VISUAL SCENEMAKER INTEGRATION IN BLENDER

TIMO GÜHRING & JANNA HERRMANN

1 INTRODUCTION

In the context of the seminar "Software design and architectures for interactive avatars: a hands-on approach" we developed an integration of Visual SceneMaker[1] in Blender [2]. Visual SceneMaker is a software for creating interactive presentations specialized to users with no programming experience and Blender is a open source 3D creation suite.
The integration is mainly about avatar animation. It is possible, to create scenes in Visual SceneMaker with actors, speeches and actions. These defined action and speech activities will be sent via a UDP socket connection to Blender, in which actors perform the actions whenever a message from SceneMaker is retrieved. An actor can be an avatar itself or objects in the environment, for example a light that can be turned on, off or dimmed.

2 MANUAL

In order to make the integration of Visual SceneMaker in Blender work, please follow the steps below:

1. Change to your terminal.
2. Execute the following commands inside the terminal:

```
$ git clone git@github.com:JeannedArk/seminarsmcomm.git
$ cd ./seminarsmcomm
$ sh start.sh
```

3. Close the Blender window with the default setup. Select the Blender window with Anna as character. Press „P“ to start the game mode.
4. Visual SceneMaker will automatically open. Click „Open a project“ > select the „UDP-Forwarder“ project provided in the cloned repository > click the „select“ button > click the Play button
5. Now you can see the character moving in Blender to the corresponding actions defined by the graph.

3 IMPLEMENTATION

Our implementation presupposes some constraints especially in the naming of actions and avatars in order to work correctly. One constraint is, that the name of avatar, which should

1 http://scenemaker.dfki.de/
2 https://www.blender.org/

perform the action, is the same in Blender and Visual SceneMaker. Another one is, that the actions must be named the same in both programs.

Design Choices

Together with our implementation we had to make several design choices.
We decided to use JSON as a transport format, because it is a well known and light-weight format. Another very important fact is, that there is no need for manual parsing of the retrieved string messages on python side.
Furthermore we used a command pattern to execute the activities, because of the high grade of flexibility and dynamic and therefore it builds the base for an easy maintenance and extension in the future. Programmatically spoken the command pattern bewares the concept of separation of concerns.

Visual SceneMaker

To make the communication work, there is a class implemented in Visual SceneMaker called UDPForwarder. It parses the information from the scenes in SceneMaker in action activities and speech activities and sends them, formalized as strings in JSON representation to the localhost address on port 2300. This can be easily changed to other desired addresses and ports.

Blender

In Blender we needed an abstract class which represents the activities modeled in SceneMaker. This class is implemented by the two subclasses ActionActivity and SpeechActivity, to be consistent to the SceneMaker implementation. For executing the activities we used the command pattern. Currently, only the executing of ActionActivities is implemented.
For the communication we implemented a module called SceneMakerCommunication. Here, the data from the current running Visual SceneMaker instance is recieved and mapped to the ActionActivity or the SpeechActivity class. For communication it creates a socket with the defined configuration. The retrieved activities will be further propagated to Blender. The retrieving of the data is outsourced to another thread.
Incoming data is stored by this thread in a threadsafe queue. The update method in the communication module pops the oldest activity from the queue and executes it, if existing. Additionally the action is only played if no other action of this object is currently not playing an action. Otherwise actions would interfere with each other and stop playing the old one. Therefore we had to extend the standard python 3 threadsafe queue with a peek operation, because the standard queue only implements a get function, which automatically pops the action. The PeekQueue allows us to check the potential next action in the queue without removing it.

4 RESULTS AND FUTURE WORK

So far, only action activities of the actors were implemented as actions in Blender. Speech activities are currently sent and recognized as speech, but not interpreted as actions by the avatars in Blender. In order to do that, an integration of a Text-to-Speech software, like for example MaryTTS in Blender would be necessary.
For evaluation, we performed a stress test with a scene in which events will be sent every 5 ms. The result showed that it performed very fast on a local machine, so that there was no recognizable delay in receiving the events. Also the communication protocol can be variably

extended andsent to arbitrary network addresses.
Altogether the result of our work is a flexible and powerful solution to communicate between Visual SceneMaker and Blender.