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 non-programming users and Blender is a open source 3D creation suite.
The integration is about mainly about avatar animation. It is possible, to create scenes in Visual SceneMaker with actors, speech and actions. The action and speech together with some information and the corresponding actor will be send to blender, where an avatar (with the same name as the actor in SceneMaker) performs 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 on, off or dimmed.

2 MANUAL

In order to make the integration of the VisualSceneMaker 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 -b
    presentation
$ sh start.sh
```

3. Close the Blender window with the default setup. Select the Blender window with Anna as character. Press „P“, the game mode starts.
4. Visual SceneMaker will automatically open. Click „Open a project“ > select the „UDPForwarder“ project > click the „select“ button
5. Click the Play button
6. Now you can see the character moving to the corresponding actions defined by the graph.

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

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 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.

3.1 Design Choices

Together with our implementation we had to make several design choices.
We decided to JSON 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 strings messages on python side.
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. Other reasons to choose this command design pattern iare architectual reasons and a better overview because of the separation of concerns.

3.2 VisualSceneMaker

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 all known broadcast addresses on this machine.

3.3 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, similar to SceneMaker. For executing the activities we use a 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.

4 RESULTS AND FUTURE WORK

The result of our work is a flexible and powerful solution for the communication from Visual SceneMaker to Blender. So far, only action activities of the actors were implemented as actions in Blender. Speech activities are currently send 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 on a local machine very fast, so that there was no recognisable delay in receiving the events. The communication protocol can be variably extended. Communication can be set to arbitrary network addresses.