An authoring tool to provide group and crowd animation using Natural Language scripts

Guido Mainardi
Pontifical Catholic University of Rio Grande do Sul
School of Technology, VHLab
Porto Alegre, Brazil
guido.mainardi@edu.pucrs.br

Vinícius Cassol
Pontifical Catholic University of Rio Grande do Sul
School of Technology, VHLab
Porto Alegre, Brazil
vjcassol@gmail.com.br

Soraia Raupp Musse
Pontifical Catholic University of Rio Grande do Sul
School of Technology, VHLab
Porto Alegre, Brazil
soraia.musse@pucrs.br

Aline Normoyle
Bryn Mawr College
Bryn Mawr, USA
anormoyle@brynmawr.edu

Norman Badler
University of Pennsylvania
Philadelphia, USA
badler@seas.upenn.edu

***Abstract*—Virtual environments have become ubiquitous, expanding beyond games into the domains of architecture, engineering, psychology, education, and archaeology. Furthermore, virtual humans can further enhance these environments when they provide compelling and coherent behaviors. In this paper, we present a scripting language based on simple, plain English commands. Our system assists people without game and animation expertise to populate large environments and complex scenarios. To validate our approach, we develop a prototype using Unreal Engine 4 and author a variety of indoor and outdoor agent simulations. Furthermore, we test our prototype with both experienced and inexperienced users, creating scenarios for a residence, mall, psychology scenario, and archaeological site.**

***Index Terms*—Crowd, authoring tools, virtual human animation**

I. Introduction

One way anthropologists study the past is to re-imagine the culture, customs, and rituals of ancient civilizations. Because ancient civilizations have few lasting remains, researchers construct theories about how people lived by synthesizing diverse sources, such as written accounts, buried artifacts, and even folklore. A helpful resource for analyzing such theories is visualizing them using virtual environments [1]. For example, packing streets, corridors, and spaces with large numbers of people can give insights into the circulation of people through the site along with possible locations of doors. While this is a specific motivation, in this paper we propose a scripting language technique developed for generic purposes and for people who may not have advanced programming and computer animation skills.

Thanks to freely available game engines, such as Unreal Engine, we now have tools that allow us to simulate places and populations, with a certain realism. However, controlling characters and groups using such tools is challenging. People without animation, programming, and game expertise must first learn how to use these tools in order to author simulations themselves. Thus, we propose an interface, based on a natural language scripting language, to facilitate the translation from the desired scenario into a visual animation.

In this work, we abstract the world-building features in Unreal Engine, so that people without animation/programming skills can focus on creating scenarios. Using this system, authors can define the roles and behaviors of different people, small groups, and crowds. Our proposed approach uses a scripting language to facilitate the process of creating hypotheses and visualizing them through computer animation [1]. This work presents a prototype that allows users to populate virtual environments with different types of people and activities. The prototype can interpret scripts (command sequence in English) that describe the behaviors of characters as well as conditional and temporal events. For example, a scenario designer can define shopping and restaurant behaviors in a mall as well as behaviors that determine how characters react to a fire alarm. These features make it possible to describe complex scenarios, where agents perform different behaviors, in a faster, simpler, and easier way. Using this model, we can define both the daily routines of ancient peoples as well as behaviors tailored to specific events, such as those related to ceremonies and rituals. The model has the potential to

streamline and facilitate the testing of several simulations for the validation of anthropological (or other) hypotheses.

We organize the paper as follows: Section II describes existing research into the use of scripts and frameworks for controlling agents. Section III describes our method and our script syntax. Section IV describes our results and simulated scenarios. Finally, Section V provides final comments and lists future work.

II. Related work

Previous research has studied the use of scripts for authoring virtual human behaviors. Kallmann et al. [2] proposed a system that allowed users to specify "plug-in" behaviors for agents within a virtual environment. The plug-ins took the form of scripts written in Python that referenced characters and objects in a shared virtual environment. Using this system, users could create and place characters and objects, trigger object interactions, make perceptual queries, or trigger character animations and locomotion. Similar to their system, our system loads agent behaviors like plugin-ins to an existing virtual world; however, our system does not require Python programming knowledge.

Musse and Thalmann [3] proposed a scripting system that allowed users to describe the behavior of groups and individual agents. Groups and agents could have different degrees of autonomy, ranging from independent to fully controlled by a script. Using this system, users had three ways to control agents and groups: via scripted behaviors; via behavior rules with events and reactions; and via the external control of agents in real-time.

More recently, Trescak and Bogdanovych [4] described their approach for creating more natural-looking behaviors by integrating planning algorithms that achieve agent goals. This research aims to simulate more realistic non-player characters (NPCs) in video games.

Gao *et. al.* [5] proposed the use of virtual space ontologies (a data model used to make inferences) to help application designers, who are usually non-computer professionals, to create scenarios with large numbers of agents. This work also sought to improve the reusability of authored agents.

Scripts are a common technique for creating and controlling agents across many domains, such as simple simulations, educational applications, and multimedia presentations. Vosinakis and Panayiotopoulos [6] described their *SimHuman* system, which provided a programming library and utilities for making virtual worlds populated with agents. Within e-Learning, Adamo-Villani *et. al.* [7] used text-based scripts to create a virtual instructor-avatar that interacted with students.

This work is inspired by the work of Chow *et al.* [1] which describes the need for parameterizable and easily customizable virtual environments to test hypotheses about how people may have used ancient anthropological sites. Furthermore, the ability to create different simulations of virtual humans must be easy and straight-forward, as most anthropologists do not have animation and 3D programming experience.

III. Method

We develop a proof of concept using Unreal Engine. In our approach, an experienced designer must set up the characters and environment needed for the scenario beforehand so that a potentially inexperienced user can refer to these scene objects in their scripts. Our system re-uses Unreal Engine's features for path planning, rendering, and character animation to set up an application.

First, an experienced designer annotates the environment with bounding boxes, called *RegionBox*s in our system, that associate parts of the environment with labels that the script can reference. For example, a *RegionBox* might be labeled with "kitchen" or "temple". Second, this designer would import any character models and animations needed by the application so they can be associated with different *Agent* types that the script can reference.

Our prototype both creates and controls agents for the user, so users do not need to be aware of the internal technical details of Unreal Engine. We use the two main callback methods in Unreal to implement our script commands, *BeginPlay* and *Tick*. *BeginPlay* executes when the simulation starts. During *BeginPlay*, our system loads and places the characters who will participate in the simulation. *Tick* executes once per frame (e.g. in a loop). During *Tick*, our system updates the positions and current activities of the characters.

Our prototype implements a *CrowdController* class that manages all the characters in the application based on the user's scripts. The CrowdController performs "Scripts initialization" in *BeginPlay* and the main "Execution loop" in *Tick*.

The CrowdController uses the engine's *navmesh* features to represent the areas of the map where characters have access. Locations on the navmesh serve as objectives for agents and indicate which parts of the map agents can use for navigation. Although our scripting system could use the built-in navigation and collision avoidance features in Unreal, our implementation uses Biocrowds [8] to simulate agents, because of Biocrowd's features for simulating crowds with varying densities.

Writing and executing scripts are described in the following three phases. Figure 1 illustrates each phase using different colors:

1) *Script authoring*: Outside of the application, the user describes the agents to spawn as well as their desired behaviors (green boxes in Figure 1).
2) *Script initialization*: When the application starts (e.g. in BeginPlay), our system generates agents in their initial positions and states. (orange boxes in Figure 1).
3) *Execution loop*: Each frame (e.g. in Tick), the application checks for event conditions, assigns behaviors to agents, and updates the positions of agents in transit (blue and red boxes in Figure 1).

We describe each phase in the following sections.

A. Script Authoring

Our prototype takes two or more scripts, in text format, as input (green boxes in Figure 1). The user must provide two types of scripts: a main script and a behavioral script.

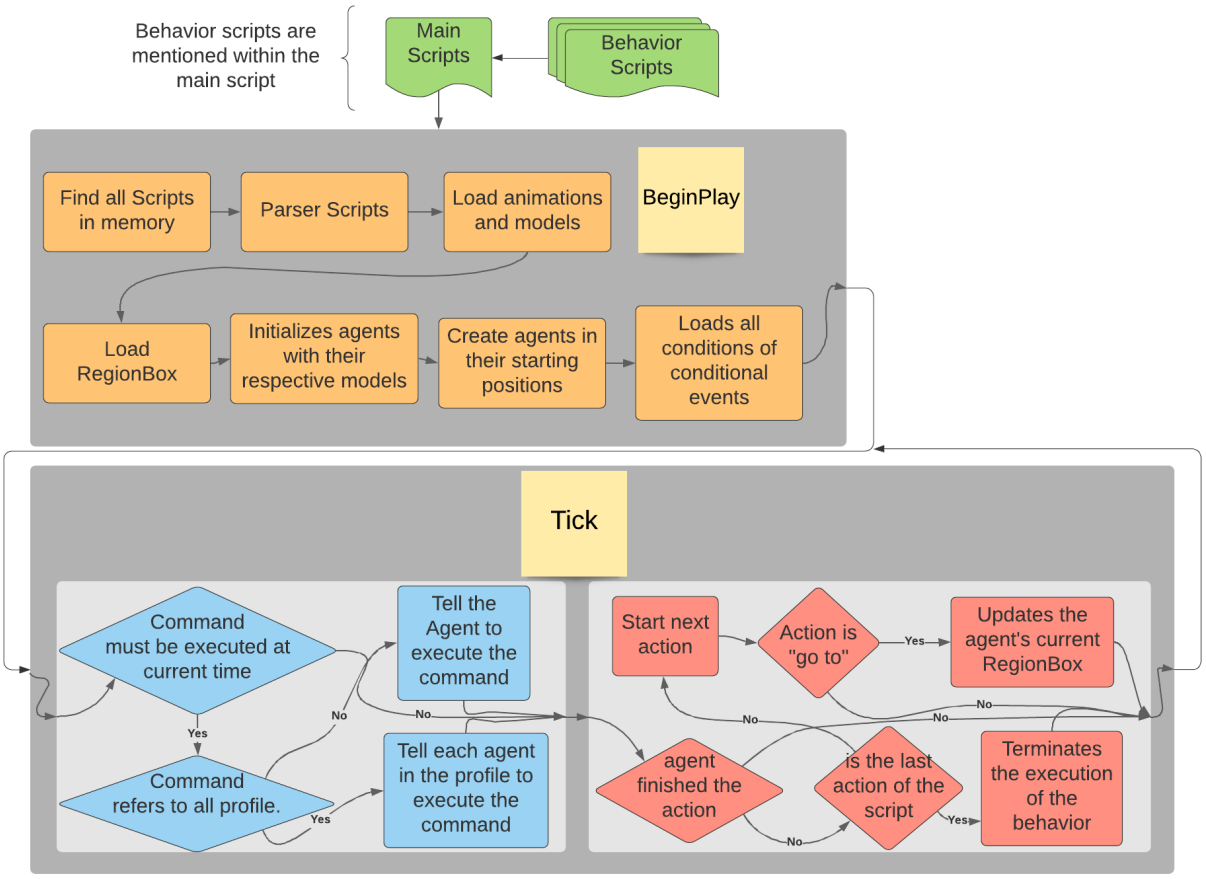

Fig. 1. Model developed.

1) Main Script: The main script creates agents, designates behaviors for them, and defines temporal and conditional events. Temporal events are triggered at a given time. Conditional events are triggered based on the locations of agents. All event-based behaviors replace an agent's current behavior. For example, if the user defines a temporal event behavior at time t, the agent will abort its current behavior and switch to the new one. The main script supports four commands:

- *Create*: This command spawns a given number of agents at a given location and corresponding to a given profile and (optional) model. A profile is an agent type defined by the user. Profiles typically refer to an agent role, such as "Guard" or "Resident". A model corresponds to a mesh and animations. The designer sets up the model along with the environment as part of the Unreal application. The basic command structure is:

  "***Create*** *<Number>* *<Profile>* ***in*** *<Place>*"

  The above command will create "*Number*" agents with the "*Profile*" at the location "*Place*". Locations are explained in Sections IV-A and IV-C. Above, no model is specified so agents are created with the default model. To specify an alternate model, the user uses the "**as**" keyword.

  "***Create*** *<Number>* *<Profile>* ***as*** *<Model>* ***in*** *<Place>*"

- *When*:
  This command defines a conditional event that triggers when an agent belonging to a given profile goes to the given location. This command follows the following structure:

  "***When*** *<Profile>* ***in*** *<Place>* *<Command>*"

  Above, when an agent of type "*Profile*" goes to "*Place*", the "*Command*" (event) will trigger.
- *Profile ID*:
  This command defines a temporal event that triggers for a specific agent belonging to a given profile. Agent identifiers are positive integers, assigned at creation. The first agent has "*ID*" 1, the second has "*ID*" 2, and so on successively. The command structure is:

  "*<Profile>* *<ID>* ***time*** *<t>* ***run/loop*** *<Script>*"

When the simulation time "*t*" is reached, the agent with profile "**Profile**" and identifier "*ID*" will start the behavior described in the "*Script*". There are two options for this command: "***run***" and "***loop***". The first option executes the given *Script* once. The second option repeats the given *Script*, until it is interrupted or the program is terminated.

- *All Profile*:
  This command defines a temporal event that triggers for all agents of a given profile. The command structure is:

  "***All*** *<Profile>* ***time*** *<t>* ***run/loop*** *<Script>*"

  When the simulation time "*t*" is reached, all agents with profile "**Profile**" will start the behavior described in the "*Script*".

2) Behavior: Behavior scripts define simple instructions that control an agent. For example, behavior scripts command an agent to go to a given location or run a given animation. Behavior scripts do not reference time. The instructions happen in sequence until there are no more instructions or the system assigns a new behavior to the agent. Behavior scripts currently support two command types:

1) *Go to*: Instructs an agent to navigate to a location in the environment. Locations are specified using *RegionBox*s that are pre-configured in the application. Its structure is:

   "***go to*** *<Place>*"

   The agent that is following this behavior goes to a random point inside the *RegionBox* named "*Place*".
2) *Play*: This command causes the agent to perform an animation. Animations are pre-configured in the application as part of the character model. Its structure is:

   "***play*** *<Animation>*"

   The agent executing this behavior will play the animation with the name "*Animation*".

B. Script initialization

The initialization step loads the information necessary to run a scenario. The initialization step takes place in *BeginPlay* and is represented by the orange boxes in Figure 1. This step loads and parses the main and behavior scripts (Section III-A). Then, this step checks that no script references any 3D models, animations, or locations (e.g. *RegionBox*s) that are not present in the Unreal application. Finally, this step initializes all the agents, with their respective 3D models, placing them at their point of origin (a *RegionBox* referenced by the script) and initializes all events that can be triggered when an agent goes to a specific *RegionBox*.

C. Execution loop

During each simulation tick (frame), our prototype updates the positions and behaviors of each agent. The system assigns behaviors based on any events specified in the main script (blue box in Figure 1) or based on behavior scripts (red box in Figure 1). If an agent is currently executing a behavior and an event triggers, the event behavior replaces the existing one. When executing a behavior script, agents complete each command in order. Commands may change the position and animation of the agent. If a script does not loop, the agent remains stationary until a new behavior is assigned to it.

IV. Results

We developed a prototype in Unreal that allows the user to control the behaviors of large numbers of agents. We present the following scenarios to demonstrate the features of the system:

1) *Region Annotation Example*: In Section IV-A we explain how to prepare a scenario so that agents can be controlled using our scripting system.
2) *Residence Example*: In Section IV-B we describe a simple scenario consisting of one agent living in a house.
3) *Mall Example*: Section IV-C describes a complex scenario, with many agents in a fire simulation, in a mall.
4) *Authoring evaluation*: Section IV-D describes the experiences of two authors using the system. One, who is an expert in simulating agents, and one, who is a novice but with a background in psychology.

A. Region Annotation Example

Our scripting system requires the environment and characters to be pre-configured. To demonstrate this configuration, we develop a scenario based on a residence, containing a bedroom, bathroom, kitchen, and living room (shorted to "room"). The residence model was inspired by the floor plan of Figure 2. The environment designer annotates different regions of interest using invisible bounds objects, called *RegionBox*s. Script writers refer to the names of different region boxes in order to spawn agents or specify goals for path planning. Figure 4 illustrates the scenario of Figure 2 recreated in Unreal and showing each *RegionBox* with its name.

B. Residence Example

We developed a simple scenario using the environment setup in section IV-A. In this scenario, a single agent walks between rooms. Rooms are marked in green in (Figure 4) and references to each room are highlighted in yellow in Figure 3. This scenario uses two scripts. The main script (green underline on the left of Figure 3) creates one agent in the living room (annotated as "Room") and then specifies an event at *time 3* seconds that triggers the GoTo behavior script. The GoTo script has a blue underline on the right of Figure 3.

In Figure 4 you can see the agent waiting at its spawn location, since it has not yet been assigned a behavior. After 3 seconds, the agent begins the GoTo behavior and starts walking towards the kitchen (Figure 5). Once the agent reaches the

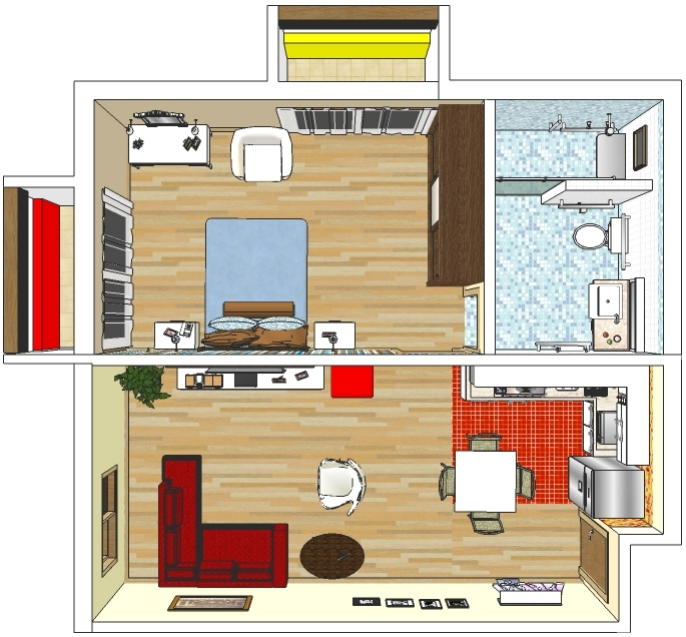

Fig. 2. Blueprint modeled in the example case.

Main.txt	GoTo.txt
Create 1 Agent **in** Room Agent 1 **time** 3 **run** GoTo	**go to** Kitchen **go to** Bathroom

Fig. 3. Scripts used in The Residence example.

kitchen, it switches to the next command in "*GoTo.txt*", and the agent begins walking towards the bathroom. After reaching the bathroom, the agent returns to a waiting state. When going to a location specified by a *RegionBox*, a random goal position is chosen within the bounds of the box for the agent to walk towards.

C. Mall Example

To demonstrate how our prototype supports authoring large environments and crowds, we create a non-trivial mall scenario Figure 9 shows the mall's blueprint and Figure 7 shows the corresponding model in Unreal.

We create a scenario consisting of shoppers, walkers, and guards. At time 30 seconds, a guard triggers an alarm that results in all characters evacuating the building (Figure 8). This scenario demonstrates the following features:

- *Animations*: In the behavior script, "*Buy.txt*", (Figure 11, underlined in red) agents perform the a "thinking" animation as they consider what they are going to buy. Figure 11 shows a screenshot of this behavior.
- *Agent models*: In the main script (Figure 8, underlined in pink) we load two non-default character models: the "*Macarena*" and the "*Highlight*". These models correspond to characters with different behaviors.

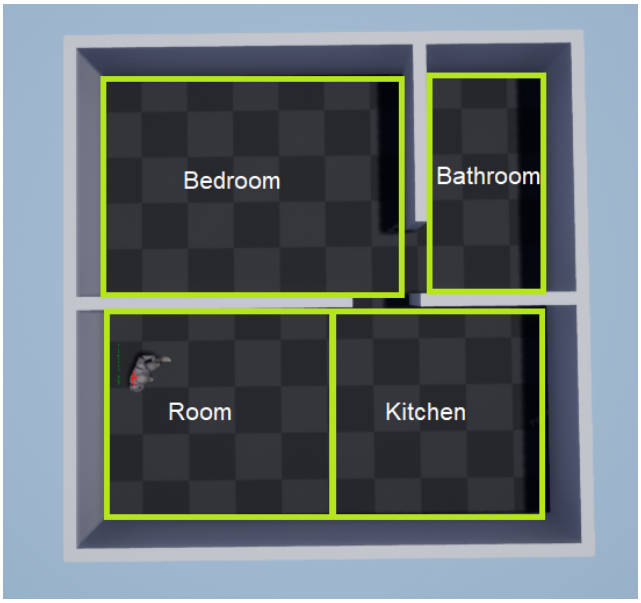

Fig. 4. Agent Created in Room, in the Residence example.

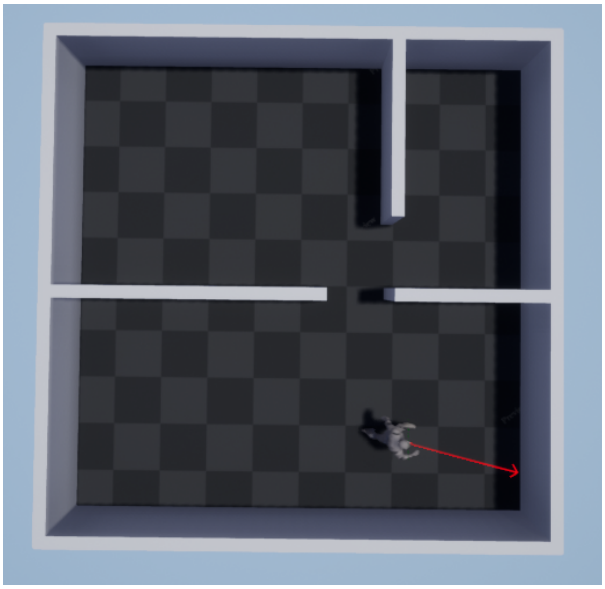

Fig. 5. Agent walking to the Kitchen.

- *Conditional Events*: Unlike temporal events, which are based on time, conditional events trigger based on whether agents are located in a given area (Figure 8, illustrated with a blue rectangle). In this scenario, agents repeat the "fire" behavior when they are in the "MallOffice". Agents go to the "MallOffice" in response to a fire alarm. Figure 14 shows a screenshot of agents executing this behavior.
- *Behavior profile*: Different categories of agents are specified when they are created from the main script (Figure 8, underlined in orange). These categories correspond to a profile which can be referenced by subsequent com-

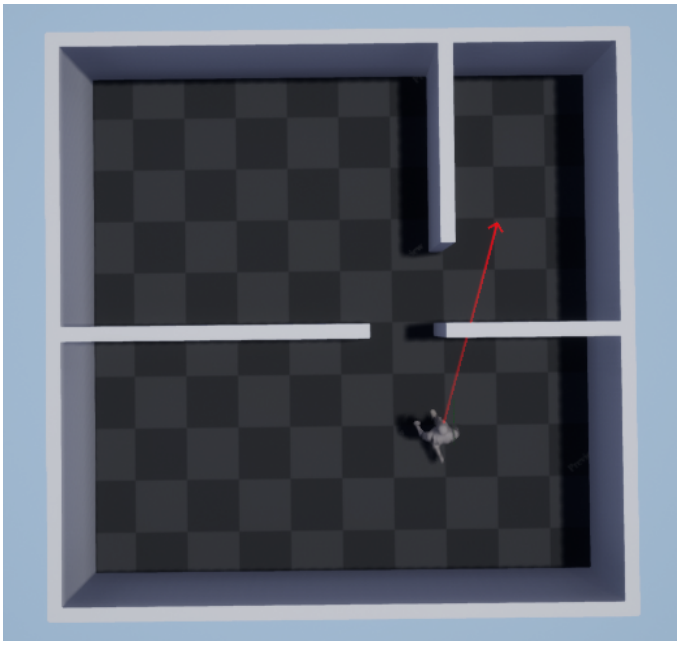

Fig. 6. Path to the bathroom. An arrow points from the character to a random location.

Fig. 7. Mall, modeled in Unreal.

mands, using the keyword "*All*". For example, all agents matching the "Buyer" profile executes the "Buy" behavior (Figure 8, underlined in green). Similarly all agents matching the "Guard" profile executes the conditional event "Fire" (Figure 8, underlined in brown).

- *Random*: Agents goals can be chosen randomly from all *RegionBox*s having the same name. For example, in Figure 11, shoppers choose random store locations. Users can alternatively set flags to indicate sets of random locations. For example, the flag "Public" indicates all public spaces available for walking (Figure 8, see "Walk.txt").

Main.txt

```
Create 1 Guard as Higlight in Hallway2
Create 1 Guard as Higlight in Hallway6
Create 10 Buyer as Macarena in Random Store
Create 50 Walker in Random Public
When Guard in MallOffice All Guard loop Fire
When Guard in MallOffice All Buyer loop Fire
When Guard in MallOffice All Walker loop Fire
Guard 1 time 0 loop Guard1
Guard 2 time 0 loop Guard2
All Buyer time 0 loop Buy
All Walker time 0 loop Walk
Guard 1 time 30 run Alarm
```

Fig. 8. Main script for the mall scenario.

Fig. 9. Mall Blueprint

Guard1.txt

```
go to Hallway4
go to Hallway2
```

Guard2.txt

```
go to Hallway8
go to Hallway6
```

Fig. 10. Guard script for the mall scenario.

Buy.txt	Walk.txt
go to Random Store **play** MaarenaThinking	**go to** Random Public

Fig. 11. Walk script for the mall scenario.

Alarm.txt	Fire.txt
go to MallOffice	**go to** ExergencyExit

Fig. 12. Fire evacuation scripts for the mall scenario.

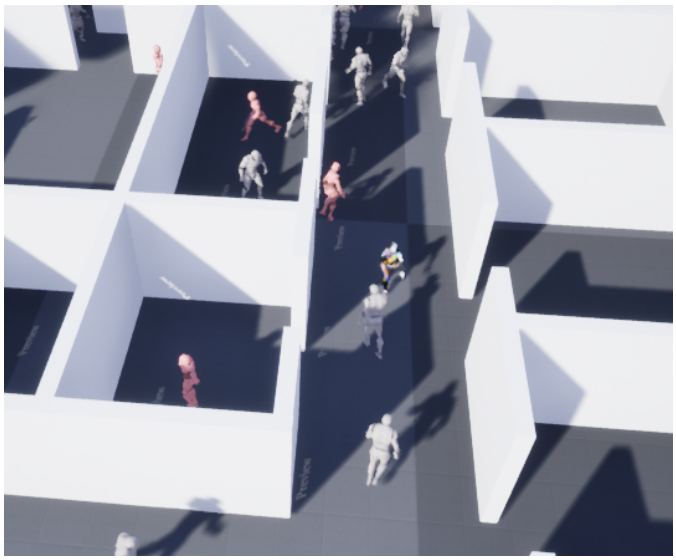

Fig. 13. Agent behaviors in the mall scenario. In the bottom left, one agent runs the "thinking" animation.

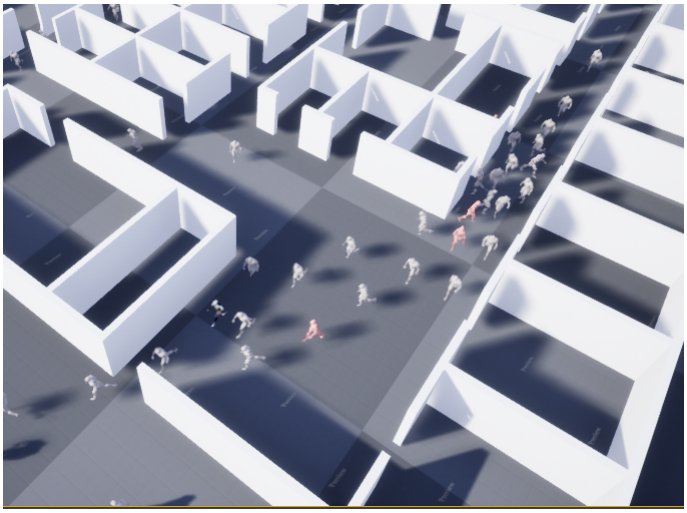

Fig. 14. Agent behaviors in the mall scenario. Agents execute conditional events to implement a fire evacuation.

D. Authoring Evaluation

To evaluate our approach, we performed guided sessions where we asked potential users to create scenarios using our prototype. We performed two distinct rounds of testing: in the first one, we invited a specialist in computer animation and simulation to give technical feedback; in the second one, we invited a psychology student who did not have computer animation expertise or technical programming knowledge, but was interested in human behavior simulation.

During the test sessions, users were asked to learn the different script commands so they could perform a list of tasks using the prototype. The tasks were performed within the mall model presented in the Section IV-C and are listed below:

1) Create an agent with the model "*Highlight*" in a random "*Store*".
2) Create an agent with the model "Macarena" in "*Hallway4*" that sits all the time (using the animation of "*MacarenaSitting*").
3) Create 10 agents that walk the halls endlessly.
4) Create 10 buyers with the "Macarena" template who walk from store to store by running the "*MacarenaThinking*" animation when they arrive at each store. When one of the buyers goes to the "AlecrimCroche" store, all the buyers must leave the mall.
5) Add any new animation and create a script that uses it.
6) Create a simulation where several people are waiting outside the mall. When a guard enters the mall, all waiting people must go to the stores in the mall and move between them.

E. Expert user feedback

The expert user initially found the system difficult to learn, until he understood how the scripts and behavior profiles worked. After the first moment of learning, he was able to create different scenarios with the agents within the test scenario.

The expert user reported that the prototype made it possible to define simple agent behaviors quickly and that the prototype was practical for scenario development. The prototype speeds up the process of changing information to carry out new tests in the same scenario. When you have several agents running randomly, the gain of time using scripts, instead of the engine itself, is not very significant, but when they start to create more specific situations, such as conditional events or agents being created in specific places, the scripts can save a great deal of time.

F. Novice user feedback

The second session tested the system with a user having no technical skills over the Zoom platform. A Psychology student interested in virtually reproduce and observe human behavior was briefly introduced (during approximately 30 minutes) to the system. Then, she started to follow the test script described in section IV-D. It is important to mention that the authoring prototype needs to be set up by an Unreal experienced user, and then the animation authoring can be done by a non-expert modeler. In our tests, the environment was prepared and then we give the computer controls (mouse and keyboard) to the student, over the Zoom platform. During the introduction, the

authors showed the student how to run the simulation in Unreal and how to edit the scripts.

The student was able to successfully perform the requested tasks and was able to create and link scripts. In a follow-up task, the student was asked to produce a new scenario related to psychology and using the 3D mall scenario. In this scenario, the student was asked to create two independent groups, with 10 agents each, and control their motion so they travel to a specific point of the mall (the room Jeronimo, illustrated in Figure 16). One group was composed of *sad* people; the other of *happy* people. After the groups meet at Jeronimo, the agents switch to new behaviors that lead them out of the mall, e.g. MainEntry for the sad agents and EmergencyExit for the happy agents. The happy agents used the "*Macarena*" character model and the sad agents used the default Unreal model.

In summary, the Psychology student was able to successfully author two scenarios. They were able to create groups as well as control their motion and animation. The student was able to create all the test scenarios in 1 and a half hours, which included the initial presentation time. The scripts created by the student can be seen in Figure 15 and Figure 16. According to the scripts the simulation was structured as follows:

1) Ten agents defined as happy were created on random positions in the scene and defined with the model Macarena;
2) Ten agents defined as sad were created in random positions in the scene (the default unreal human model was applied here);
3) More 10 agents (using Macarena model) were randomly created into the stores in order to represent other mall customers;
4) Both happy and sad group were requested to follow the walk script and move to *Jeronimo Store* after their creation.
5) On time 30, all the agents were requested to leave the mall, but sad people used the *Main Entry Region* while happy people used the *EmergencyExit Region*.

After the test session, we interviewed the student. She reported feeling secure and able to create simulations even without a technical background. In addition, she highlighted how easy it was to provide an animation with groups of characters. She believes that tools, such as the presented in this paper, can contribute to research in other areas, such as her field of psychology, where the user could simulate and visualize hypotheses in the early stages of research.

The student also reported areas where the system could be made easier to use. In particular, she recommended we use non-technical terms for keywords such as <loop>, and that we allow the user to author all scripted behaviors in the same file.

V. Final Considerations

In this work, we proposed an authoring tool that uses English Language scripts to control agents and virtual crowds in a pre-mapped scenario. We believe proposing a natural language

Main.txt

```
Create 10 Happy as Macarena in random
Create 10 Sad in random
Create 10 Buyer as Macarena in Random Store
All Happy time 0 loop Walk
All Sad time 0 loop Walk
All Sad time 30 loop Leave
All Happy time 30 loop Scape
```

Walk.txt

```
go to Jeronimo
```

Leave.txt

```
go to MainEntry
```

Scape.txt

```
go to EmergencyExit
```

Fig. 15. The scripts created by the Psychology student during our evaluation of the prototype.

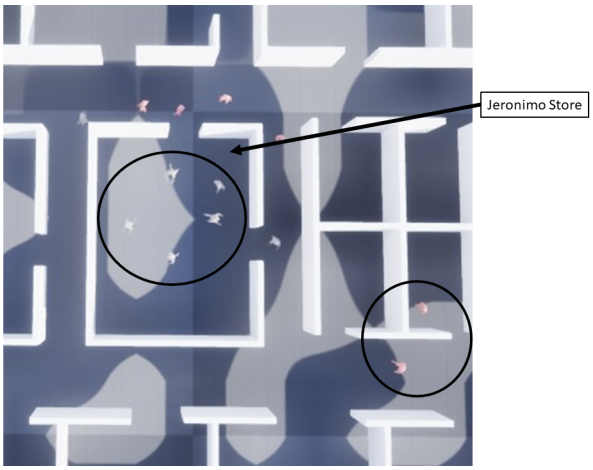

Fig. 16. Simulated environment with happy (2 pink agents on the right of the image) and sad agents (white agents inside Jeronimo Store).

script to create scenarios in Unreal is our main contribution. Furthermore, the ability to create scenarios using such scripts is an important feature because it can allow non-graphics experts – who do not have animation and game programming knowledge – to do research involving the reproduction of human behaviors in a virtual environment.

We developed a prototype and tested it with two users – one an expert in computer animation and game programming and one with a background in Psychology but no animation and game expertise. Both were able to successfully create scenarios using the system and both provided useful feedback for improving the system further.

We also plan to improve the scripting system in the following ways:

- *When agent meets*: This command would trigger a behavior when one agent comes across another agent of a

given profile. Its structure would be:

"***When*** *<Profile1>* ***meets*** *<Profile2> <Command>*"

So, when an agent of profile "*Profile1*" gets close to an agent of profile "*Profile2*" it triggers the command.

A. Future Work

We identify two distinct, and important, paths for future work: the *technical development* side and the *application* side.

On the *technical development* side, we intend to make technical improvements and perform an in-depth validation, comparing our method against other state-of-the-art authoring tools and popular behavior authoring approaches, such as behavior trees. Our other goal is to make stand-alone versions of our prototype applications, so a user can test the system without installing Unreal. This will make obtaining feedback faster and easier. We are also considering features that would allow users to add their models and animations to an existing scenario without Unreal.

On the *application side*, we intend to integrate our system with an existing scenario based on the ancient pilgrimage site of Pachacamac [1]. In this context, our scripting system would be tested with archeologists to test hypotheses about how the site was used via visual simulations of large numbers of agents. This work is already in progress: Figures 17 shows a temple configured with *RegionBox*s and 18 shows characters that are controlled using the scripting system described in this paper.

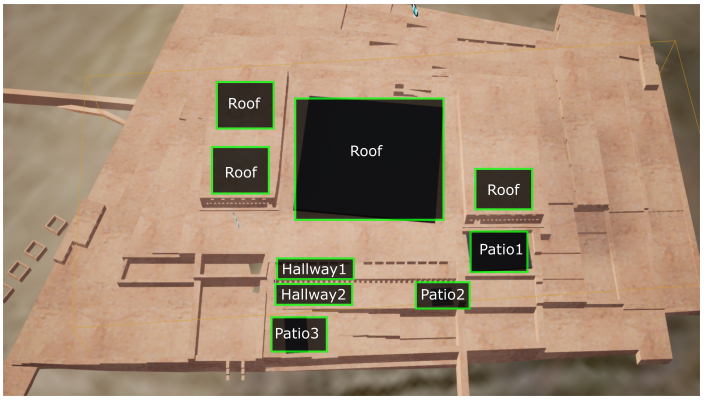

Fig. 17. Pachacamac environment and specified locations.

Fig. 18. Character evolving in the environment.

ACKNOWLEDGMENT

The authors would like to thank Luise Lindemann Kunzler for her help in testing our platform. Also, thank to CNPq and CAPES for partially funding this work.

REFERENCES

[1] K. Chow, A. Normoyle, J. Nicewinter, C. L. Erickson, and N. I. Badler, "Crowd and procession hypothesis testing for large-scale archaeological sites," in *2019 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, 2019, pp. 298–2983.

[2] M. Kallmann, J.-S. Monzani, A. Caicedo, and D. Thalmann, "Ace: A platform for the real time simulation of virtual human agents," in *Computer Animation and Simulation 2000*, N. Magnenat-Thalmann, D. Thalmann, and B. Arnaldi, Eds. Vienna: Springer Vienna, 2000, pp. 73–84.

[3] S. R. Musse and D. Thalmann, "Hierarchical model for real time simulation of virtual human crowds," *IEEE Transactions on Visualization and Computer Graphics*, vol. 7, no. 2, pp. 152–164, 2001.

[4] T. Trescak and A. Bogdanovych, "Simulating complex social behaviours of virtual agents through case-based planning," *Computers & Graphics*, vol. 77, pp. 122 – 139, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0097849318301596

[5] Z. Gao, L. Ren, Y. Qu, and T. Ishida, "Virtual space ontologies for scripting agents," in *Massively Multi-Agent Systems I*, T. Ishida, L. Gasser, and H. Nakashima, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 70–85.

[6] S. Vosinakis and T. Panayiotopoulos, "A tool for constructing 3d environments with virtual agents," *Multimedia Tools and Applications*, vol. 25, no. 2, pp. 253–279, Feb 2005. [Online]. Available: https://doi.org/10.1007/s11042-005-5607-y

[7] N. Adamo-Villani, J. Cui, and V. Popescu, "Scripted animation towards scalable content creation for elearning—a quality analysis," in *E-Learning, E-Education, and Online Training*, G. Vincenti, A. Bucciero, and C. Vaz de Carvalho, Eds. Cham: Springer International Publishing, 2014, pp. 1–9.

[8] A. de Lima Bicho, R. A. Rodrigues, S. R. Musse, C. R. Jung, M. Paravisi, and L. P. Magalhães, "Simulating crowds based on a space colonization algorithm," *Computers & Graphics*, vol. 36, no. 2, pp. 70 – 79, 2012, virtual Reality in Brazil 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0097849311001713