Erasmus+
2020-1-UK01-KA201-079177

8. Advanced Scripting

INTRODUCTION

- ✔ Objects can wait for messages in specific channels.
- ✔ You can use any channel you want from -2147483648 to 2147483647. Channel 0 is an open channel and is used whenever an avatar writes something in the ‘nearby’ chat.
- ✔ You can send messages to other channels by writing / and the number of the channel. For example “/3000 hello” will send the message “hello” to channel 3000. Avatars nearby will not see this message in chat.
- ✔ Using messages you can communicate information between different objects. You can also communicate information between parts of the same ‘linked’ object but you do not use channels for that.

MESSAGES

- To allow an object to wait for messages on a specific channel, you have to use the llListen function first, indicating the channel you want it to listen to and specific filters for the messages that can receive or the the allowed senders (e.g. specific avatar or object). This command is usually called in the state_entry event of the object..

```
integer llListen( integer channel, string name, key id, string msg );
```

MESSAGES

- ✔ To handle the incoming messages, you need to use the “listen” event.
- ✔ When a message is sent to the specified channel, the commands inside the “listen” event will run. There you can adapt the behavior you want based on the message that was received.

```
listen( integer channel, string name, key id, string message ){ ; }
```

MESSAGES

- To send a message from another object, you can use the llSay command, indicating the channel number and the message

```
llSay( integer channel, string msg );
```

MULTIPLE PARAMETERS

- ✔ Sometimes you may want to send multiple parameters through a message.
- ✔ A solution would be to create a string message that contains all the data you want to send separated by a specific character (e.g a colon ':' character).
- ✔ When you receive the message in the listen event you can split the message based on that character using the 'llParseString2List' command.

```
list llParseString2List( string src, list separators, list spacers );
```

DIALOGUE MENU

- ✔ Another way to send a message to a channel is using a dialogue menu for user.
- ✔ The command llDialog will generate a selection menu for a specific user, with a message and some options/buttons. When the user selects one of the buttons, the message is sent to a specific channel.

```
llDialog( key avatar, string message, list buttons, integer channel );
```

LINKED SETS

- ✔ Communication between parts of a set uses a similar approach, however you don't have to use channel numbers. You can specify the parts you want the message to be sent.
- ✔ To handle messages from other parts, you need to add a "link_message" event inside a script of the part that will handle the message.

```
link_message( integer sender_num, integer num, string str, key id ){;}
```

LINKED SETS

- To send a message to another part of the linked set you can use the ‘llMessageLinked’ command.
- The first argument specifies the linked set ID of the particular part you want to send the message or one of the following values (LINK_ROOT, LINK_SET, LINK_ALL_OTHERS, LINK_ALL_CHILDREN, LINK_THIS).
- Similar to ‘llSay’ you send a string message, but you can also send an integer and a key variable which is useful when you want to send multiple information.

```
llMessageLinked( integer link, integer num, string str, key id );
```

LINKED SETS

- ✔ In many cases you have some script in the ROOT part of a linked set and want to manipulate some aspects of the other members of the group (e.g their color, texture or transparency).
- ✔ Although this can be done using the above mentioned approach with messages, the LSL language offers a set of commands that you can use from scripts in the ROOT object to manipulate other parts.

LINKED SETS

- For example llSetLinkAlpha, llSetLinkColor and llSetLinkTexture can be used from the ROOT object to change the Transparency, Color and the Texture of other parts correspondingly.
- These commands are similar to the regular ones, but they have an additional argument to specify the parts that you want to manipulate.

```
llSetLinkAlpha( integer link, float alpha, integer face );

llSetLinkColor( integer link, vector color, integer face );

llSetLinkTexture( integer link, string texture, integer face );
```