toward-feed" action is stored into a history table when the system interactively responds to the user input, but won't be added into the forage behavior until the whole sequence has been completed and validated.

When a gesture suggests the completion of a sequence (e.g. a check sign for a new behavior's construction), all missing details for the edited behavior will be automatically specified with default values. For a newly-created behavior, the system will ask the user to input a behavior icon and list it on the left of the screen. The edited behavior will be applied to all characters of the same type in the world, which enables users to generate crowd animation easily.

To properly display the trajectories means to map the 2D path of mouse into 3D so that the trajectories will look properly when users move the camera. Currently, trajectories, including arrows, wandering signs, circles, etc, are mapped directly on the ground in the virtual world.

Finally, Figure 5 is the text representation of a behavior generated automatically by the system. Currently, we are using our own tagged format. But for more practical use, it could be converted into other standardized formats that are used in script for commercial applications.

3.4 The Simulation Engine

For each time step, the simulation engine selects the most suitable behavior and stage for each of the characters, according to the environment, their perceptions, and inner properties. Following [3] and [32], the intentions of all behaviors in our system increase with time. Whenever the intention exceeds a particular threshold, the system checks whether the context has met a behavior's ground preconditions. The forage behavior, for example, is selected only when a food is perceived by the pigeon. When more than one behavior's preconditions are met, the one to be performed will be randomly selected, with the behaviors' probabilities proportional to their intention values.

After a behavior is selected, the character will proceed to one stage after another whenever its respective precondition is satisfied; and a behavior is completed as the action in the top stage is carried out. That is, the pigeon will not walk towards the feed until detecting it; it will not eat and destroy the feed until arriving at the feed; and the forage behavior will be completed when the feed is eaten and destroyed. If a precondition fails to maintain during the execution of a behavior, the procedure will trace backwards and resume from the highest stage where the preconditions are held true. In other words, if the feed approached by a pigeon is eaten by others interruptedly, the pigeon will trace back and check if any other feeds exist. If even the ground perceptions are not met by the context, i.e. the pigeon can perceive no feed at all the system will re-select the behaviors to be performed through the randomized algorithm.

The above describes how the behaviors and actions are selected. As for characters' navigation, we achieve this task with ease by using Reynolds's OpenSteer library[4]. Finally, the animation is rendered with OpenGL. Note that the available animation clips only describe specific actions; thus, at all time, both during the authoring and simulating process, the playback module verifies whether to insert transitions between consecutive actions. For instance, when a user tries to make a standing puppy sit, he/she may drag the sitting animation onto the puppy, and the transition from standing to sitting is inserted automatically for efficiency. The consistency between each animation pair is predefined by the animators.

```
<Behavior>
  CharacterName                 Dove
  BehaviorName                  Forage
  Intention                     90.0
  <GroundPercepts>
    <Percept>
      Radius                    20.0
      PerceptType               1
      TargetType                Food
    </Percept>
  </GroundPercepts>
  <BehaviorLayer>
    <PrimaryPair>
      <Percept>
        Radius                  20.0
        PerceptType             1
        TargetType              Food
      </Percept>
      <Action>
        AnimationType           walking
        NavigMeans              walk
        SteeringStyle           pursuit
      </Action>
    </PrimaryPair>
  </BehaviorLayer>
  <BehaviorLayer>
    <PrimaryPair>
      <Percept>
        PerceptType             1
        TargetType              Food
        Radius                  0.4
        AxisAngle               0.0
        RangeAngle              40.0
      </Percept>
      <Action>
        AnimationType           eating
        Influnce                Destroy
      </Action>
    </PrimaryPair>
  </BehaviorLayer>
</Behavior>
```

Figure 5: An example of the generated text representation.

4 Authoring the Behaviors

In this section, we show how users can author the behaviors with mouse gestures. Within about 10 minutes, six behaviors (e.g., cleaning self, forage) for the pigeons and four for the dogs (e.g., chasing) can be created using our authoring system. Together, the two kinds created a lively crowd scene, as shown in Figure 6.

Here we discuss about the detailed authoring process with two examples. The first example, building a forage behavior of a pi-

[4]http://opensteer.sourceforge.net/

Figure 6: The result simulation scene.

geon, is to demonstrate the authoring process of a multi-stage behavior. It consists of a ground perception (perceiving a feed), two stages of actions (walking toward the feed and eating it), and a consequence (destroying the feed). The second example, building a chasing-fleeing behavior, shows how to create interactions between two characters, which is an important task in behavior authoring.

4.1 Building a Forage Behavior

The process of building a forage behavior is shown in Figure 7. The construction begins as we select the subject character (the pigeon) by double-clicking with the left button, and then select the target (the feed) with a single-click. The authoring interface thus builds a ground perception describing the target-type (i.e. feed) and its relative position, as well as a terrain-type ground perception.

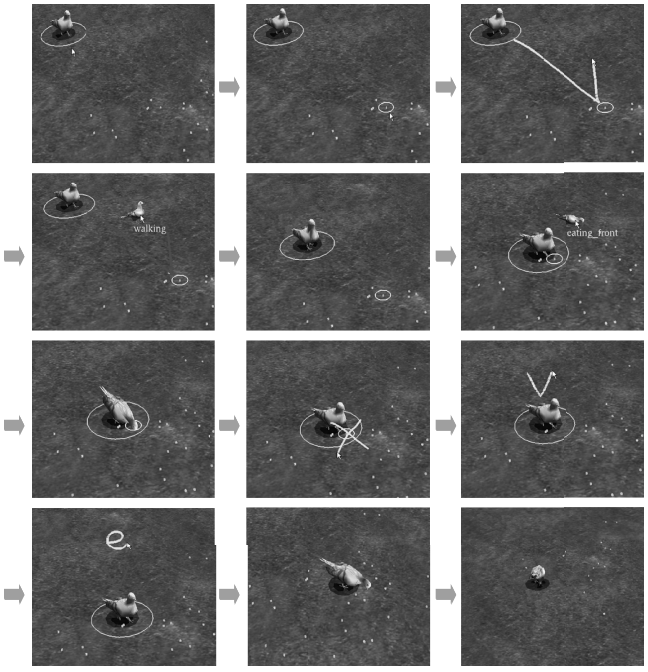

Figure 7: Authoring the forage behavior.

When the pigeon reaches the feed after the walking animation is dragged onto the arrow, A stage is constructed with a moving action with a specified animation clip, and a perception concerning about the target-type and its relative direction (instead of the absolute direction in the world).

Another stage will be created after we drag the eating animation, and it has a new perception that the target position is relatively closer, and a new action with the eating animation. Then, the X sign on the target is to denote the consequence of the eating action - destruction of the feed, and a left-click is made on the feed before the X sign to suggest a coming manipulation on the target.

Finally, after drawing a check gesture to finish the construction, we draw an "e" as the behavior icon, and the behavior is successfully created. Thus, whenever a feed is perceived, the pigeon will move toward it, eat it, and then the feed will be destroyed. If the feed is right in front of the pigeon, the pigeon will eat it without approaching, because it is already close enough.

4.2 Building a Chasing-Fleeing Interaction

The second example builds an interaction between two characters (Figure 8). After being grouped, the two characters are marked by blue circles as interacting with one another. Then, we take turns to reverse the roles of the dog and the pigeon as the subject and the target. First, the dog is selected as the subject, and the pigeon the target. While we drag the animation onto the arrow and construct the running-toward-pigeon action, the dog does not move toward the pigeon immediately. This is designed for the convenience of users' authoring process. Nevertheless, the running animation is played as if the dog is eager to run toward the pigeon, so the user can get the feedback that the manipulation works. Then we select the pigeon as the subject and the dog as the target. We create the pigeon's fleeing behavior, triggered by its detection of the dog. Finally, the behavior icon is drawn once and will be shared by the two characters, but the meanings are different. The most distinguished part of this example from the previous one is that each of the characters will have a different behavior, and two text files will be generated.

5 Evaluation & Discussion

The final part of this paper is the evaluation and discussion. We first introduce the fundamentals we base on - two major theories from the Human-Computer-Interaction (HCI) literature - and then describe about the usability test we made and its discussion. By combining both qualitative and quantitative evaluations, we claim that the superiority of the gesture-based behavior authoring manner is verified.

5.1 Groundings: The HCI Literature

To make behavior authoring truly accessible to the novices, we tried to stand from users' points of view by examining the HCI and psychology literature. Based on *flow* [2, 9] and *direct manipulation* [28], we have enabled users to focus solely on the animated characters throughout the authoring process by combining the editing interface with the simulation window. This "What You Do Is What You Get" (WYDIWYG) strategy has become the core value of our approach.

Flow. Mihaly Csikszentmihalyi used the phrase "being in the flow" to picture people's full engagement in activities when challenges and skills are balanced [9]. Later, Bederson [2] bridged this idea to interface design later to highlight that interfaces should target at keeping users in the flow by avoiding interruption and supporting creativity and enjoyment. Accordingly, we conclude two important properties for an effortless behavior authoring manner.

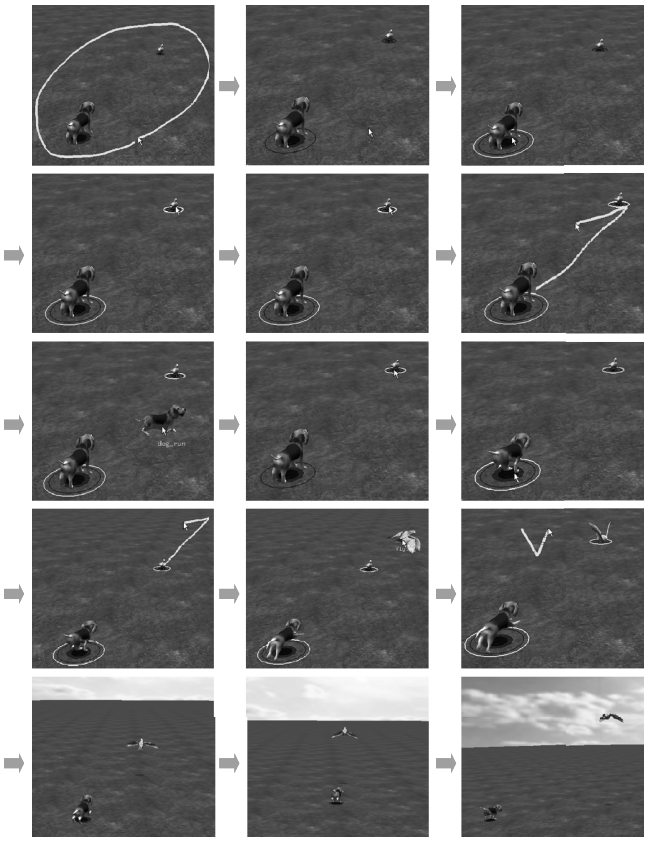

Figure 8: Authoring the chasing-fleeing interaction.

First, since users have extremely limited short term memory [2], asking users to navigate among windows to compose behavioral descriptions may strain users' memory and interrupt their flow. Unlike other tools that require inter-window navigation [8, 29], our behavior authoring tool keeps users focus on the characters and eliminates interruptions. This is done by combining the editing interface with the simulation window. Second, immediate feedback allows users to be clear about the consequences of their actions, as oppose to leaving them uncertain until the "play" button is pressed at the end (e.g., [8, 29]). More importantly, immediate feedback reinforces users to engage in the authoring processes, keeping them enlightened interested throughout the activity.

Direct Manipulation. Ben Shneiderman proposed to bring users comprehensible, predictable, and controllable interfaces with visual representation of object and action of interests, physical actions instead of complex syntax, and rapid reversible operations [28]. His assertion, "The closeness of the task domain to the interface domain reduces operator problem-solving load and stress," has provided a strong foundation for our WYDIWYG strategy. Explicitly, because the characters are simultaneously authored and simulated, *the users need neither to transform the behaviors into an abstract representation (e.g. code or graphical state machines) nor to convert the code back to the behaviors in minds while reviewing the algorithms*.

5.2 Usability Test

To verify that the proposed idea is favorable for behavioral animation creation, our system was tested with 14 users, including 13 males and 1 female. Most of these users are undergraduate students. None of these testees had written programs related to agents' behaviors, however all of them were made clear about the aim of this tool. The users first read a 2-page instruction describing introduction of the test, behavior authoring, the gesture table, and statements concerning the test policy and privacy issues. The test goes on with a 10-minute oral explanation, example demonstration and question answering, followed by a 20-minute usage of the system and the completion of a questionnaire.

The questionnaire consists of 15 questions (in Mandarin Chinese), and is based on the Computer System Usability Questionnaire[5]. From the following summarization of the result, one can get some idea about how the gesture-based authoring idea contributes to the research area:

- 92.8% of the people found the authoring process interesting or very interesting.
- 92.8% of the people agreed or very much agreed that the system's interface is pleasant to them.
- 85.7% of the people needed less than 30 seconds to choose the next gesture to use, in which 57% required less than 10 seconds.
- 85.7% found this tool easy to learn or very easy to learn.
- 78.5% liked or very much liked the system's interface.
- 71.4% of the people felt satisfied or very satisfied about how easy it is to use the system.
- 57.1% believed that himself/herself can become a skilled user very soon.

Nevertheless, the usability test also suggested us directions for improvements as well as users' main concerns during the authoring process. For example, almost half of the users disagreed with that he/she could efficiently edit the desired behaviors, even though from the above data we can see that, generally speaking, 1) the gestures to be utilized in each step can be selected shortly, and 2) the tool is quite easy to learn. As we investigated the reason of this dissatisfaction through asking the testees, the most frequent answer received was about the inadequacy of our recognition engine. That is, from time to time they needed to draw a gesture several times to make the intended input.

We think that this is problem exists mainly because traditional recognition algorithms, including the library we used, were not affine-transform-invariant, which unfortunately is something fatal to our system's performance. To clarify, let us take the arrow sign as an instance. When a user draws an arrow sign from a pigeon to one of the feeds on the ground, all possible arrows may be different in direction, size, and even length ratio between the arrow's two segments. To recognize a 2D gesture as a function depiction in a 3D virtual world, truly it may be difficult if the recognizer is not robust against such variations. In our implementation, an "arrow sign" was actually the union of a set of arrows with different directions, and all input gestures are auto-scaled before the recognition process. From this usability test, we are now aware the crucial role this recognition phase plays, and a more desirable engine will be an important future direction.

Another thing that needs to be addressed is the amount of gesture types. Some users felt that the provided gestures were not enough to use, as we expected. (After all, the mere gestures can no way be general nor adequate.) Nevertheless, an additional piece of information we wished to gain from this test is exactly how to design the proper gesture set. From the test, 85.7% of the people thought that the most proper amount of gesture types is under 15, in which 71.4% of the people feel less than 10 types of gestures would be

[5] http://www.acm.org/∼perlman/question.cgi

easier to use. A question thus arises. Since memorizing the gestures takes efforts and may even eliminate the benefits brought by gestures' direct characteristics, should we turn to some other interface primitives such as buttons, scroll bars, etc.? If we do, then only functions that need gestures' various superiorities will be formed as gestures, whereas others can be achieved with conventional interface primitives (e.g., it is actually fine to replace a check sign with a button-click). However, searching for the button is potentially harmful to the authoring flow because of long searching time, and the advantages of WYDIWYG strategy may also be traded off because some functions may not be performed *directly on the characters* any more. The best way to design the gesture set remains a puzzle, but here we introduce the major concerns in this topic, and we hope to invite more researchers to join the research field and to explore more about this enticing authoring manner.

6 Conclusion and Future Work

A novel gesture-based behavior authoring technique is proposed for prototyping behavioral without using code or graphical state machines. With this technique, users can edit the behaviors directly on the simulated characters. The authoring process is user-friendly and efficient, and the resulting behaviors can be automatically rendered into a text file for further refinements by skilled users or programmers. A usability test was done with 14 university students, and the result showed that the proposed technique has turned behavior authoring into an interesting, easy-to-learn, and pleasant interaction process.

Our first step toward gesture-based behavior authoring approach is promising. However, the gesture-based approach is still in its infancy. First, a recognition engine particularly designed for this purpose is called for. Second, we are searching for a more complete while compact set of gestures, that is, although not necessarily 100% complete, but applicable for a fair amount of cases. We also attempt to allow users to edit the created behaviors while they are simulated (i.e. on-the-fly programming), which, as we believe, will make debugging much easier. We hope to stay with the gesture-based approach and its advantages, as we continue to work on other new features. And, based on this work, we are also looking forward to that a more general, efficient, effective, and user-friendly tool for behavior authoring can be evolved.

7 Acknowledgement

We greatly appreciate James Davis, Erika Chuang, Yung-Yu Chuang, Jane Yung-Jen Hsu, Ming Ouhyoung, and Chao-Ming (James) Teng for their insightful suggestions to this paper.

References

[1] Yasmine Arafa, Kaveh Kamyab, and Ebrahim Mamdani. Character animation scripting languages: a comparison. In *Proc. AAMAS 2003*, pages 920–921, 2003.

[2] Benjamin B. Bederson. Interfaces for staying in the flow. *ACM Ubiquity*, 5(27), 2004.

[3] Bruce Blumberg, Marc Downie, Yuri Ivanov, Matt Berlin, Michael Patrick Johnson, and Bill Tomlinson. Integrated learning for interactive synthetic characters. *ACM Transactions on Graphics*, 21(3):417–426, 2002. (Proc. SIGGRAPH 2002).

[4] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

[5] Joanna J. Bryson and Lynn A. Stein. Modularity and design in reactive intelligence. In *Proc. IJCAI 2001*, pages 1115–1120, 2001.

[6] Jia chi Wu and Zoran Popović. Realistic modeling of bird flight animations. *ACM Transactions on Graphics*, 22(3):888–895, 2003. (Proc. SIGGRAPH 2003).

[7] Jonathan M. Cohen, Lee Markosian, Robert C. Zeleznik, John F. Hughes, and Ronen Barzel. An interface for sketching 3D curves. In *Proc. I3D 1999*, pages 17–21, 1999.

[8] Matthew J. Conway. *Alice: Easy-to-Learn 3D Scripting for Novices*. PhD thesis, University of Virginia, 1997.

[9] Mihaly Csikszentmihalyi. *Flow: The Psychology of Optimal Experience*. Perennial, 1991.

[10] Allen Cypher and David Canfield Smith. KidSim: end user programming of simulations. In *Proc. CHI 1995*, pages 35–36, 1995.

[11] James Davis, Maneesh Agrawala, Erika Chuang, Zoran Popović;, and David Salesin. A sketching interface for articulated figure animation. In *Proc. SCA 2003*, pages 320–328, 2003.

[12] Jonathan Dinerstein, Parris K. Egbert, Hugo de Garis, and Nelson Dinerstein. Fast and learnable behavioral and cognitive modeling for virtual character animation. *Computer Animation and Virtual Worlds*, 15(2):95–108, 2004.

[13] Daniel Fu, Ryan Houlette, and Oscar Bascara. An authoring toolkit for simulation entities. 2001.

[14] John Funge, Xiaoyuan Tu, and Demetri Terzopoulos. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. In *Proc. SIGGRAPH 1999*, pages 29–38, 1999.

[15] Takeo Igarashi, Sachiko Kawachiya, Hidehiko Tanaka, and Satoshi Matsuoka. Pegasus: a drawing system for rapid geometric design. In *Proc. CHI 1998*, pages 24–25, 1998.

[16] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3D freeform design. In *Proc. SIGGRAPH 1999*, pages 409–416, 1999.

[17] Sumedha Kshirsagar, Nadia Magnenat-Thalmann, Anthony Guye-Vuillème, Daniel Thalmann, Kaveh Kamyab, and Ebrahim Mamdani. Avatar markup language. In *Proc. EGVE 2002*, pages 169–177, 2002.

[18] James A. Landay and Brad A. Myers. Interactive sketching for the early stages of user interface design. In *Proc. CHI 1995*, pages 43–50, 1995.

[19] Joseph J. LaViola, Jr. and Robert C. Zeleznik. MathPad^2: a system for the creation and exploration of mathematical sketches. *ACM Transactions on Graphics*, 23(3):432–440, 2004. (Proc. SIGGRAPH 2004).

[20] James R. Lewis. IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *International Journal of Human-Computer Interaction*, 7(1):57–78, 1995.

[21] Henry Lieberman. *Your Wish Is My Command: Programming by Example*. Morgan Kaufmann, 2001.

[22] Jean-Arcady Meyer. The animat approach : Simulation of adaptive behavior in animals and robots. In *Proc. NPI 1998*, pages 1–21, 1998.

[23] Gavin S. P. Miller. The motion dynamics of snakes and worms. *ACM Computer Graphics*, 22(4):169–178, 1988. (Proc. SIGGRAPH 1988).

[24] Soraia Raupp Musse and Daniel Thalmann. Hierarchical model for real time simulation of virtual human crowds. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):152–164, 2001.

[25] Ken Perlin and Athomas Goldberg. Improv: A system for scripting interactive actors in virtual worlds. In *Proc. SIGGRAPH 1996*, pages 205–216, 1996.

[26] Alexander Repenning and Corrina Perrone. Programming by example: programming by analogous examples. *Communications of the ACM*, 43(3):90–97, 2000.

[27] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. volume 21, pages 25–34, 1987. (Proc. SIGGRAPH 1987).

[28] Ben Shneiderman. Direct manipulation: A step beyond programming languages. *IEEE Transactions on Computers*, 16(8):57–69, 1983.

[29] Karl Sims. Evolving virtual creatures. In *Proc. SIGGRAPH 1994*, pages 15–22, 1994.

[30] Mankyu Sung, Michael Gleicher, and Stephen Chenney. Scalable behaviors for crowd simulation. *Computer Graphics Forum*, 23(3):519–528, 2004. (Proc. EG 2004).

[31] Matthew Thorne, David Burke, and Michiel van de Panne. Motion doodles: an interface for sketching character motion. *ACM Transactions on Graphics*, 23(3):424–431, 2004. (Proc. SIGGRAPH 2004).

[32] Xiaoyuan Tu and Demetri Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. In *Proc. SIGGRAPH 1994*, pages 43–50, 1994.