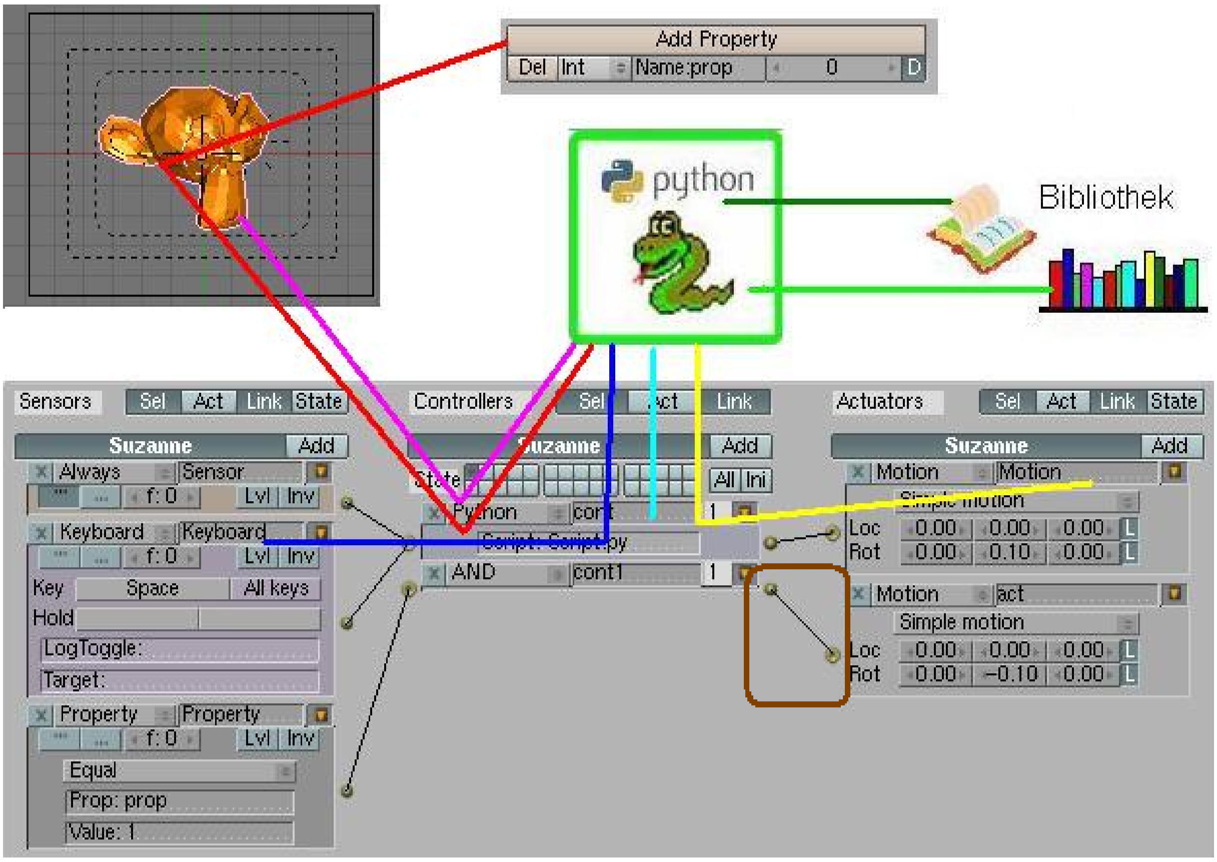
Pumpkin Run – (WSAG) – Multi Player Setup

Basics

That computer can communicate over a network; every computer needs a unique name. This unique Name is called **IP**. If a computer is bound in a local network (**LAN**), it also has a **LAN IP** besides of the **Internet IP.** The **LAN IP** can only be addressed with in the local network. The **Internet IP** from all across the World. A computer can handle more then one communication at a time. To keep the different communications organized, it structures them by using different **Ports**. If tow computers wants to communicate with each other the need a transmit/receive station. This station is called **socket**. A computer can handle more then one socket at a time.

Basics: Python to Logic Brick



(The colored lines in the picture above correspond to the same collared script lines below.)

```
from GameLogic import *
Cont = getCurrentController()
Actuator = Cont.getActuator("Motion")
Sensor = Cont.getSensor("Keyboard")
Obj = Cont.getOwner()
Property = Obj.prop
```

```
7.  if Sensor.isPositive():
8.          addActiveActuator(Actuator,1)
9.          Obj.prop = 0
10. else:
11.         addActiveActuator(Actuator,0)
12.         Obj.prop = 1 if Sensor.isPositive():
```

The first 6 lines of this scrip art he most frequent used pieces of code used in GBPS (Game Blender Python Scripts) They take care of the python can work together with the object and its logic bricks.We now will take a closer look at thise lines:

```
1.  from GameLogic import *
```

From the library we `import` the book called `GameLogic`. By importing this book we have access to all the information written inside of this book. Instead of importing the whole Book you can also just import the single pages that contain the information that is really needed in this script. This would be done with the following code:

```
1.  Import GameLogic
```

With this option you will always have to give reverence in which book the information needed can be found.

Example:

1.

```
1  from GameLogic import *
2  getCurrentController()
```

2.

```
1.  Import GameLogic
2.  GameLogic.getCurrentController()
```

See the difference? ;)

```
2.  Cont = getCurrentController()
```

This command gets the `Controller` in to the script and saves it to the variable `Cont`. The logic brick we just made accessible, is the `Controller` that gives the order to execute the python script. With this we have created the first access bridge to from the python script to the logic bricks. Over this bridge we will now make us access to the rest of the important elements.

```
3.  Actuator = Cont.getActuator("Motion")
```

Over the variable `Cont` we get access to the Controller from where we will make the next access bride to the Actuator with the name Motion. We do this whit the command `.getActuator(„Motion‟)`. The path over the Controller to the Actuator named Motion is the saved in the variable `Actuator`.

```
4.  Sensor = Cont.getSensor("Keyboard")
```

The same way as above we will now get our self access to the `Sensor` named `Keyboard` und save its access way in the variable `Sensor`.

```
5.  Obj = Cont.getOwner()
```

Other then the logic bricks, there is the object that owns all this logic brick. To get access to this object we do the fallowing:

First we will access the Controller over the Variable Cont. and the whit the code `.getOwner()` we will ask for the owner of this logic brick and save it in to the variable `Obj`.

```
6.  Property = Obj.prop
```

As a last step we would like to have access to the property of this object. The get access to this property we will have to start with the variable `Obj` and add a dot and the property name to it. (In our case that would be `.prop`) This property we will now save in with the variable `Property`.

Maybe you now ask your self why we did not use the same way to access the property as we used to access the sensor and actuator. Could we not have use the command `Cont.getProperty(”prop”)` ?

The answer is no. This is because there is no direct connection between the Controller and the property. Connections are made be connecting the to yellow dots whit a line (See the brown circle in the Picture above.) In the fallowing lines (8 -11) we will now make use of all this accesses that we have saved in variables.

```
7. ► if Sensor.isPositive():
8. ►     addActiveActuator(Actuator,1)
9. ►     Obj.prop = 0
```

In the Sensor named Keyboard we have defined the Space-Key. This means that this sensor is Positive if the Space-Key is pressed. Line 8 of this code says that if the Sensor is Positive it should execute the lines below that are indented. Line 9 the number at the end of this command decides is the Actuator should be negative or positive. (`0` = negative `1` = positive) Line 10 gives the property named `prop` the value `0`.

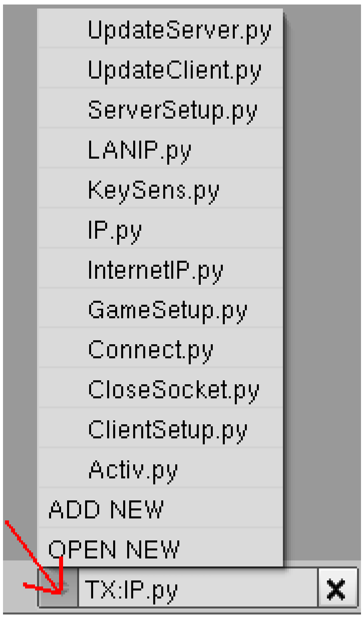
```
11.► else:
12.►         addActiveActuator(Actuator,0)
13.►         Obj.prop = 1
```

Line 11 says that if line 8 was negative it should execute the lines below that are indented. Line12 makes the `Actuator` negative by setting its value to `0`. Line 13 gives the property named `prop` the value `1`.

You can try out this script by opening the File BasicPython.blend. See what happens if you press the Space-Key wile the game engine is running?

Network setup:

Now that we know the basics on how GBPS work, we can take a closer look at the scripts that power a network game. In this tutorial we will only concentrate on the setup of a network, so I already have prepared a .blend file and we now will take a look at the important GBPS in it.
Please open the file WSAG-PumpkinRun.blend and take a look at the scripts listed below.



LANIP.py

ServerSetup.py

GameSetup.py

UpdateServer.py

ClientSetup.py

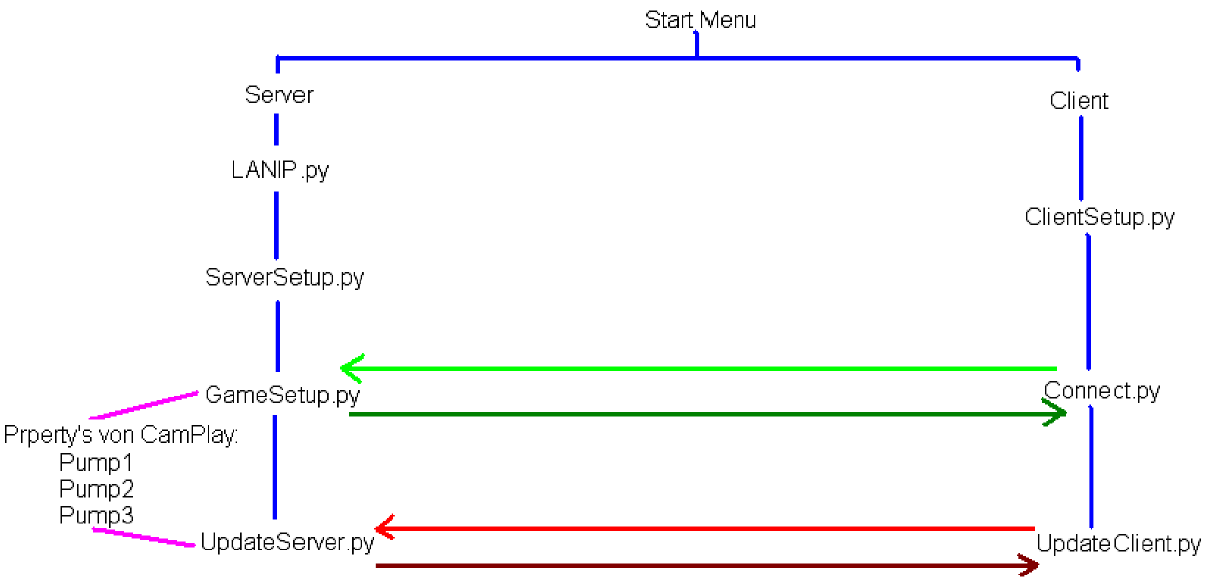
Connect.py

UpdateClient.py

CloseSocket.py

IP.py

Your first thought after look at these scripts, just might be: “Wow is this, a lot of Code, do I really want to learn this much?”. But if you take a bit closer look, you will notice that many Lines repeat. So actually you will not have to learn all to many lines of code here. Before we start to study the scripts, it’s worth wile to take some time to study the diagram below. It shows how the different scripts work together and Transport the Play Info from one Client to the server and from there to all the other Clients.



LANIP.py

This script will display the local IP when you start a server. Then you can tell all the clients to enter this IP to connect to the just started server.

```
1. ▶ from GameLogic import*
2. ▶ Cont = getCurrentController()
3. ▶ Obj = Cont.getOwner()
4. ▶ import socket
5. ▶ IP = socket.gethostbyname(socket.gethostname())
6. ▶ Obj.Text = IP
```

The first 4 line should be clear after reading the Basic above.

```
5. ▶ IP = socket.gethostbyname(socket.gethostname())
```

This code gets your local `IP` address and saves it to the variable `IP`.

```
6. ▶ Obj.Text = IP
```

It calls up the property with the name Text and gives it the value that is saved in the variable IP. Now the Text object that this property belongs to will display the IP in runtime mode.

You sure noticed that there is a second script that handles the displaying of you IP. That's second script is called InternetIP.py and as the name says it displays your Internet IP if you run the script. If you Runt the .blend file and select to be a server you can choose what script should be started the LANIP.py or the InternetIP.py. Even do the displaying of your IP is handy it not directly has something to do with the actual network communication, that's why we wont explain this second script any closer here. If you wish to find more Info about it you can read the chapter (27)WSAG – Server Mod – Display Your IP in the WSAG tutorial. The WSAG tutorial that contains a lot of Information about network programming with Blender can be downloaded at www.wsag.ch.vu

ServerSetup.py

This script will setup all the Sockets that the server will need to communicate whit the clients.

```
1.  ▶ from GameLogic import *
2.  ▶ from socket import *
3.  ▶ cont = GameLogic.getCurrentController()
4.  ▶ Obj = cont.getOwner()

5.  ▶ if Obj.OneTime == 0:
6.  ▶         host = ''
7.  ▶         MainPort = 10000
8.  ▶         GameLogic.sMain = socket(AF_INET,SOCK_DGRAM)
9.  ▶         GameLogic.sMain.bind((host,MainPort))
10.▶         GameLogic.sMain.setblocking(0)

11.▶         Pump1Port = 10010
12.▶         GameLogic.sPump1 = socket(AF_INET,SOCK_DGRAM)
13.▶         GameLogic.sPump1.bind((host,Pump1Port))
14.▶         GameLogic.sPump1.setblocking(0)

15.▶         Pump2Port = 10020
16.▶         GameLogic.sPump2 = socket(AF_INET,SOCK_DGRAM)
17.▶         GameLogic.sPump2.bind((host,Pump2Port))
18.▶         GameLogic.sPump2.setblocking(0)

19.▶         Pump3Port = 10030
20.▶         GameLogic.sPump3 = socket(AF_INET,SOCK_DGRAM)
21.▶         GameLogic.sPump3.bind((host,Pump3Port))
22.▶         GameLogic.sPump3.setblocking(0)
23.▶         Obj.OneTime = 1
```

By just counting the lines of this script it can hit you like a brick wall. There are 23 Lines which we have to learn. But as I already said earlier, if you look closer there is not all that much new stuff here.

The aqua colored lines (1 – 4) are the standard Basic codes, so nothing new here. Then we have the red colored lines (5, 6, 23) these are new and unique within this script. The light green lines (7 – 10) are new, but will repeat themselves in the dark green lines (11 -22).

So the only lines we really have to pay any attention to are 5 – 10 and 23. So that's only 7 lines any you already will understand all of this script.

```
5.  ▶ if Obj.OneTime == 0:
23.▶     Obj.OneTime = 1
```

Line 5 calls up the property `OneTime` and if its value is equal to `0` it will execute the lines below that are indented.

Line 23 will change the value of the property `OneTime` to `1` By changing the property we make sure that the indented lines below line 5 only run onetime.

6. ► `host = ''`

Saves a empty text holder in to the variable `host`.

The purpose of this empty text holder is do make a place where later the computer name can be written in to. So you may ask; why not write the computer name in to it right away? Well you could do that but then this script would only run on your own computer. So that's why we leave it empty and let the socket fill it out later all by it self. You also would have the option to write localhost between the „”“, that would say the script right a way to enter the name of the local computer in there… But why write more then necessary?

7. ► `MainPort = 10000`

Saves the value `10000` in to the variable `MainPort`. This value will bet the port number over witch the socket will communicate. That's why you will have to make sure that your Firewall and your Router won't block this and the other ports that will be added later.

Information on how you can setup your Firewall and router to Forward information overt this Port, can be found in the Manuals of your Firewall and Router. If any questions or problems come up then you can ask them in the gameblender.org Forum in this link http://www.gameblender.org/viewforum.php?f=21

8. ► `GameLogic.sMain = socket(AF_INET,SOCK_DGRAM)`

The code socket will create a socket that uses the protocol that is defined inside of the brackets and saves it to the global variable `GameLogic.sMain`. `AF_INET,SOCK_DGRAM` is the definition for a UDP protocol. We will use this protocol because it is very fast and will keep on working even if it once can't send or receive a packet.

9. ► `GameLogic.sMain.bind((host,MainPort))`

The Code `.bind((host,Mainport))` is used to bind the information that we have saved in to variable `host` and `Mainport` into the socket. Now the socket knows on what computer and over witch port he should communicate over.

10. ► `GameLogic.sMain.setblocking(0)`

The socket will already work with the lines above. But as I already mentioned by line 8 we would like to make sure that the script will keep on running even if the socket did not send or receive a packet. This can happen very easy, if the connection having a problem, the Client and server have different streaming rates the connection is temporary used by a other program… That's why we will use the code `.setblocking(0)`. The `0` in the brackets will tell the script that it should not block even if it did not send or receive anything. If you would but a 1 there, the script would try to receives /send a packet until it has had success. If you would do this everything would be as slow as the slowest client that is connected with the server.

The lines 11 – 22 (dark green) repeat the lines 7 -10 (light green). The only difference is the port numbers and the socket names. Every client will have a own socket and port to communicate with the server. This way the script can work much efficient, then if it would have to do send and receive all the information of the different Clients over one single port and Socket.

So now we have a Main socket that has the duty to pick up the contact whit the Clients and let them know if there is still a free player available and over what port the Client should communicate from now on. And then we have the Pump1 – 3 sockets that are here to handle the communication of the players 1 – 3.

GameSetup.py

```
1.  ► from GameLogic import *
2.  ► from socket import *
3.  ► from cPickle import *
4.  ► cont = GameLogic.getCurrentController()
5.  ► obj = cont.getOwner()

6.  ► try:
7.  ►         Data, CLIP = GameLogic.sMain.recvfrom(1024)

8.  ►         if obj.Pump1 == 0:
9.  ►              Info = 1
10.►              Data = dumps((Info))
11.►              GameLogic.sMain.sendto(Data, CLIP)
12.►              obj.Pump1 =1

13.►         elif obj.Pump2 == 0:
14.►              Info = 2
15.►              Data = dumps((Info))
16.►              GameLogic.sMain.sendto(Data, CLIP)
17.►              obj.Pump2 = 1

18.►         elif obj.Pump3 == 0:
19.►              Info = 3
20.►              Data = dumps((Info))
21.►              GameLogic.sMain.sendto(Data, CLIP)
22.►              obj.Pump3 = 1

23.►         else:
24.►              Info = "Server Full"
25.►              Data = dumps((Info))
26.►              GameLogic.sMain.sendto(Data, CLIP)

27.► except:
28.►         pass
```

The aqua colored lines (1 – 5) are the well know basic codes. The only question here is what information have we imported with the `cPickle` book?: `cPickle` hold the Information on how you can pack Data in to a container (something similar to a Zip program) so that it can be send save over the network, making sure that when the receiver unpacks the container he gets the date in the same order as it was packed.

```
6.  ► try:
7.  ►         Data, CLIP = GameLogic.sMain.recvfrom(1024)

27.► except:
28.►         pass
```

The red Lines start and end whit the trying of receiving Date over the `sMain` socket. Lines 27 and 28 tell the script to just pass on if any of the indented lines below line 6 fail. The only line that actually can fail is the Line 7. It will fail when it tries to receive and there is no Date there to receive. This can for example happen if no Client is sending any Data. Now we will take a little closer look at the line 7: The variable `GameLogic.sMain` contains the information about the socket. The code `.recvfrom` commands the socket to receive data. `(1024)` defines the maximal size, that the buffer can receive at once. Every time we receive Date we get 2 Blocks. The first block contains the data and is saved in the variable `Data`. The second block contains sender address and is saved in the variable `CLIP` (= Client IP)

```
8. ▶            if obj.Pump1 == 0:
```

Checks if the property `Pump1` is equal the value `0`. If this is true the indented line below will be executed.

```
9. ▶                 Info = 1
```

Saves the value`1` into the variable `Info`.

```
29.▶                 Data = dumps((Info))
```

Now we will use the information from the cPickle book for the first time. The command `dumps` will pack any data inside the double brackets in to a container and save it to the variable `Data`. Important you have to use double brackets here "`((Info))`" and not single brackets "(Info)"

```
11.▶                 GameLogic.sMain.sendto(Data, CLIP)
```

To send Data we use `GamLogic.sMain` to call up the socket and give it the command `.sendto`. If you send Data it's just as in line 7, you need 2 things: The Data and the address that this packed should be send to. This info you put inside the brackets.

```
12.▶            obj.Pump1 =1
```

The property `Pump1` gets the value `1` so that the server knows that this player is used by a Client.

The dark green Lines are a repenting the just explained light green lines. The `elif` command is used instead of the `if` command. This way the indented lines below will only be executed if the `if/elif` command above was negative.

UpdateServer.py

So now we get to the last script on the server side. It's the heart piece of communication on the server side. It makes sure that the game stays up to date on the server and all the clients. For we have a longer bit of code here, it was divided in to chapters. Chapter titles can be written right in to the script, by just adding an "#" in front of that line. The "#" tells the script that this information should not be used when the script runs.

```
#---------------IMPORT-----------------#
   1. ▶ from GameLogic import *
   2. ▶ from socket import *
   3. ▶ from cPickle import *
   4. ▶ cont = GameLogic.getCurrentController()
   5. ▶ obj = cont.getOwner()

#----------------DATA------------------#
   6. ▶ scene = getCurrentScene()
   7. ▶ objPump1 = scene.getObjectList()["OBPump1"]
   8. ▶ objPump2 = scene.getObjectList()["OBPump2"]
   9. ▶ objPump3 = scene.getObjectList()["OBPump3"]
   10.▶ PosPump1 = [0,0,0]
   11.▶ OriPump1 = [[0,0,0],[0,0,0],[0,0,0]]
   12.▶ PosPump2 = [0,0,0]
   13.▶ OriPump2 = [[0,0,0],[0,0,0],[0,0,0]]
   14.▶ PosPump3 = [0,0,0]
   15.▶ OriPump3 = [[0,0,0],[0,0,0],[0,0,0]]
```

#-------------------RECEIVE-------------------#

```
Pump1 = 1
try:
         DataPump1, CLIPPump1 = sPump1.recvfrom(1024)
         obj.Pump1 = 1
except:
         if obj.Pump1 != 0:
              obj.Pump1 = obj.Pump1 + 5
         if obj.Pump1 > 500:
              objPump1.setPosition([-15, 10, 8])
              obj.Pump1 = 0
         Pump1 = 0

Pump2 = 1
try:
         DataPump2, CLIPPump2 = sPump2.recvfrom(1024)
         obj.Pump2 = 1
except:
         if obj.Pump2 != 0:
              obj.Pump2 = obj.Pump2 + 5
         if obj.Pump2 > 500:
              objPump2.setPosition([-12, 13, 6])
              obj.Pump2 = 0
         Pump2 = 0

Pump3 = 1
try:
         DataPump3, CLIPPump3 = sPump3.recvfrom(1024)
         obj.Pump3 = 1
except:
         if obj.Pump3 != 0:
              obj.Pump3 = obj.Pump3 + 5
         if obj.Pump3 > 500:
              objPump3.setPosition([-11, 10, 4])
              obj.Pump3 = 0
         Pump3 = 0
```

#---------------Set Game Play----------------#

```
Pump1Shoot = 0
Pump2Shoot = 0
Pump3Shoot = 0

if Pump1 == 1:
         UPPump1 = loads(DataPump1)
         PosPump1 = [UPPump1[0][0],UPPump1[0][1],UPPump1[0][2]]
         OriPump1 = [UPPump1[1][0],UPPump1[1][1],UPPump1[1][2]]
         objPump1.setPosition(PosPump1)
         objPump1.setOrientation(OriPump1)
         objPump1.KeySens = UPPump1[2]
         Pump1Shoot = UPPump1[2]
```

```
60.▶ if Pump2 == 1:
61.▶         UPPump2 = loads(DataPump2)
62.▶         PosPump2 = [UPPump2[0][0],UPPump2[0][1],UPPump2[0][2]]
63.▶         OriPump2 = [UPPump2[1][0],UPPump2[1][1],UPPump2[1][2]]
64.▶         objPump2.setPosition(PosPump2)
65.▶         objPump2.setOrientation(OriPump2)
66.▶         objPump2.KeySens = UPPump2[2]
67.▶         Pump2Shoot = UPPump2[2]

68.▶ if Pump3 == 1:
69.▶         UPPump3 = loads(DataPump3)
70.▶         PosPump3 = [UPPump3[0][0],UPPump3[0][1],UPPump3[0][2]]
71.▶         OriPump3 = [UPPump3[1][0],UPPump3[1][1],UPPump3[1][2]]
72.▶         objPump3.setPosition(PosPump3)
73.▶         objPump3.setOrientation(OriPump3)
74.▶         objPump3.KeySens = UPPump3[2]
75.▶         Pump3Shoot = UPPump3[2]

#------------------SEND------------------#
76.▶ PosPump1 = objPump1.getPosition()
77.▶ OriPump1 = objPump1.getOrientation()
78.▶ PosPump2 = objPump2.getPosition()
79.▶ OriPump2 = objPump2.getOrientation()
80.▶ PosPump3 = objPump3.getPosition()
81.▶ OriPump3 = objPump3.getOrientation()
82.▶ Data = dumps((PosPump1,OriPump1,Pump1Shoot,PosPump2,OriPump2,Pump2Shoot,PosPump3,OriPump3,Pump3Shoot))
83.▶ if Pump1 == 1:
84.▶         GameLogic.sPump1.sendto(Data, CLIPPump1)
85.▶ if Pump2 == 1:
86.▶         GameLogic.sPump2.sendto(Data, CLIPPump2)
87.▶ if Pump3 == 1:
88.▶         GameLogic.sPump3.sendto(Data, CLIPPump3)
```

First chapter (Import):

Everything here should be clear by now.

Second chapter (Data):

We will setup variables that contain all the data that we will need in the fallowing lines of code.

```
6. ▶ scene = getCurrentScene()
```

The command `getCurrenScene()` will get all the elements of the current scene in to the script. These elements will be saved in the variable `scene`.

```
7. ▶ objPump1 = scene.getObjectList()["OBPump1"]
```

The variable `scene` is containing all the elements of the current scene, now we would like to have access to all the Objects in the scene, this will happen by using the command `.getObjectList()`. In the square brackets we write the name of the object that we would like to save to the variable `objPump1`. Its important that in front of the Objects name you ad the letters `OB`. Wow now you have made your self access to a object that is not even linked to the script. The dark green lines (8&9) will save the other to player objects to variables.

```
10.► PosPump1 = [0,0,0]
```

In the variable `PosPump1` we provide a list with 3 segments. This list later will be filed out with the coordinates of the Player. The 3 segments `[0,0,0]` stand for X, Y, Z.

```
11.► OriPump1 = [[0,0,0],[0,0,0],[0,0,0]]
```

This list is a bit more complex. It contains 3 x 3 segments. This will be used to save the orientation of the player. In Blender the orientation of a object is calculated with the 3x3 Matrix. If you wish to get more info about the 3x3 Matrix you can read the chapter **(23)WSAG – Gameplay – Setting up the Players – 3 x 3 Matrix** in the WSAG tutorial. This tutorial can be downloaded at www.wsag.ch.vu

The dark green lines (12 -15) do provide the lists for the Player 2 & 3 (Pump 2 & 3) Third chapter (Receive):

The data of the individual clients will be received and at the same time the script will check if the Client is still active.

```
16.► Pump1 = 1
```

Gives the variable `Pump1` the value `1`

```
17.► try:
18.►         DataPump1, CLIPPump1 = sPump1.recvfrom(1024)
19.►         obj.Pump1 = 1
```

This lines should feel similar by now. The script will try to receive data from the Port `sPump1`. If it receives data the property Pump1 is set to 1.In the variable `DataPump1` the revived data will be saved and in the `CLIPPump1` variable the senders address.

```
20.► except:
```

This command works together whit the try command from line 17. if the indented lines below the try command fail then the indented lines below the except command will be executed.

```
21.►         if obj.Pump1 != 0:
22.►              obj.Pump1 = obj.Pump1 + 5
```

If the property `Pump1` is not equal `0` then this properties value will be added `5`.

```
23.►         if obj.Pump1 > 500:
24.►              objPump1.setPosition([-15, 10, 8])
25.►              obj.Pump1 = 0
```

If the property `Pump1` is bigger then `500` the players coordinates will be set `([-15, 10, 8])` and the property `Pump1` will get the value `0`. This Lines decides that if more the 100 the try of receiving date from the Port for Client 1 (sPump1) the player is disconnected and its port is free for a other Client to use. In other words it is a very simple disconnecting system.

```
26.►         Pump1 = 0
```

Gives the variable `Pump1` the value `0`. As you sure have noticed in line 16 the variable `Pump1` was given the value 1. The value if this property shows if data was received for this client or not, this is important to know, because it shows us for witch Player we will have to process data.

In the dark green lines (27 -48) as usual the the same we just looked at for player 1 is done for the other 2 players.

Forth chapter (Set Game Play):

We will use the received data to keep the game up to date for each player.

```
49.► Pump1Shoot = 0
50.► Pump2Shoot = 0
51.► Pump3Shoot = 0
```

This lines give the variables `Pump1Shoot`, `Pump2Shoot` and `Pump3Shoot` the value `0`. Because we will use this variables later its important that they have a value. Because you can only process a variable if it has a value.

```
52.► if Pump1 == 1:
```

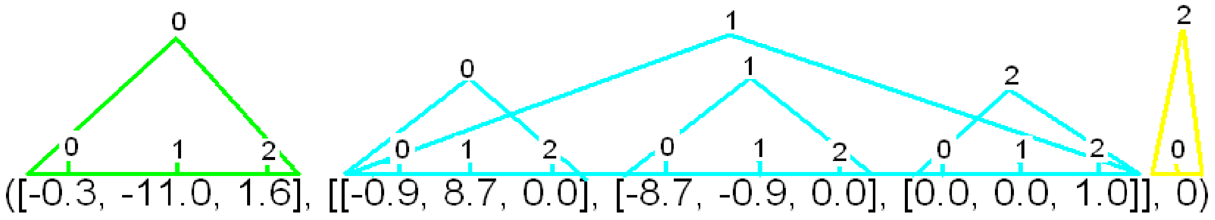
The principle of this line is clear by now – right. If the value of Pump1 is 1, it means that we received date from player 1 (see line 19)

```
53.►         UPPump1 = loads(DataPump1)
```

The received date will be unpacked and saved to the variable UPPump1, We remember that the data we send over the Network is packed in to a container. The loads command here unpack any data that is inside the brackets behind it.

The date we will have do process here look like this:

([-0.3, -11.0, 1.6], [[-0.9, 8.7, 0.0], [-8.7, -0.9, 0.0], [0.0, 0.0, **1.0**]], 0) The first block (light green marked) give the coordinates of the player (X,Y,Z)
The second block (aqua colored) give the orientation of the player as a 3x3 Matrix.
The third block (yellow) gives us the info if the player is shooting = value 1 or not value 0.
The blocks in this list are assigned as fallowed:
The numbering of the blocks starts with 0. So The light green block is assigned whit the value 0, the aqua colored block with 1 and the yellow block with 2. So if we would like to assign the number 1.0 in the aqua colored block (it is underlined and bold in the example above), we first assign the aqua colored block with Block[1] an the we assign the third block in side the aqua colored block with Block[0][2]. Now we assign the third number in side this block Block[0][2][2] So you see we peal away layer by layer until we reach the information we want. The picture below tries to show this principle in a graphical way:



```
54.►         PosPump1 = [UPPump1[0][0],UPPump1[0][1],UPPump1[0][2]]
```

The light green data block (Block[0]) is being saved in to the List that we have created earlier in the variable `PosPump1.`

UPPump1[0][0] = Position on the world axis X UPPump1[0][1] = Position on the world axis Y UPPump1[0][2] = Position on the world axis Z

```
55.►         OriPump1 = [UPPump1[1][0],UPPump1[1][1],UPPump1[1][2]]
```

As in the line above we save the data Block[1] in to the list in variable `OriPump1`. For more information what the single numbers mean you will have to study the chapter about 3x3 Matrix, mentioned above.

```
56.►         objPump1.setPosition(PosPump1)
57.►         objPump1.setOrientation(OriPump1)
```

The lists we just have filed out are now used to set the orientation and position of the Player.

```
58.▶            objPump1.KeySens = UPPump1[2]
59.▶            Pump1Shoot = UPPump1[2]
```

The value in `UPPump1[2]` is assigned to the property `KeySens` and is also saved in the variable `Pump1Shoot`.

You remember at the beginning of this script we gave this variable the value 0. Now it gets a the value 0 or depending on if the client is pressing the shoot button or not.

So as always in the dark green lines (60 – 75) the same is repeated for the player 2 and 3.

Fifth chapter (Send):

The up to date player data will be send to all of the Clients.

```
76.▶ PosPump1 = objPump1.getPosition()
77.▶ OriPump1 = objPump1.getOrientation()
```

Gets the position and orientation of the player and saves it to variables. Lines 78 – 81 do the same thing to the other 2 clients

```
82.▶ Data=
   dumps((PosPump1,OriPump1,Pump1Shoot,PosPump2,OriPump2,Pump2Shoot,PosPump3,OriPump3,Pump3Shoot))
```

The data that we just saved to the different Variables will now be packed in to a container.

```
83.▶ if Pump1 == 1:
84.▶            GameLogic.sPump1.sendto(Data, CLIPPump1)
```

First we check if we received data from Client 1. You remember that we used the variable `Pump1` to save the value `0` or `1` depending if we received data or not. If data was received we have the senders address and now can send the data to that address.

So now we have studied all the scripts that are relevant to setup a Server. Next we will look at the Client setup.

Because a lot of what we just have studied will be reused in the Client scripts, I will not explain them again. The lines that are repeated from the server code will be marked gray. Also the dark green and aqua colored lines will only be mention if the contain something that is not clear yet.

ClientSetup.py

```
1. ▶ from GameLogic import *
2. ▶ from socket import *
3. ▶ cont = GameLogic.getCurrentController()
4. ▶ Obj = cont.getOwner()

5. ▶ Obj.Connect = 120
6. ▶ if Obj.OneTime == 0:
7. ▶            ServerIP = GameLogic.IP
8. ▶            Serverport = 10000
9. ▶            Clientname = ''
10.▶            ClientPort = 10001
11.▶            GameLogic.sStart = socket(AF_INET,SOCK_DGRAM)
12.▶            GameLogic.sStart.bind((Clientname,ClientPort))
13.▶            GameLogic.host = (ServerIP,Serverport)
14.▶            GameLogic.sStart.setblocking(0)
15.▶            Obj.OneTime = 1
```

Just as on the server side, the socket are setup. (at first only the socket that will be used to contact the server)

7. ► `ServerIP = GameLogic.IP`

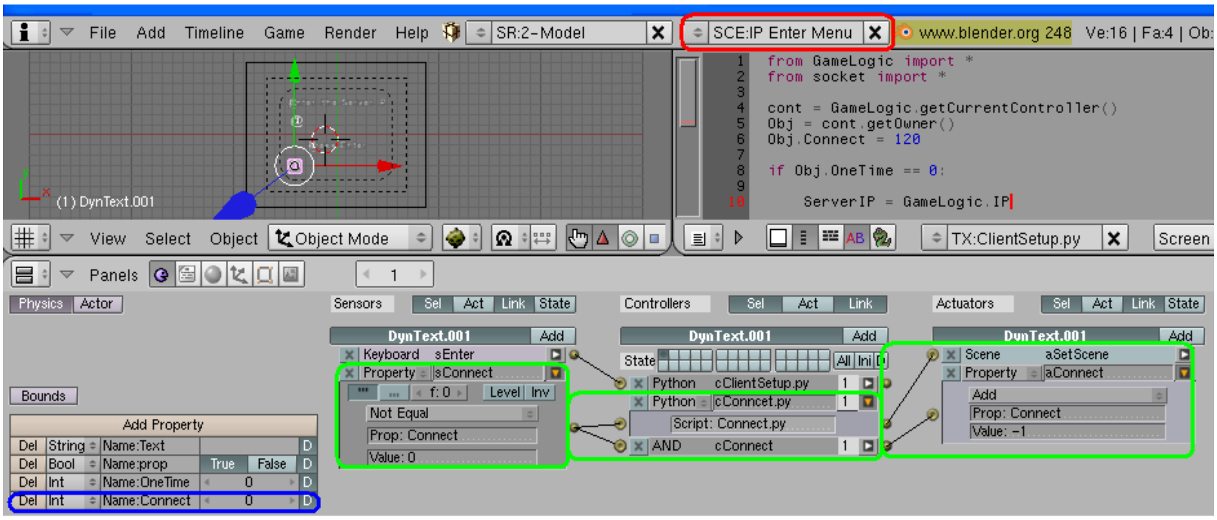
In the script IP.py the IP address that was entered with the keyboard will be saved in the global variable `GameLogic.IP`. The Global variable is now changed in to a local variable. Global variables are the ones that start with `GameLogic.` Of course we could use the global variable right away. But often it makes sense to transform the global variable to a local variable. The reason to do this can be to keep things more ordered or if you would like to change the value of the variable with out affecting the Global variable. Global variables can be accessed from any script in any scene, local variables only within the own script.

8. ► `Serverport = 10000`

The port that we will use to contact the server is saved in the variable `Serverport.` This number has to be the same as we used in the ServerSetup.py script to setup the sMain port for the server. It has to be the same because the server will only listen on this script for clients that would like to join the game.

5. ► `Obj.Connect = 120`

That this command gives the property `Connect` the value `120` should be clear – but what effect will this have? To understand this you will have to open the scene "IP Enter Menu" (Red circle in the picture below) There you select the Text plane named DynText.001 (It's the selected Object in the 3D view). Now you can see the property with the name Connect that has a standard value of 0 (Blue circle. In the green circle total left you can see that the property Sensor will start to pulse if the property value is not equal to 0. These Sensor impulse starts the script Connect.py and the actuator Connect that will reduce the Property value by 1 every time it is activated. With this the connect script will be run until the property Connect has the value 0 again.



```
from GameLogic import *
from socket import *

cont = GameLogic.getCurrentController()
Obj = cont.getOwner()
Obj.Connect = 120

if Obj.OneTime == 0:

    ServerIP = GameLogic.IP
```

Del	String	Name:Text			D
Del	Bool	Name:prop	True	False	D
Del	Int	Name:OneTime	0		D
Del	Int	Name:Connect	0		D

Sensors:
Keyboard sEnter
Property sConnect
f: 0 Level Inv
Not Equal
Prop: Connect
Value: 0

Controllers:
State
Python cClientSetup.py 1
Python cConnect.py 1
Script: Connect.py
AND cConnect 1

Actuators:
Scene aSetScene
Property aConnect
Add
Prop: Connect
Value: -1

Connect.py

So now we will see what this script will try to accomplish in these 120 impulses.

1. ► `from GameLogic import *`
2. ► `from socket import *`
3. ► `from cPickle import *`
4. ► `cont = GameLogic.getCurrentController()`
5. ► `Obj = cont.getOwner()`
6. ► `Actuator = cont.getActuator("aSetScene")`

```
try:
        Info = "connect"
        Data = dumps((Info))
        GameLogic.sStart.sendto(Data, GameLogic.host)
        Data, SRIP = GameLogic.sStart.recvfrom(1024)
        Info = loads(Data)
        GameLogic.Activ = Info

        if Info == 1:
            ServerIP = GameLogic.IP
            Serverport = 10010
            Clientname = ''
            ClientPort = 10011
            GameLogic.sPlayPort = socket(AF_INET,SOCK_DGRAM)
            GameLogic.sPlayPort.bind((Clientname,ClientPort))
            GameLogic.host = (ServerIP,Serverport)
            GameLogic.sPlayPort.setblocking(0)
            GameLogic.sStart.close()
            Actuator.setScene("Play")
            addActiveActuator(Actuator,1)

        if Info == 2:
            ServerIP = GameLogic.IP
            Serverport = 10020
            Clientname = ''
            ClientPort = 10021
            GameLogic.sPlayPort = socket(AF_INET,SOCK_DGRAM)
            GameLogic.sPlayPort.bind((Clientname,ClientPort))
            GameLogic.host = (ServerIP,Serverport)
            GameLogic.sPlayPort.setblocking(0)
            GameLogic.sStart.close()
            Actuator.setScene("Play")
            addActiveActuator(Actuator,1)

        if Info == 3:
            ServerIP = GameLogic.IP
            Serverport = 10030
            Clientname = ''
            ClientPort = 10031
            GameLogic.sPlayPort = socket(AF_INET,SOCK_DGRAM)
            GameLogic.sPlayPort.bind((Clientname,ClientPort))
            GameLogic.host = (ServerIP,Serverport)
            GameLogic.sPlayPort.setblocking(0)
            GameLogic.sStart.close()
            Actuator.setScene("Play")
            addActiveActuator(Actuator,1)

        if Info == "Server Full":
            Obj.Text = "Server Full"
            Obj.Connect = 0
```

```
53.▶ except:
54.▶         pass

55.▶ if Obj.Connect == 100:
56.▶         Obj.Text = "Connecting"
57.▶ if Obj.Connect == 5:
58.▶         Obj.Text = "Connection Failed"
```

Wow almost nothing new here – almost everything is gray here. This means you have already learned a lot about network programming. So will make a summary of what is done in this lines: Line 1 – 6 :Basic Setup that lets us interact with the Object and its logic bricks. Line 7 – 13: Tries to send data to the Server and receive its answer. Line 14 – 25: If the answer of the Serve is equal to 1, then the socket for Player one will be opened and the contact socket (`sStart`) will be closed. The next scene will be loaded. Line 26 – 49: Repeats the lines 14 -25, fort he other 2 player. Line 50 -54: If the server communicates that there is no free player available the message `Server Full` will be displayed. Line 55 – 58: If the property connect is equal to `100`, the message `Connecting` is displayed. If its value is equal to `5`, it will display `Connection Failed`.

So now we will take a closer look at the light green and red lines in this script.

```
23.▶                 GameLogic.sStart.close()
```

The socket `sStart` will be cloesed with the command `.close()`.

```
24.▶                 Actuator.setScene("Play")
```

The actuator is called up and receives the command `.setScene`. With this the actuator is set to load the scene that is in the brackets.

```
25.▶                 addActiveActuator(Actuator,1)
```

With the command `addActiveActuatort` the actuator that is written inside of the brackets will be set active or inactive, depending on the value that is fallowed. 0 = inactive, 1 = active. By setting the Actuator that we just have set to load the scene Play to active the loading process will be activated.

```
50.▶         if Info == "Server Full":
```

If the message received from the server (That has been saved to the variable Info) is equal to `Server Full`, the indented lines below will be activated..

```
51.▶                 Obj.Text = "Server Full"
```

The property Text will be assigned with the String `Server Full`. This String will be displayed on the Text plane during runtime.

By the way a string is a Text other then a mathematics value.

```
52.▶                 Obj.Connect = 0
```

Gives the property Connect the value 0. This will stop this script. (The script only runs if the Property Sensor is sending signals (pulses) and it only dose so if the value property Connect is not equal to 0. By stopping this script you get the chance to retry to connect the server or enter a new IP.

UpdateClient.py

Now that we have a connection to the server we will have to keep the game up to date and synchronized with the other Clients.

```
#---------------IMPORT-----------------#
from GameLogic import *
from socket import *
from cPickle import *
cont = GameLogic.getCurrentController()
obj = cont.getOwner()

#----------------DATA------------------#
Shoot = 0
if obj.KeySens == 1:
         Shoot = 1

PosYou = obj.getPosition()
OriYou = obj.getOrientation()

scene = getCurrentScene()
objPump1 = scene.getObjectList()["OBPump1"]
objPump2 = scene.getObjectList()["OBPump2"]
objPump3 = scene.getObjectList()["OBPump3"]
PosPump1 = [0,0,0]
OriPump1 = [[0,0,0],[0,0,0],[0,0,0]]
PosPump2 = [0,0,0]
OriPump2 = [[0,0,0],[0,0,0],[0,0,0]]
PosPump3 = [0,0,0]
OriPump3 = [[0,0,0],[0,0,0],[0,0,0]]

#------------RECEIVE/SEND--------------#
Data = dumps((PosYou,OriYou,Shoot))
GameLogic.sPlayPort.sendto(Data,GameLogic.host)

rec = 1

try:
         Data, SRIP = GameLogic.sPlayPort.recvfrom(1024)
except:
         rec = 0

if rec ==1:
         if Data:
              UPData = loads(Data)
         if obj != objPump1:
              PosPump1 = [UPData[0][0],UPData[0][1],UPData[0][2]]
              OriPump1 = [UPData[1][0],UPData[1][1],UPData[1][2]]
              objPump1.setPosition(PosPump1)
              objPump1.setOrientation(OriPump1)
              objPump1.KeySens = UPData[2]
```

```
37.►        if obj != objPump2:
38.►            PosPump2 = [UPData[3][0],UPData[3][1],UPData[3][2]]
39.►            OriPump2 = [UPData[4][0],UPData[4][1],UPData[4][2]]
40.►            objPump2.setPosition(PosPump2)
41.►            objPump2.setOrientation(OriPump2)
42.►            objPump2.KeySens = UPData[5]

43.►        if obj != objPump3:
44.►            PosPump3 = [UPData[6][0],UPData[6][1],UPData[6][2]]
45.►            OriPump3 = [UPData[7][0],UPData[7][1],UPData[7][2]]
46.►            objPump3.setPosition(PosPump3)
47.►            objPump3.setOrientation(OriPump3)
48.►            objPump3.KeySens = UPData[8]
```

WoW just one command that we not have studied jet in this script

```
29.►        if Data:
```

This command says nothing else then if the variable `Data` exists and has a value assigned to it. The intended lines below should be executed. This variable exists if you have received data from the server. (received data is saved in the variable `Data`)

A quick summary of this script:

Line 1 – 5:Basic Code. Line 6 – 10: Gets info about your own player. Saves position, orientation and if the shooting key is pressed to variables.
Line 11 – 14: Gets all the players of the current scene in to the script and saves hem to variables.
Line 15 -20: Provides the lists that we will use later, to save the current status of the players.
Line 21 + 22: Pack's the data of your player in to a container so that it can be send over the network. Line 23 Line 27: Tries to receive data from the server. If it receives data the variable `rec` gets the value `1` otherwise `0`.
Line 28: All the intended lines below will be executed if the variable `rec` has the value `1`.
Line 29 + 30: The received container is unpacked.
Line 31 – 36: The lists are filed out with the received date. Then the Lists are used to actualize the players.
Line 37 – 48: Repeats the line 31 – 36, fort he other 2 players.

The last 2 Steps are to disconnect from the server (close sockets) and to recommend the last entered IP in the scene "IP Enter Menu". (So that you can quick reconnect to the server.

A small explaining to these:

If the Live of a player is over (Candle is burned down), it changes to the scene "Game Over" and disconnects from the server. There you have the option to stop playing or to go to the scene „IP Enter Menu" where you can reconnect to the server.

When the „Game Over" scene is loaded the script CloseSocket.py will be activated. If you press the R-Key the scene „IP Enter Menu" will be loaded and the script IP.py will be executed. So lets see what these last 2 scripts do:

CloseSocket.py

```
from GameLogic import *
cont = GameLogic.getCurrentController()
Obj = cont.getOwner()

GameLogic.sPlayPort.close()
```

All lines in this script should be clear. It simply closes the “sPlayPort” socket.

IP.py

```
1. ▶ from GameLogic import *
2. ▶ cont = GameLogic.getCurrentController()
3. ▶ obj = cont.getOwner()

4. ▶ if obj.OneTime == 0:
5. ▶ try:
6. ▶          obj.Text = GameLogic.IP
7. ▶          obj.OneTime = 1
8. ▶ except:
9. ▶          pass

10.▶ GameLogic.IP = obj.Text.
```

Every command in this script is known by now, so I only will do a small summary: Lines 4 – 9: Will try one Time to give the property `obj.Text` the value saved in `GameLogic.IP`. It’s imported that this command will only run one time, because in line 10 the value of the property obj.Text is saved to the variable Gamelogic.IP. So if lines 4 – 9 would run every time the commands is line 6 and 10 would loop and overwrite each other so that the text entry would always stay empty, instead of changing to what you enter by keyboard. The lines 4 – 9 only have the purpose to recommend the last IP entered after a game over, to speed up the reconnection.

Final Words

Congratulation now you know all the basics of a network setup. Of course there is a lot more you can do within a good multi player game then we have looked at here.

For example:

- Dynamic number of players (Unlimited number of Players)
- Pickups
- Multi level handling
- Different types of player to choose from
- and a lot more

For more Information about multi player network setups go to www.wsag.ch.vu
If you have any questions, problems or ideas just post them at: http://www.gameblender.org/viewforum.php?f=21