CHASE: Character Animation Scripting Environment

Christos Mousas *
Dept. of Computer Science
Dartmouth College
Hanover, NH, USA

Christos-Nikolaos Anagnostopoulos †
Dept. of Cultural Technology and Communication
University of the Aegean
Mytilene, Greece

Abstract

This paper presents the three scripting commands and main functionalities of a novel character animation environment called CHASE. CHASE was developed for enabling inexperienced programmers, animators, artists, and students to animate in meaningful ways virtual reality characters. This is achieved by scripting simple commands within CHASE. The commands identified, which are associated with simple parameters, are responsible for generating a number of predefined motions and actions of a character. Hence, the virtual character is able to animate within a virtual environment and to interact with tasks located within it. An additional functionality of CHASE is supplied. It provides the ability to generate multiple tasks of a character, such as providing the user the ability to generate scenario-related animated sequences. However, since multiple characters may require simultaneous animation, the ability to script actions of different characters at the same time is also provided.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: character animation, scripting environment, scripting action, narrative generation

1 Introduction

Character animation can be characterized as a complex and time-consuming process. This is especially true when animating virtual characters based on key-frame techniques, as this requires prior knowledge of software solutions. Moreover, artistic skills are also required since the virtual character should animate as naturally as possible.

In order to avoid time-consuming processes in animating virtual characters, motion capture technologies now provide high quality and realistic animated sequences. This is possible because the ability to capture real humans in the act of performing is achieved through the provided required motions. The advantages of motion capture techniques are numerous, especially in the entertainment industry. However, the captured motion data, itself, it is not always usable, since virtual characters should be able to perform tasks in which the required constraints are not always fulfilled. Thus, methodologies that retarget [Gleicher 1998], warp [Witkin and Popovic 1995], blend [Kovar and Gleicher 2003][Park et al. 2002], splice [Van Basten and Egges 2012], interpolate [Kovar et al. 2002][Mukai and Kuriyama 2005] etc., the motion data have become available to help the animators to create the required motion sequences. In addition to the motion synthesis techniques that are based on software solutions, animating a virtual character through programming is also difficult. This is especially true in cases where animators, artists and students do not have the required programming skills. Hence, animating virtual characters in order to visualize ideas and generate simple scenarios in which virtual characters evolve can be a very complex process.

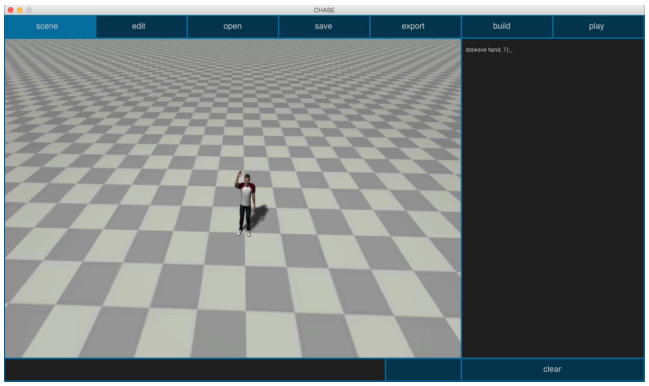

Figure 1: The interface of CHASE.

Based on the aforementioned difficulties that inexperienced programmers can face, this paper introduces a simple, easy-to-use, scripting environment for animating virtual characters, which is based on a small number of scripting commands. The scripting environment presented (see Figure 1), which is called CHASE, provides a user with the ability to script the action of a character as well as to script possible interaction between a character and objects that are located within the virtual environment [Mousas and Anagnostopoulos 2015].

In order to implement CHASE the following parts were developed. Firstly, identifying the basic actions that a character should be able to perform and also generating the basic scripting commands. Secondly, a number of parameters that should allow the user not only to synthesize the required motion of a character, but also to gain a higher level of control of each action of the character were defined. By using a rich number of motions that a character can perform, as well as by associating these actions with specified keywords, a motion dataset is created. The input commands are handled by a number of developed background algorithms, which are responsible for retrieving the desired motions and synthesizing the requested actions of the character. During the application's runtime, CHASE synthesizes the requested motion of the character and displays the final animated sequence.

The remainder of this paper is organized as follows. Section 2 covers related work in character animation by presenting previous solutions for animating virtual characters that are based on interactive or automatic techniques. Previously developed scripting environments for the animation of virtual characters are also pre-

*e-mail: christos@cs.dartmouth.edu
†e-mail: canag@ct.aegean.gr

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Request permissions from Permissions@acm.org.
VRCAI '15, October 30 – November 1, 2015, Kobe, Japan.
© 2015 ACM. ISBN 978-1-4503-3940-7/15/10…$15.00
DOI: http://dx.doi.org/10.1145/2817675.2817677

sented and discussed. A system overview of CHASE is presented in Section 3. The script commands, possible parameters, and additional functionalities that have been developed for CHASE are presented in Section 4. The results, which indicate the potential use of a scripting environment by users who are inexperienced in programming, are presented in Section 5. Finally, conclusions are drawn and potential future work is discussed in Section 6.

2 Related Work

This section presents work that is related to the solution presented. Specifically, the following paragraphs present methodologies that use different input devices or easily specified constraints for animating virtual characters, systems that provide to a user the ability to synthesize task-based or scenario-related animated sequences, and previously proposed scripting environments for character animation. Finally, the advantages provided by CHASE comparing by previous solutions are presented.

Interactive character control can be classified according to the input device that is used for the character animation process [Sarris and Strintzis 2003]. In general, the character controller can be a standard input device, such as a keyboard and a joystick [McCann and Pollard 2007]. Alternatively, it can be more specialized, such as text input [Oshita 2010], prosodic features of speech [Levine et al. 2009], drag and drop systems where the motion sequences are placed into a time-line [Oshita 2008], sketch-based interfaces [Davis et al. 2003] or the body of a user [Chai and Hodgins 2005], while the motion is captured by motion capture technologies. Each of the previously mentioned methodologies has advantages and disadvantages. The choice of the most appropriate input device depends on the actual control of the character's motion that the user requires.

A variety of methodologies for the animation of a virtual character based on easily specified constraints have also been examined. These solutions are based on motion graphs [Kovar et al. 2002] literature such as [Safonova and Hodgins 2007], simple footprints [Van De Panne 1997] that a character should follow, on space-time constraints as proposed in [Cohen 1992], or statistical models [Min and Chai 2012] that are responsible for retrieving and synthesizing a character's motion. However, even if easily specified constraints enable a user to animate a character, different frameworks that permit either the interactive or automatic animation of a character have been developed. In [Feng et al. 2012], which is a task-based character animation system, by using a number of screen buttons, the user is able to animate a character and make it interact with objects that are located within the virtual environment. Other methods, such as [Thiebaux et al. 2008][Kapadia et al. 2011][Shoulson et al. 2013], which can be characterized as scenario-based character animation systems, provide automatic synthesizing of a character's motion based on AI techniques.

In the past, researchers developed scripting languages and systems in the field of embodied conversational agents. The XSAMPL3D [Vitzthum et al. 2012], AniLan [Formella and Kiefer 1996], AnimalScript [Rößling and Freisleben 2001], SMIL-Agent [Balci et al. 2007] and many others enable a user to script a character's actions based only on predefined command. Among the best-known markup languages for scripting the animation of virtual characters are the Multi-Modal Presentation Markup Language [Prendinger et al. 2004], the Character Markup Language [Arafa and Mamdani 2003], the Multimodal Utterance Representation Markup Language [Kranstedt et al. 2002], the Avatar Markup Language [Kshirsagar et al. 2002], the Rich Representation Language [Piwek et al. 2002], the Behavior Markup Language [Vilhjalmsson et al. 2007] and the Player Markup Language [Jung 2008], which developed for controlling the behavior of virtual characters.

The representation of all previously mentioned languages is based to an XML-style format that allows users to script tasks featuring virtual characters. However, these languages focus more on communicative behavior such as gestures, facial expression, gaze and speech of virtual reality characters, instead of providing functional characters that can generate scenario-related animated sequences.

Various solutions that are similar to the presented methodology were proposed previously for the animation of virtual characters based on scripting commands. StoryBoard [Gervautz and Schmalstieg 1994] provides the ability to integrate a scripting language into an interactive character animation framework. Improv [Perlin and Goldberg 1996], another framework with which to create real-time behavior-based animated actors, enables a user to script the specific action of a character based on simple behavior commands. STEP [Huang et al. 2002] framework provides a user the ability to script such actions as gestures and postures. This methodology, which is based on the formal semantics of dynamic logic, provides a solid semantic foundation that enriches the number of actions that a character can perform.

The majority of previously developed scripting environments and markup languages provide only specific actions that a character can perform. An additional limitation is the inability of such systems to enhance a character's synthesized motion. Therefore, a user always receives a lower level of control of the synthesized motion of a character. Moreover, in cases in which a user must generate an animated sequence where many characters will take part, a great deal of effort will be required due to the difficulty of scripting multiple actions for multiple characters. This is especially true for users who wish to generate a sequence with animated characters, but are inexperienced in programming.

These difficulties are overcome in the presented scripting environment. Firstly, instead of enabling a user to script an animated character based on XML-related formats, a simplified scripting environment with its associated scripted language, which is based only on three commands, is introduced. Secondly, since a character should be able to perform concurrent actions, a simple extension of the basic command handles this. Therefore, the user achieves a higher level of control of a character's action. Moreover, in cases where a user must animate more than one character simultaneously, one can specify the character that should perform the requested action by adding an additional method to the existing command for a character. Finally, in cases where a user must generate an animated character in a multitask scenario, by simply specifying the row in which the task should appear, the system will synthesize the tasks requested automatically.

We assume that the described unique functionalities that are implemented in CHASE will enable a user to synthesize compelling animated sequences in which a variety of virtual characters are involved. Hence, in view of the simplicity of the developed commands, in conjunction with the associated parameters, the proposed methodology is quite powerful in comparison to the previous solution. In addition, the easy-to-use and easy-to-remember commands make the presented scripting environment effective, especially for users who are inexperienced in programming.

3 System Overview

This section briefly describes the proposed system. Specifically, a variety of background algorithms are responsible for recognizing the input commands and synthesizing the motion of a character. The developed background algorithms communicate with the animation system, which is responsible for generating a character's motion, as well as with a path-finding methodology to retrieve the path that the character should follow when a locomotion sequence

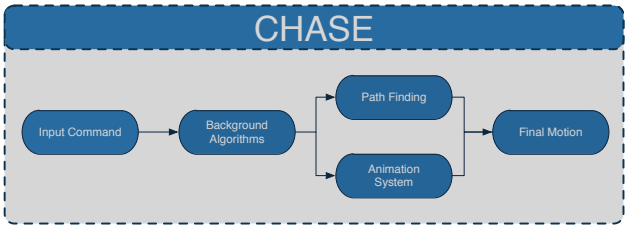

Figure 2: The architecture of CHASE

is required. Finally, CHASE synthesizes and displays the requested motion sequence. Figure 2 represents the procedure.

3.1 Interface

The interface of CHASE (see Figure 1) is characterized by its simplicity. In its current implementation, it consists of a scene panel that displays the resulting animations, an edit mode panel to edit the input objects, a progress bar that shows the progress of the displayed animation, a scripting box, and a few buttons for use in building, playing and clearing the written scripts. Finally, buttons that save the scripted code and export the generated animated sequences are also provided.

A downloadable version of the presented system, documentation specifying all of its capabilities, and examples of scenes can be found on the CHASE webpage (*url omitted for review purposes*).

3.2 Third-Party Implementations

A number of techniques and libraries are used to construct CHASE. CHASE uses the Recast/Detour library [Mononen accessed 29/11/2014] for the path finding process and collision avoidance with the environment. Concurrent actions are generated based on a simple layering methodology similarly to the one proposed in [Oshita 2008]. Finally, a similar to [Lang accessed 29/11/2014] full-body inverse kinematics solver was implemented to handle the postures of a character, while interacting with objects located within the virtual environment.

4 Scripting Character Animation

Developing scripting commands for animating a virtual character can be characterized as a complex process since a virtual character should be able to perform a variety of actions. In this section, the identifications of the basic scripting commands that are necessary to enable the virtual character to navigate and interact within a virtual environment are presented. Moreover, by introducing additional methods called by the main scripts, the system generates concurrent actions of a character, as well as animates multiple characters simultaneously. Finally, an additional functionality of CHASE for scripting multitask animated sequences for the generation of scenario-related animated characters is presented.

4.1 Identifying Scripting Commands

The application that is presented has been developed for users who are inexperienced in programming. Thus, simple, easily memorized, scripting commands are necessary. To generate the required scripting commands, one must begin by identifying the possible actions or type of actions that a character should perform. Generally, a character should be able to perform simple actions such as waving its hand, tasks related to locomotion such as moving to a target position and interaction tasks such as grasping with its hand an object that is located in the three-dimensional environment. It is apparent that these are the three basic types of actions that a virtual character should be able to perform. Based on this general description, three basic scripting commands were developed: the `do(parameters)`, the `goTo(parameters)` and the `interactWith(parameters)`.

The `do(parameters)` command provides a character with the ability to perform a single action. The `goTo(parameters)` forces a character to move within the given virtual environment. The final command is responsible for making the virtual character capable of interacting with a variety of tasks. Hence, the third command, the `interactWith(parameters)`, is responsible for providing the ability to control a variety of the character's actions.

For these commands, the `parameters` within the parentheses indicate the possible parameters that each of the scripting commands could receive (see Section 4.2). Due to the various parameters that each command receives, a user is provided with the means to develop both abstract and specified action of a character. For example, with the `goTo(parameters)` command, it is possible not only to generate the required locomotion of a character, but also to enable a user to gain better control of the synthesized motion of a character, since the user can specify how the locomotion of a character should be generated. The following section presents the basic parameters that each command receives.

4.2 Command Parameters

A task assigned to a character can be performed in a variety of different ways. For example, a sequence of locomotion to a target position can be performed by walking, running, etc. motions. Hence, in cases where a user needs a higher level of control of the synthesized motions of a character, parameters that enhance these actual actions generated by the previously mentioned scripting commands should be defined.

The first command that implemented the `do(parameters)` command, enables a user to script simple actions of a character. This command has a single mandatory parameter, which indicates the action that the character should perform. However, optional parameters to specify the body part or the duration of the task can also be used. Specifically, the user can request a single action by calling `do(action)`, as well as specify the target where the action should be performed, the duration of the action and the body part that should perform the requested action. This command initially permitted a character to perform the requested action without the need to perform a locomotion sequence (i.e., to wave its hand while staying in its position). However, the `do(parameters)` command can also be used to permit the character to perform locomotion tasks, since one can request that a character perform a walking motion. Based on these parameters that can be inserted into the `do(parameters)` command, a user has the means not only to generate the requested action, but also to generate an action that should fulfill user-specified constraints.

The `goTo(parameters)` command enables the character to perform locomotion tasks. The user identifies a mandatory parameter, which is the target position that the character should reach. However, the user is also able to use an additional optional parameter that specifies the motion style that will animate the character. Therefore, a character's locomotion to a target position can be scripted either by (i) inserting the target position such as `goTo(target)` when a simple walking motion of the character is desired or (ii) inserting `goTo(target, motion style)` when both target position and motion style are specified.

The final command that is implemented in CHASE, the `interactWith(parameters)`, can be characterized as more complex than the two previously mentioned commands. The reason

is that there are numerous possible interactions between a character and an object. If a character is asked to interact with an object, various actions can be generated. Even if possible to associate actions with specific body parts of a character in a pre-processing stage, there are also possible variations of the required actions. These variations may be related to the character's body or to the duration of the display of the action. For example, scripting a character to kick a ball may also require specifying the foot that should perform this action. Moreover, asking a character to knock a door may also require specifying the duration in the knocking. For that reason, four different parameters have been defined. The first two parameters (`object name` and `interaction module`) are mandatory. They indicate the object that the character should interact with, and the interaction module that should be generated. However, depending on the user's requirements for generating a specific action, two more optional parameters could also be inserted. The first one (`body part`) enables the user to choose which of the character's body parts should perform the requested action. In the current implementation, the user is permitted to choose the hand or foot that will perform the action. The second parameter (`duration`) enables the user to choose the time (in seconds) required for the requested action.

Based on the possible parameters that each command could receive, the following should be noted. Firstly, while the user did not specify any optional parameter for a scripted command, the system generates the required action taking into account a predefined set of parameters that are associated with each action of the character. For example, if a user requests that a character kick a ball, the system will display only a single kick by the character. The reason is that a ball kicking action is defined as to be performed only once to avoid synthesizing meaningless and repeated motions. Secondly, it should be noted that each optional parameter is independent. This means that the user is not required to specify all of the optional parameters provided by each command. Therefore, the user may control specific components of the requested action. A simple example of this capability of the commands illustrates this. While using the `do(parameters)` command, the user may request that only either the `body part` or the `duration` parameter, or both of these, be filled. In any case, the system's decision in generating the requested motion is not influenced by other factors since it is capable of recognizing the correct form of the scripted command in all of the aforementioned cases.

The three commands that are examined in this paper in conjunction with the associated parameters that can be used to animate a virtual character are summarized in Table 1. In addition, a small set of possible keywords that the user could employ in order to animate virtual characters is presented. It is assumed that an additional control parameter for the synthesized motion could be quite beneficial, since it enables the user not only to animate a character, but also to force the system to synthesize the user's actual wish. Complete documentation of all possible actions that can be synthesized by the character can be found in the CHASE webpage (*url omitted for review purposes*).

4.3 Scripting Concurrent Actions

Virtual characters, such as humans, should be able to perform more than one action simultaneously. This section presents the scripting process for concurrent actions that a character can perform. The concurrent action functionality is based upon the ability to specify the body part that should perform the action in conjunction with the base action that has been requested. The concurrent action lies between the `do(parameters)` and either the `goTo(parameters)` or the `interactWith (parameters)` commands. Specifically, to have a character perform concurrent actions, the `do(parameters)` command is attached to either the `goTo(parameters)` or the `interactWith (parameters).` A simple example follows. To cause a character to perform a motion, such as waving its hand while walking to a target position, the system permits the user to script the desired walking motion of a character and to request the additional motion that the system should generate. Hence, the previous example can be requested simply by scripting `goTo(target, walk).do(wave hand, handR).` Thus, by permitting the user to generate additional actions of a character, while another action is in progress can, be quite beneficial when more complex animated sequences are required. Therefore, this additional functionality provides a higher level of control over a requested action of a virtual character.

Algorithm 1: A simple example for generating a multitask scenario.

Data: Input commands of a user
Result: The result animated sequence
task[1] = do(wave hand, handR, 3);
task[2] = goTo(ball, walk).do(wave hand, handL);
task[3] = interactWith(ball, punch, handR);
task[4] = do(jump);
task[5] = do(wave hand, handR, 2);

4.4 Scripting Multiple Characters

In animated sequences it is quite common for more than one character to participate in a single scenario. Hence, by extending the three scripting commands, CHASE also enables a user to script more than one character simultaneously. This is achieved by attaching an additional command to one of the three basic commands, called `characterName(parameter)`. This command specifies the character that should perform an action, permitting the user to control multiple characters, in cases where more than one character participates in the animation process. A simple example of forcing a specific character to perform an action follows. Consider a character named `Rudy` who is required to walk to `target`. This procedure could be called by simply scripting `goTo(target).characterName(Rudy).`

4.5 Scripting Multiple Tasks

In scenario-related sequences that involve virtual characters, the latter should be able to perform a variety of tasks one after the other. Thus, this paper presents a method to script multiple tasks, such as enabling a user to synthesize long animated sequences. Generally, the tasks that a character can perform are characterized by their linearity. Specifically, a task begins while a previous task is completed, and the procedure continues until there are no other tasks for a character to perform.

Based on the foregoing, a multitask scenario in a general form can be represented as components of an array that has a dimensionality equal to $N \times 1$, where N denotes the total number of tasks that a character should perform. By assigning each of the actions an array called `task[index]`, a user can generate long animated sequences. This is achieved by allowing the user to assign singe tasks at each `index` value of the `task` array. A simple example of a multitask scenario appears in Figure 3, as well as in the accompanying video. Its scripting implementation is represented in Algorithm 1.

It is quite common in multitask scenarios to involve multiple characters. Two different approaches can be used in CHASE to script more than one character simultaneously in a multitask scenario. The first approach animates each character one after the

Commands	Parameters	Parameter Examples
do(*parameters*);		

do(action);
do(action, target);
do(action, duration);
do(action, body part, target);
do(action, body part, duration);
	action	wave hand
		jump
		walk
		kick
		etc.

	target	Vector3 (x,y,z) object name
	duration	time in seconds
	body part	handR handL footR footL upperB lowerB
goTo(*parameters*);		
goTo(target); goTo(target, motion style);	target	Vector3 (x,y,z) object name
	motion style	walk run jump walk back etc.
interactWith(*parameters*);		

interactWith(object name, interaction module);
interactWith(object name, interaction module, body part);
interactWith(object name, interaction module, duration);
interactWith(object name, interaction module, body part, duration);
	object name	any object's name contained in the scene

	interaction module	kick punch grasp sit open close etc.
	body part	handR handL footR footL
	duration	time in seconds

Table 1: Commands and associated parameters that can be used in CHASE to request an action by an animated virtual character.

Algorithm 2: By placing the actions of two different characters at different `index` values of the `task` array, the system generates each character action one after the other.

Data: Input commands of a user
Result: The result animated sequence
task[1] = goTo(ball, walk).characterName(characterA);
task[2] = goTo(ball, walk).characterName(characterB);

other. This means that the action required of a character B is generated after the action of a character A has been completed. The reason is that each task of the characters taking part in the multitask scenario have been assigned a different `index` value of the `task` array. A simple example of generating the actions of two different characters appears in Algorithm 2. However, a user should be able to animate virtual characters simultaneously in multitask scenarios. This is achieved in CHASE by using a two dimensional array named `tasks[index][index]`. In this array the first `index` value represents the row in which each action in generated, whereas the second `index` value represents the number of the character. It should be noted that each character should be represented by the same `index` value while developing a multitask scenario. Hence, the background algorithms that are implemented recognize and generate the requested tasks as separate entries. This enables the user to animate a number of characters simultaneously. A simple example in which there are two characters in a multitask scenario appears in Algorithm 3. It should be noted that a multitask scenario where multiple characters evolve in a general form can be represented as an array that has a dimensionality equal to

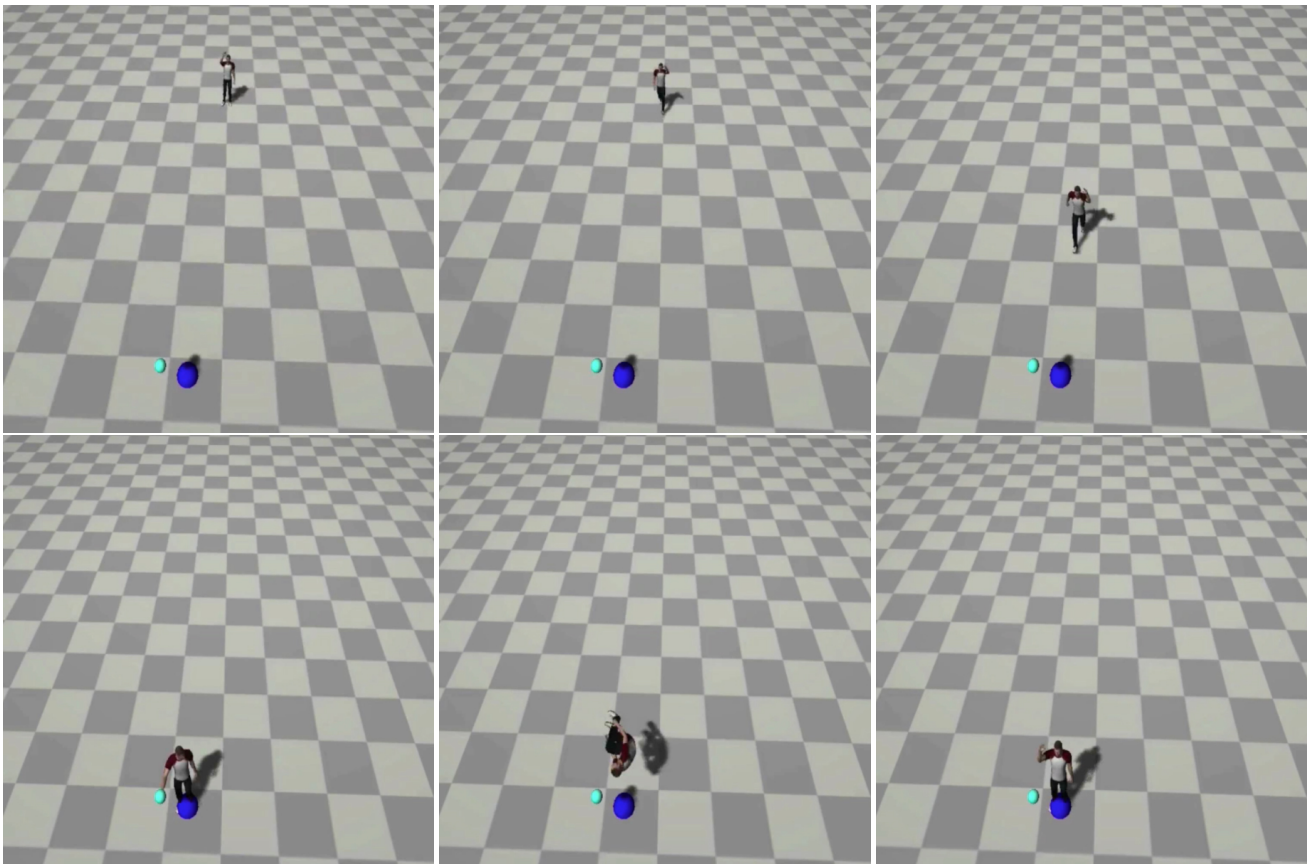

Figure 3: A multitask scenario generated by using Algorithm 1.

Algorithm 3: A multitask scenario in which there are two characters. In this scenario, `characterA` moves to its `target` position while walking, and `characterB` moves to its `target` position while running. Finally, `characterA` punches `characterB` with his right hand.

Data: Input commands of a user
Result: The result animated sequence
tasks[1][1] = goTo(target, walk).characterName(characterA);
tasks[1][2] = goTo(target, run). characterName(characterB);
tasks[2][1] = interactWith(characterB, punch, handR).characterName(characterA);

$M \times N$, where M denotes the total number of characters evolving in the multitask scenario and N denotes the total number of tasks that a character should perform.

5 Evaluation and Results

This section covers the results obtained while evaluating CHASE. To understand the efficiency of using CHASE, we designed a study to evaluate the ability of individual students to animate virtual characters that perform specified tasks. The participants were asked to use CHASE to develop a specified scenario. Two different teams of participants took place in the evaluation process. The first team consisted of eight participants who were first or second year undergraduate students in science or engineering departments. These students had a basic knowledge of computer programming (had attended only an introductory course in computer programming). However, none of them had previous experience in, or any particular knowledge of, computer graphics and animation. The second team consisted of eight participants who were in their first or second year of study in arts. None of the students of this team had any prior knowledge of programming. All participants can be characterized a computer literate.

5.1 Experimental Methodology

Each participant sat down in a quiet room in front of a desktop computer in which the CHASE application had been installed. First, the experimenter demonstrated the functionalities of CHASE for 10 minutes. During that time, each participant had the opportunity to note freely. After the demonstration, each participant was given the opportunity to discuss the system and to ask questions. An additional five minutes were provided for experimentation with CHASE. However, no further explanation of the system was given.

After this introductory procedure, the experimenter explained to the participants that they would use CHASE to generate a scenario with only one character involved. Specifically, it was asked to generate the scenario presented in Algorithm 1. The scenario was provided in a printed form in which the characters' actions were de-

scribed sequentially in writing. The actions that each character was required to perform and the position that was to be reached were written in bold letters in the printed document. The basic commands and parameters that the users were required to employ were supplied on a separate sheet. The time (in seconds) that was required to complete the given scenario was recorded. However, each participant was permitted a maximum of 15 minutes to complete the assigned scenario. After a participant completed the assigned scenario, he or she evaluated the difficulty of generating an animated sequence using the commands and functionalities of CHASE by assigning a score of 1 (difficult) to 7 (easy-to-use). Finally, since CHASE was developed to enable inexperienced programmers to visualize and generate their own stories that involved virtual characters, the participants were asked to evaluate CHASE by assigning a score of 1 (would not like to use it again) to 7 (I would strongly like to use it again).

5.2 Results

We recorded separately for each team the time that participants took to complete the given scenario. A pairwise $t-test$ revealed that science and engineering students completed the given scenario more quickly than the students from the arts department. Specifically, the mean time for the science and engineering students was estimated at $m = 316.9$ seconds with a standard deviation of $\sigma = 69.2$. The arts department students had a mean time of $m = 657.2$ seconds with a standard deviation of $\sigma = 86.7$. Based on these results it can be stated that students more familiar with programming are able to finish the given task faster. However, by considering the achievement of the arts department students of completion of the given tasks, it could be stated that also inexperienced in programming users can efficiently use such a scripting environment.

For evaluation purposes, we also recorded the students' satisfaction in interacting with CHASE by scoring the difficulty of using the developed commands in CHASE. Users were invited to evaluate the intuitiveness and difficulty of each command on a 7-point Likert scale (1 being difficult, 7 being easy). A pairwise $t-test$ found that it was easier for participants from the science and engineering departments to use the scripting commands than it was for the students from the arts department. Specifically, for the arts department students $t(8) = 3.09$, $p = 0.018$, with the science and engineering department students $m = 4.6$ on the difficulty scale and $\sigma = 1.3$, while in the arts department students $m = 3.8$ with $\sigma = 1.7$. However, a pairwise $t-test$ did not reveal any statistical difference between the two intuitiveness scales ($p > 0.05$).

The final results indicate the potential willingness of each participant to use CHASE in the future. Surprisingly, all participants responded that they would like to use CHASE in the future. Specifically, $m = 6.75$ with $\sigma = 0.46$. Therefore, it is shown that all of the participants would like to visualize their ideas and also develop their own stories that involve virtual characters by using CHASE.

6 Conclusions and Future Work

In this paper, a novel scripting environment, called CHASE, for use in animating virtual characters was presented. CHASE enables a user to request a variety of actions that a character can perform by simply using three commands. Each command, which receives a variety of parameters, is associated with specific actions that the character is able to perform. Moreover, the commands communicate with a variety of background algorithms that are responsible for generating the actions requested of the character. In addition to the scripting commands, by introducing three additional functionalities, the user is able to script concurrent actions of a character, multiple characters at the same time, and multi-task scenarios in order to generate scenario-related sequences that involve animated characters. To demonstrate the efficiency and simplicity of use of CHASE, an evaluation process was conducted. Two teams of students took part in the study. The first team consisted of students who had minimal previous programming experience, whereas the second team consisted of students who had no previous programming experience. This evaluation process has shown how easy it is to use such a scripting environment for the animation of virtual characters, as well as the potential willingness of users to employ CHASE in order to visualize their ideas and produce their own stories that involve animated virtual characters.

In its current version, CHASE provides a variety of functionalities. However, there are additional functionalities that we would like to implement in the near future. Specifically, we would like to implement additional commands in conjunction with the associate parameters to allow the system to provide complex interaction between multiple characters. In addition, we would like to expand the concurrent actions functionality by allowing the user to specify more than one body part that performs additional actions simultaneously. Moreover, in the current version of CHASE, the camera that is used to capture the virtual content is placed into the virtual environment in a specific position. Hence, in the future we would like to provide functionalities to allow the camera to be positioned by scripts, as well as to animate in response to the users' requests. We assume that the additional functionalities mentioned will be quite beneficial in generating a wide range of actions and interaction, as well as in representing the generated sequences more properly. As a result, users would be able to generate compelling sequences that involve animated virtual characters.

References

ARAFA, Y., AND MAMDANI, A. 2003. Scripting embodied agents behaviour with cml: Character markup language. In *International Conference on Intelligent User Interfaces*, ACM Press, New York, USA, 313–316.

BALCI, K., NOT, E., ZANCANARO, M., AND PIANESI, F. 2007. Xface open source project and smil-agent scripting language for creating and animating embodied conversational agents. In *International Conference on Multimedia*, ACM Press, New York, USA, 1013–1016.

CHAI, J., AND HODGINS, J. K. 2005. Performance animation from low-dimensional control signals. *ACM Transactions on Graphics 24*, 3, 686–696.

COHEN, M. F. 1992. Interactive spacetime control for animation. *ACM SIGGRAPH Computer Graphics 26*, 2 (July), 293–302.

DAVIS, J., AGRAWALA, M., CHUANG, E., POPOVIĆ, Z., AND SALESIN, D. 2003. A sketching interface for articulated figure animation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, UK, 320–328.

FENG, A. W., XU, Y., AND SHAPIRO, A. 2012. An example-based motion synthesis technique for locomotion and object manipulation. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM Press, New York, USA, 95–102.

FORMELLA, A., AND KIEFER, P. P. 1996. Anilan - an animation language. In *Computer Animation*, IEEE, New York, USA, 184–189.

GERVAUTZ, M., AND SCHMALSTIEG, D. 1994. Integrating a scripting language into an interactive animation system. In *Computer Animation*, IEEE Press, New York, USA, 156–166.

GLEICHER, M. 1998. Retargetting motion to new characters. In *25th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, New York, USA, 33–42.

HUANG, Z., ELIËNS, A., AND VISSER, C. 2002. Step: A scripting language for embodied agents. In *Workshop of Lifelike Animated Agents*, Springer-Verlag, Berlin, Germany, 87–109.

INSTITUTE FOR CREATIVE TECHNOLOGIES, accessed 29/11/2014. Smartbody. `http://smartbody.ict.usc.edu/`.

JUNG, Y. A. 2008. Animating and rendering virtual humans: Extending x3d for real time rendering and animation of virtual characters. In *International Conference on Computer Graphics Theory and Applications*, SCITEPRESS, UK, 387–394.

KAPADIA, M., SINGH, S., REINMAN, G., AND FALOUTSOS, P. 2011. A behavior-authoring framework for multiactor simulations. *Computer Graphics and Applications 31*, 6, 45–55.

KOVAR, L., AND GLEICHER, M. 2003. Flexible automatic motion blending with registration curves. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, UK, 214–224.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. *ACM Transactions on Graphics 21*, 3, 473–482.

KRANSTEDT, A., KOPP, S., AND WACHSMUTH, I. 2002. Murml: A multimodal utterance representation markup language for conversational agents. In *AAMAS Workshop Embodied conversational agents - let's specify and evaluate them!*.

KSHIRSAGAR, S., MAGNENAT-THALMANN, N., GUYE-VUILLÈME, A., THALMANN, D., KAMYAB, K., AND MAMDANI, E. 2002. Avatar markup language. In *Workshop on Virtual Environments*, Eurographics Association, 169–177.

LANG, P., accessed 29/11/2014. Root-motion. `http://www.root-motion.com/`.

LEVINE, S., THEOBALT, C., AND KOLTUN, V. 2009. Real-time prosody-driven synthesis of body language. *ACM Transactions on Graphics 28*, 5, Article No. 28.

MCCANN, J., AND POLLARD, N. 2007. Responsive characters from motion fragments. *ACM Transactions on Graphics 26*, 3 (August), Article No. 6.

MIN, J., AND CHAI, J. 2012. Motion graphs++: A compact generative model for semantic motion analysis and synthesis. *ACM Transactions on Graphics 31*, 6, Article No. 153.

MONONEN, M., accessed 29/11/2014. Recast/detour navigation library. `https://github.com/memononen/recastnavigation`.

MOUSAS, C., AND ANAGNOSTOPOULOS, C.-N. 2015. *Character Animation Scripting Environment*. Encyclopedia of Computer Graphics and Games. Springer.

MUKAI, T., AND KURIYAMA, S. 2005. Geostatistical motion interpolation. *Transactions on Graphics 24*, 3 (July), 1062–1070.

OSHITA, M. 2008. Smart motion synthesis. *Computer Graphics Forum 27*, 7 (October), 1909–1918.

OSHITA, M. 2010. Generating animation from natural language texts and semantic analysis for motion search and scheduling. *The Visual Computer 26*, 5, 339–352.

PARK, S. I., SHIN, H. J., AND SHIN, S. Y. 2002. On-line locomotion generation based on motion blending. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, UK, 105–111.

PERLIN, K., AND GOLDBERG, A. 1996. Improv: A system for scripting interactive actors in virtual worlds. In *23rd Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, New York, USA, 205–216.

PIWEK, P., GRICE, M., KRENN, B., BAUMANN, S., SCHRÖDER, M., AND PIRKER, H. 2002. Rrl: A rich representation language for the description of agent behaviour in neca. In *AAMAS Workshop on Embodied Conversational Agents*.

PRENDINGER, H., DESCAMPS, S., AND ISHIZUKA, M. 2004. Mpml: A markup language for controlling the behavior of lifelike characters. *Journal of Visual Languages & Computing 15*, 2, 183–203.

RÖSSLING, G., AND FREISLEBEN, B. 2001. Animalscript: An extensible scripting language for algorithm animation. *ACM SIGCSE Bulletin 33*, 1 (February), 70–74.

SAFONOVA, A., AND HODGINS, J. K. 2007. Construction and optimal search of interpolated motion graphs. *ACM Transactions on Graphics 26*, 3 (August), Article No. 106.

SARRIS, N., AND STRINTZIS, M. G. 2003. *3D Modeling and Animation: Synthesis and Analysis Techniques for the Human Body*. IGI Global, Hershey, Pennsylvania, July.

SHOULSON, A., MARSHAK, N., KAPADIA, M., AND BADLER, N. I. 2013. Adapt: The agent development and prototyping testbed. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM Press, New York, USA, 9–18.

THIEBAUX, M., MARSELLA, S., MARSHALL, A. N., AND KALLMANN, M. 2008. Smartbody: Behavior realization for embodied conversational agents. In *International Joint Conference on Autonomous Agents and Multiagent Systems*, ACM Press, New York, USA, vol. 1, International Foundation for Autonomous Agents and Multiagent Systems, 151–158.

VAN BASTEN, B., AND EGGES, A. 2012. Motion transplantation techniques: A survey. *Computer Graphics and Applications 32*, 3, 16–23.

VAN DE PANNE, M. 1997. From footprints to animation. *Computer Graphics Forum 16*, 4 (October), 211–223.

VILHJALMSSON, H., CANTELMO, N., CASSELL, J., CHAFAI, N. E., KIPP, M., KOPP, S., MANCINI, M., MARSELLA, S., MARSHALL, A. N., PELACHAUD, C., RUTTKAY, Z., THORISSON, K. R., WELBERGEN, H. V., AND WERF, R. J. V. D. 2007. The behavior markup language: Recent developments and challenges. In *Intelligent virtual agents*, Springer Berlin-Heidelberg, 99–111.

VITZTHUM, A., AMOR, H. B., HEUMER, G., AND JUNG, B. 2012. Xsampl3d: An action description language for the animation of virtual characters. *Journal of Virtual Reality and Broadcasting 9*, Article No. 1.

WITKIN, A., AND POPOVIC, Z. 1995. Motion warping. In *22nd Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, New York, USA, 105–108.