Two-Player Interactive Game, Computer Networking

Disclaimer

It is only fair to admit that Python Network Programming is not my core competence. Some code lines are still a bit unclear to me.
What I present here is very simple and probably not an optimal code. But it works. I'll be happy to get feedback and suggestions improving the performance.

Background

With this tutorial, I move into an area where I am still a novice. I have, however, spend quite long time figuring out how to connect two computers for interactive gaming. My code works and there is no other tutorial that covers the subject.

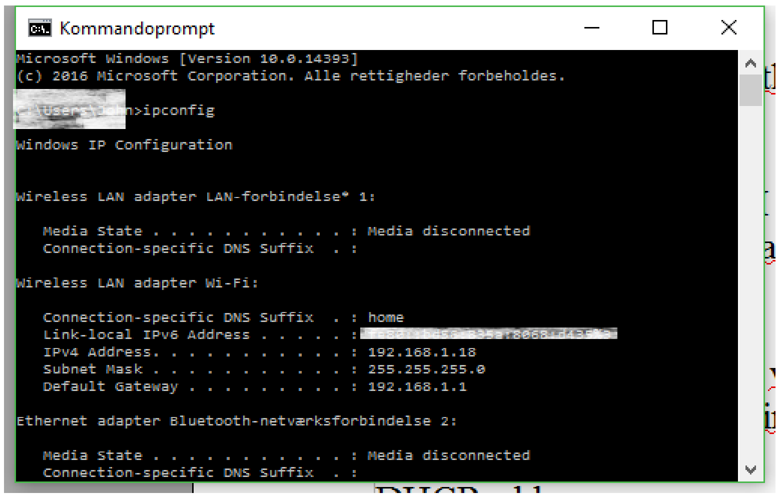
So, bear with me if something is not best practice – and do come forward with suggestions for improvements.

The code is the simplest I can think of; two icons move around on the screen. They do nothing and do not interact. But they are controlled from two different computers – and the game play is up to you.

Nevertheless, this tutorial will be fairly large. We need to cover DHCP addresses and IP addresses, the socket module and finally the interactive game.

DHCP address

DHCP is acronym for Dynamic Host Configuration Protocol. Your DHCP address is the unique number used in communication to your Router, something like ”192.168.0.18” or “10.0.0.8”. To find the DHCP address, open a command prompt and type 'ipconfig', you'll get an output like:



Kommandoprompt

```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. Alle rettigheder forbeholdes.

[redacted]n>ipconfig

Windows IP Configuration


Wireless LAN adapter LAN-forbindelse* 1:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . : home
   Link-local IPv6 Address . . . . . : [redacted]
   IPv4 Address. . . . . . . . . . . : 192.168.1.18
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.1

Ethernet adapter Bluetooth-netværksforbindelse 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
```

The address is here called Ipv4 Address.

If you plan to connect to another computer on the same Router e.g. A family member in the same house, you should also look for the DHCP address of this computer. More to read:
http://whatismyipaddress.com/dhcp

IP addresses

If you plan to connect to a computer on another network, you need to know the two Router IP addresses involved. This is the IP address of your Router when seen from outside the local network. The easiest way to get the IP address is by using one of the many free services on the Internet, such as http://whatismyipaddress.com/

NOTE:
If your Router has dynamic IP-address, the value may change.
You will need to check the IP address periodically, especially if the Router has been stopped.

Port forwarding
If you plan to connect two networks, you'll need to set-up the **port forwarding** in both Routers.
This can be a bit of a challenge, and depends of your Router. Please look-up your Router manual for details.
A signal arriving to the Router will be blocked by your firewall unless port forwarding is established. With port forwarding, the Router knows which DHCP address that listen to each port; thus, directing the signal to the correct device.

Ports
You will also need a port number for the communication. Ports are address inside the computer used when communicating. This is a whole major subject in its own right; what you need to know is that ports used here should always be in the range 1025-65535. I tend to use 9990-9999.

Communication
To communicate between two computers, we use the Python socket module.

We instantiate a small Server and let it 'listen' for data traffic on the specified port. Any data on the port will be translated to meaningful instructions using a small 'protocol' – as you will see, the protocol used here is extremely simple.

All the Server does is listen and provide data to our program.

Whenever we want to send data; we first construct the message using the protocol and we then enter a 'client cycle':

1. initialize a Client
2. send the message
3. close the Client

So, in this program clients are only for sending, and clients only 'live' to send one single message.

Code structure
The code I'll show consist of two programs. The first one is a module used by the second one.
You need to download the two programs in the same folder; the first code MUST be named 'connection.py'. I'll refer to the second program as 'game'.
The two programs must be placed on two computers. The DHCP/IP addresses of the two computes must be hard coded to line 15 and 17 in both versions of 'game'. Also, make sure that the ports are mirrored in line 16 and 18 (or use same port on both computers).

connection.py

// CODE //

The code here I've collected and assembled by a lot of reading and testing; I'm not sure of all details, so bear with me for not explaining it in detail.
On a high-level the code describes two class objects, Client and Server with their 'send' and 'receive'

methods respectively. Furthermore a small function 'def send():' executing the client cycle described above.
The two print statements in line 13 and 19 are remnants from the code development. In a real game the timeouts in communication should be handled more gracefully; this is still work in progress.

The code utilizes the socket module.
Link to the socket documentation: https://docs.python.org/3/library/socket.html
Link to socket tutorial: https://docs.python.org/3/howto/sockets.html

game

// CODE //

The idea in this code is that both players use **exactly same** code (apart from the IP and port addresses). The two player objects needed are called 'me' and 'enemy'. The code ensures that the 'me' player on one computer is perceived as the 'enemy' player on the other, and vise versa.

In line 1-2 imports are made; the module 'socket' is from the standard library, it provides all the server and client code. The module 'connection' is the code provided in this tutorial, above.

Line 15-18 is where you need to insert the IP and port addresses relevant for you. I could have wrapped this in a nice input page, but I want to keep this tutorial as simple as possible.

When you are connected to a LAN, it is possible to get the local IP address using:

// MY_SERVER_HOST = socket.gethostbyname(socket.gethostname()) //

But as you still need to enter OTHER_HOST and the port numbers, I have kept things as transparent as possible.

The class Player() is a general player object. It has three small methods; move(), draw() and make_data_package().

The make_data_package() executes the 'protocol' for data transfer.
We want to send the x- and y- position of the player; the two values are converted to string of length 4, padded with zeros as necessary. Examples: 2 → '0002'; 123 → '0123'.
Finally, the two strings are concatenated to a structure like 'xxxxyyyy', which it the data package to send.
If we jump to line 102-105, this is where data are received and unpacked; you see how we convert the string back to two integers, updating data for the other player.

The two player objects, Player_1 and Player_2, inherits from Player. The two instances only differ in starting position and color.

The small ip_value() function converts an IP address to an integer; it does so by padding each octet (three digit group) with zeros and removing the dots. Example: '192.168.0.18' → 192168000018.

As the two DHCP/IP addresses will always be different, the comparison in line 73 will always turn out 'True' on one computer and 'False' on the other. We don't care which is which, but we use this to ensure that 'me' on one computer is Player_1 and 'me' on the other is Player_2.
In this small code, it is of less importance, it only ensures two different start positions, but in more complicated games you may need to define different roles like 'master' and 'slave'.

The event_handling() function is standard, just notice the server.shutdown() included in the termination of the program.

The get_input() is also simple. Could also be made with a dictionary, but I chose a small list of tuples.

In the data_transfer() function the data package is created and send to the other computer.
Then we listen for received data which is then converted as explained above.

The update_screen() needs no comments.

Finally, in line 116-117, the two Player instances and the Server instance are created.
The small game loop in line 119-123 is simple.

Comments
I have tried to keep everything as simple as possible, only focusing on code related to the data transfer and data handling.

The 'game' is not a game, but rather a framework on which a game can be build. I plan to issue a tutorial demonstrating a full interactive game building on the code shown here; but you don't need to wait for it – you can probably make something yourself now.

Play around and feel free to use this in any way.
If you find improvements, I'll be happy to learn about it as a respond to this tutorial.