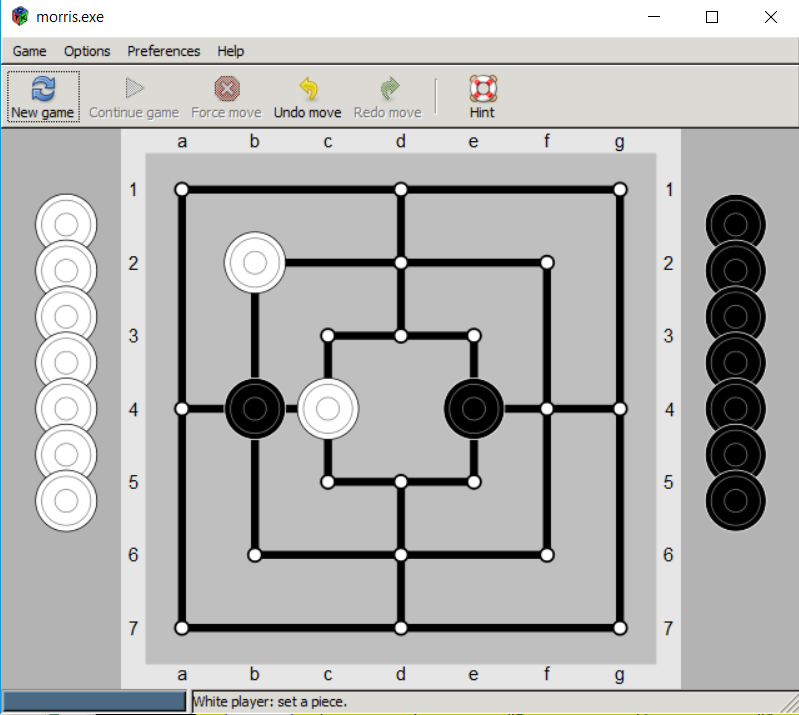
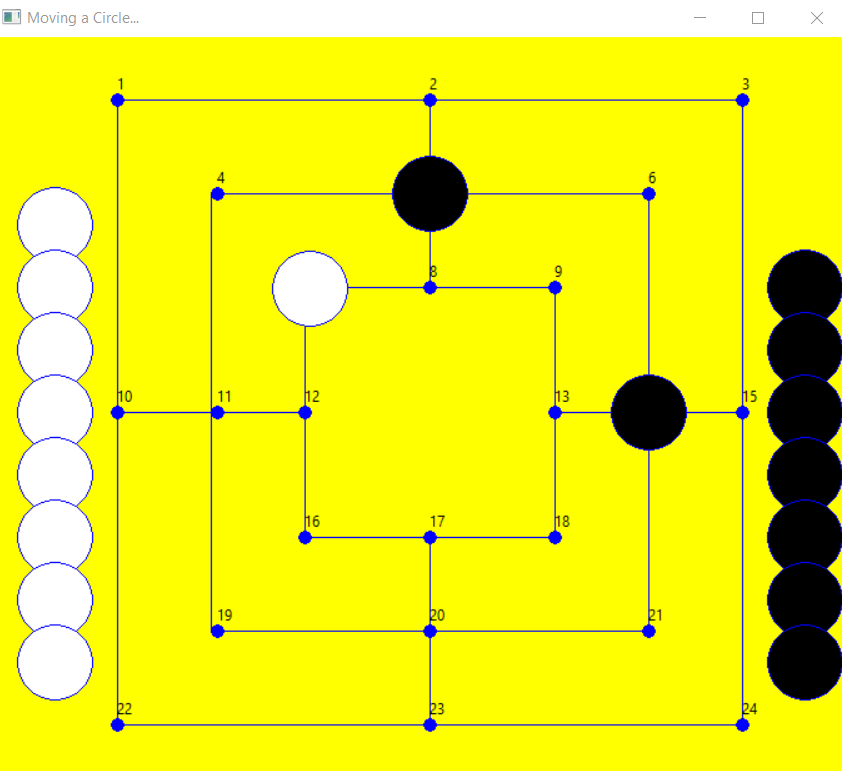
**General**

This document describes the development of the game 9 Men Morris .

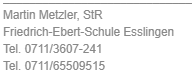


The game is part of a project between two high schools in Germany and in Israel.

The game will be played by robots one in each country.

At the begining the teachers begin to develop the concepts, learn the material and then the students will continue.

Martin Metzler ( [metzler.martin@gmail.com](mailto:metzler.martin@gmail.com) ) who teach in [Friedrich-Ebert-Schule Esslingen](https://www.fes-es.de/)  and me Nir Selickter ([nirselickter@gmail.com](mailto:nirselickter@gmail.com)), I teach at [Tichon Hadarim](http://hadarimschool.co.il/) work on the software in both side. Martin work with few technologies like Node-Red, Dashboard , MQTT and more.



I learn the MQTT as the network platform and choose to work with python.

This document will describe what I did in python. I think as a developer and as a teacher that it is very important to document work. It is good for you and it can help other to learn

I will describe step by step what I did .

Currently I did not finished the work, so I will continue to develop the game and to document my work.

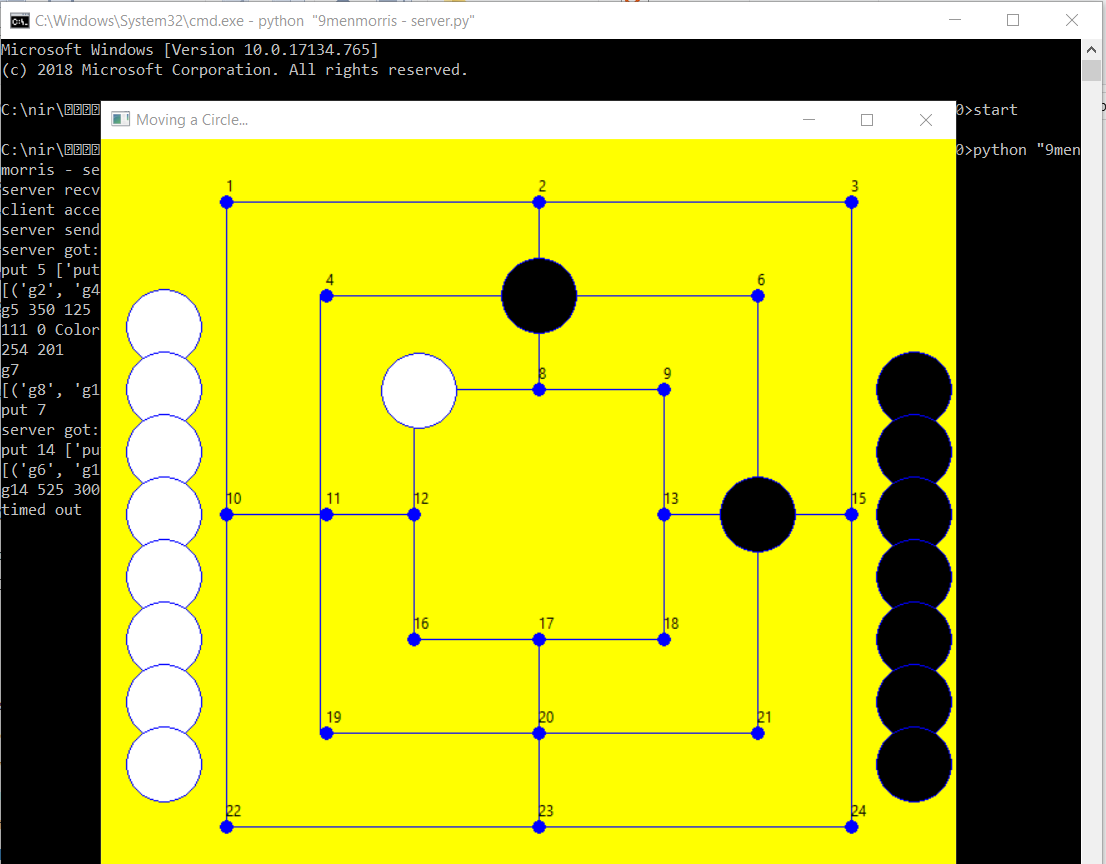
The goal is to have a version with AI , running on Raspberry PI with MQTT.

The status now(20/may/2019) is that it is possible to run 9mm in two sides, server and client.

You can put/move the 9 coins on any of 24 stations on board and the other side will reflect it on his board . The server now is with white and the client is black. The client server are connect by tcp sockets on port 8820. I did the tests in one pc, means I work with loopback connection 127.0.0.1 (but it works on two pc )

The next two pictures show you that the user at server side and user at client side see the same picture – same board

Here is how the server side look after few moves



Here is how the client side looks like after same moves



**Terms**

**Coins** – the small circle (white/black) that you may move to play. There are 9 white coins and 9 black coins.

**Station** – the place where you parking the coin. There are 24 stations. When you drag&drop the coin, it check if it was park exactly on a station

The stations are "g1","g2" ,"g3" up tp "g24".

"g100" is out of board

**Board** – the layout of the game, the yellow layout in the figure above. Include station, coins

**Messages between client to server**

1. **Put** – one side take a coin from one station and put it into other station

Example

put 4 g100 g3 – take coin 4 from station 100 (out of the board – game in state of beginning) and put it on station 3

1. **take** – take one coin from the board. This happen if one side has a mill and it can take a coin of his rival

take 4 g10 - take coin 4 from g3 (and put it in station g100 – out of board)

1. **your turn** – now you can play and I wait until you finish. When you play you can move coins from one station to other station
2. **mill** – I got a mill and I now think which of your coin I take out of the board

a 'mill' message is display in chat panel

1. **checkDb** – the remote side send the graph db

The graph db the information we know about each station – if it is occupy , by whom etc – when we got this information we compare it to our graph db. If it different it is major error and we need to debug the game. (there is option to print the graph db in gui)

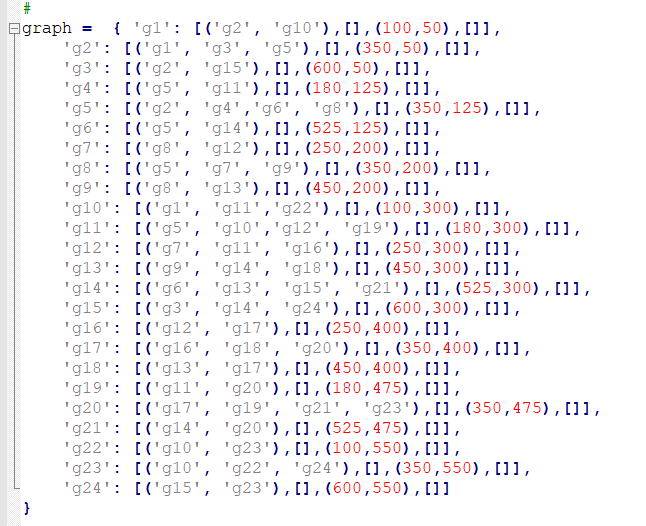
**Graph DB**

This is the DB about each station , its x, y, it's neighbors, if it is occupy , by whom.

Both sides holds the same information.

After each turn – one side send it to other side for comparing. Look at example 21

If it is different, there is major bug that shall be debug. –look at example 25



**Example 1 – client server (20 May 2019 version)**

Example 1 is the code to run the two side game as describe at the beginning of the document. It is the same like example 12. Now ( 29 may 2019, there is version 27. It is much better)

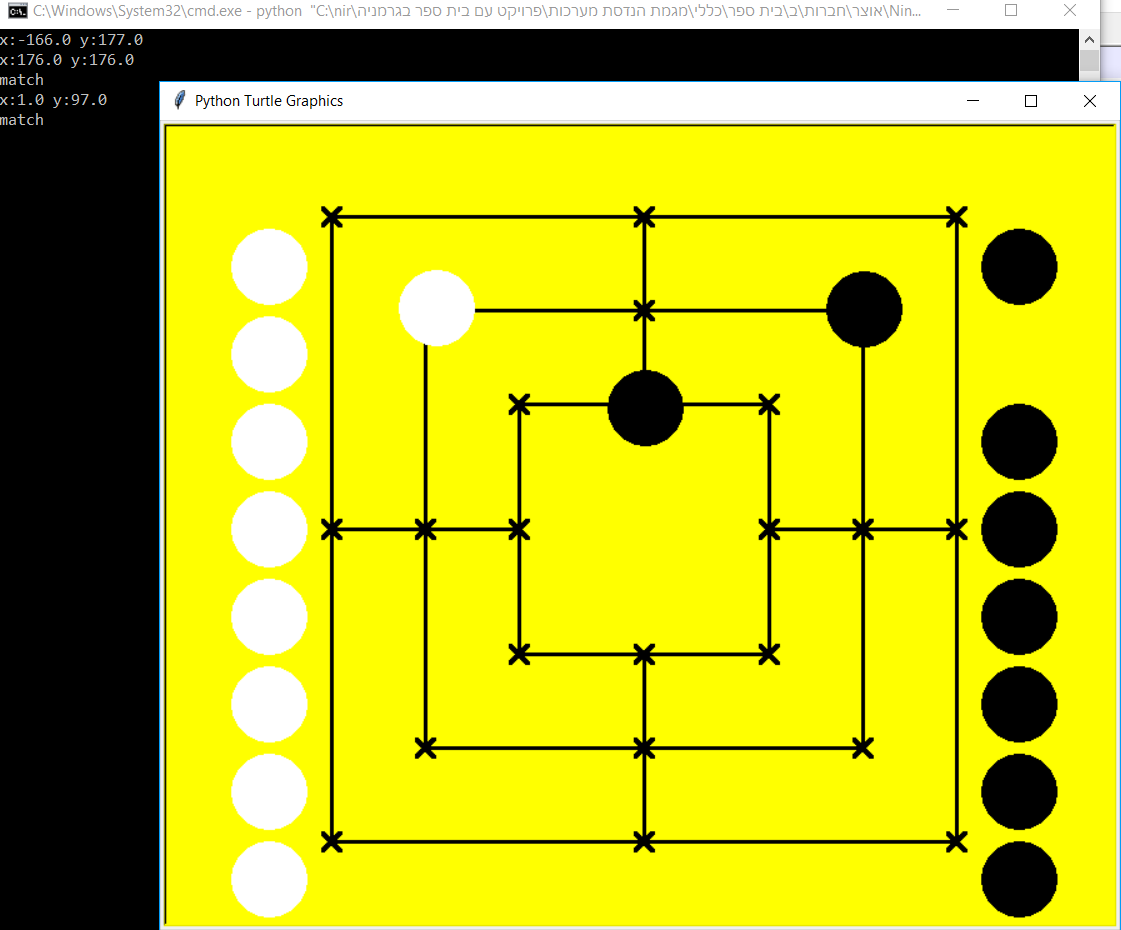
To run this you shall install python version 3 for Windows . The last version now is 3.7.3

You should also need to install WXPYTHON ( I install 4.0.3)

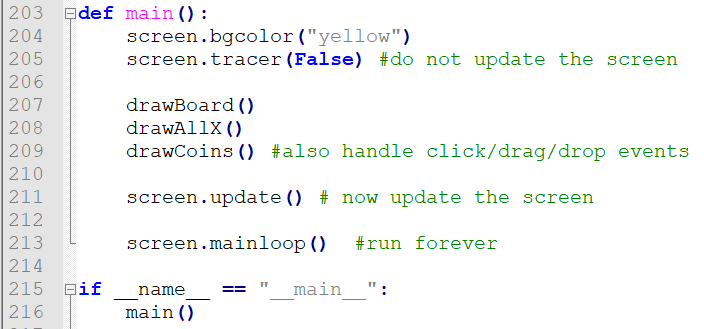
**Example 2 – turtle with drag&drop (12 April 2019 version)**

Library – import turtle

I begin to build the game in TURTLE (python module for basic graphics), I create there the basic concepts that still exist – how the board looks like and how the coins are located ¸and the concept of drag and drop . and the concept of match/hit if the coin is placed exactly on one of 24 stations



This is the main function



This is the code of the coins, it draw it at init and then it handle the drag/drop event



This code look nice but I try to add thread to this code and did not succeed. Maybe there is some way that I do not know.

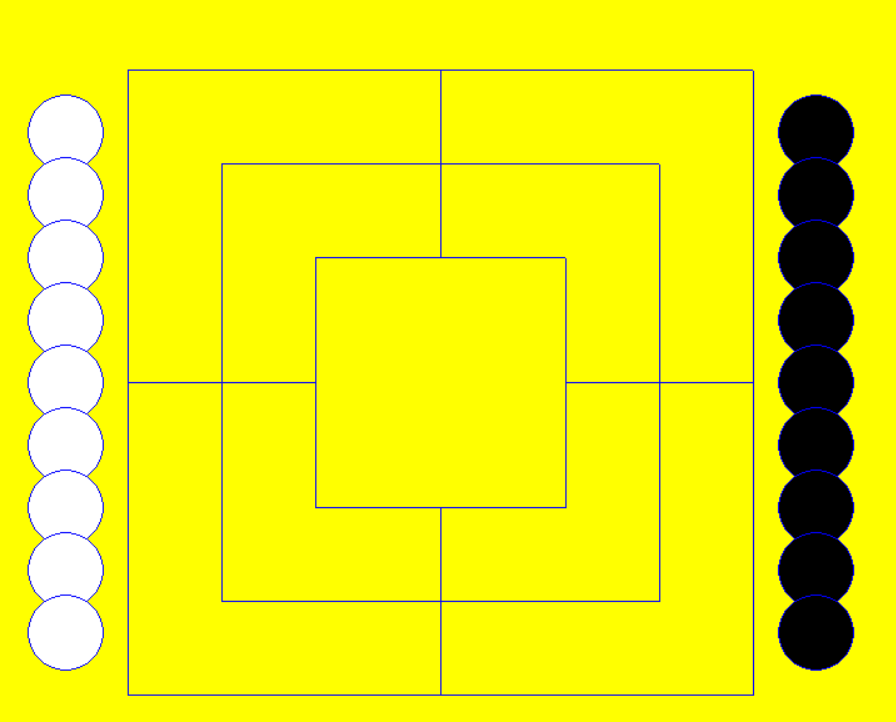
So I change the GUI from TURTLE to WXPYTHON. I know some WXPYTHON and it is Good . There are more GUI for python like QT, GTK ,KIVI and more, but for each gui you shall learn how to use it . I have some experience with WXPYTHON. Another issue is that there are designers for some gui tools like wxglade and more, but I do not know them either

**Exampl3 – wxpython –draw**

Library – import wx

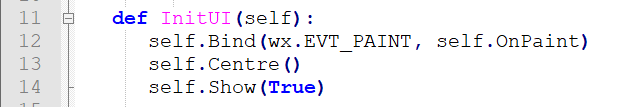
In this example I just draw the board and coins. Did not draw the 24 stations.

This code does not handle drag and drop



In wxpython , you can connect some function to drawing event got from windows.

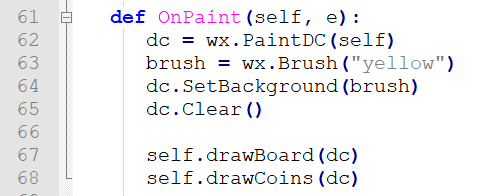
The basic event is EVT\_PAINT ( it occurs when the board is resize and in other occasions)



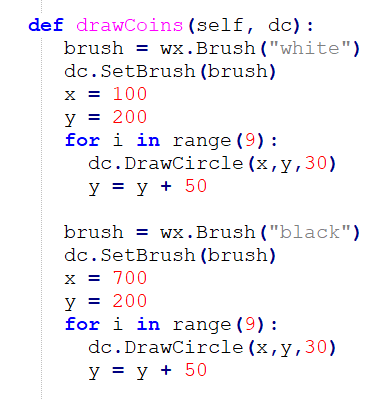
And here is the function that handle the event.

***Each time that this event is create you shall draw all the pictures items again !!!***

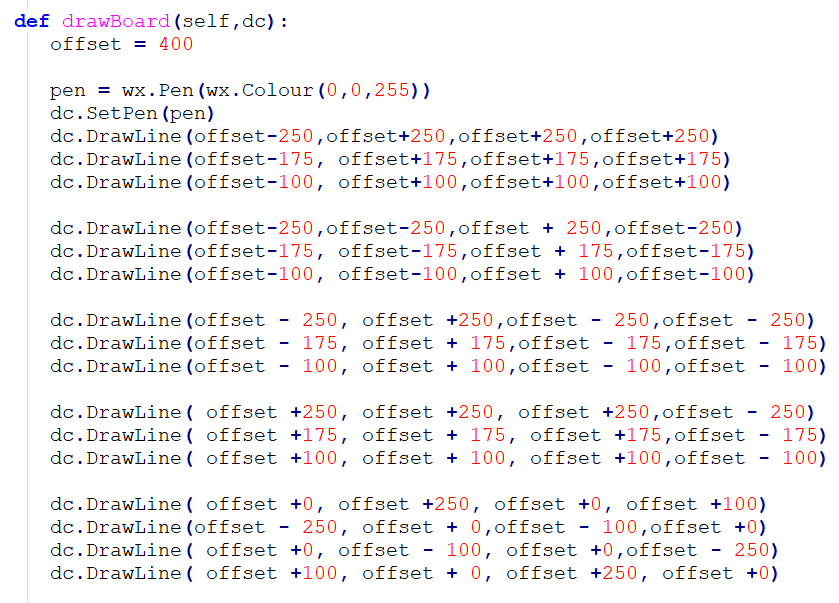
This is important information. If you do not draw all items then it will not appear !



Draw the coins



Draw the board



There is a new (may, 2019) good book about wxpython from Michael Driscoll who got [a blog about](http://www.blog.pythonlibrary.org/) python ,wxpython and other stuff, and he also answer question in [stackoverflow](https://stackoverflow.com/users/393194/mike-driscoll)

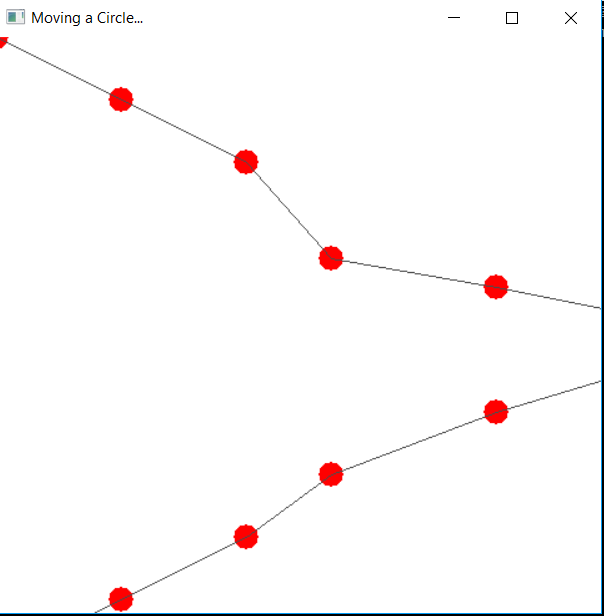


**Example 4 – drag and drop**

Library – import wx

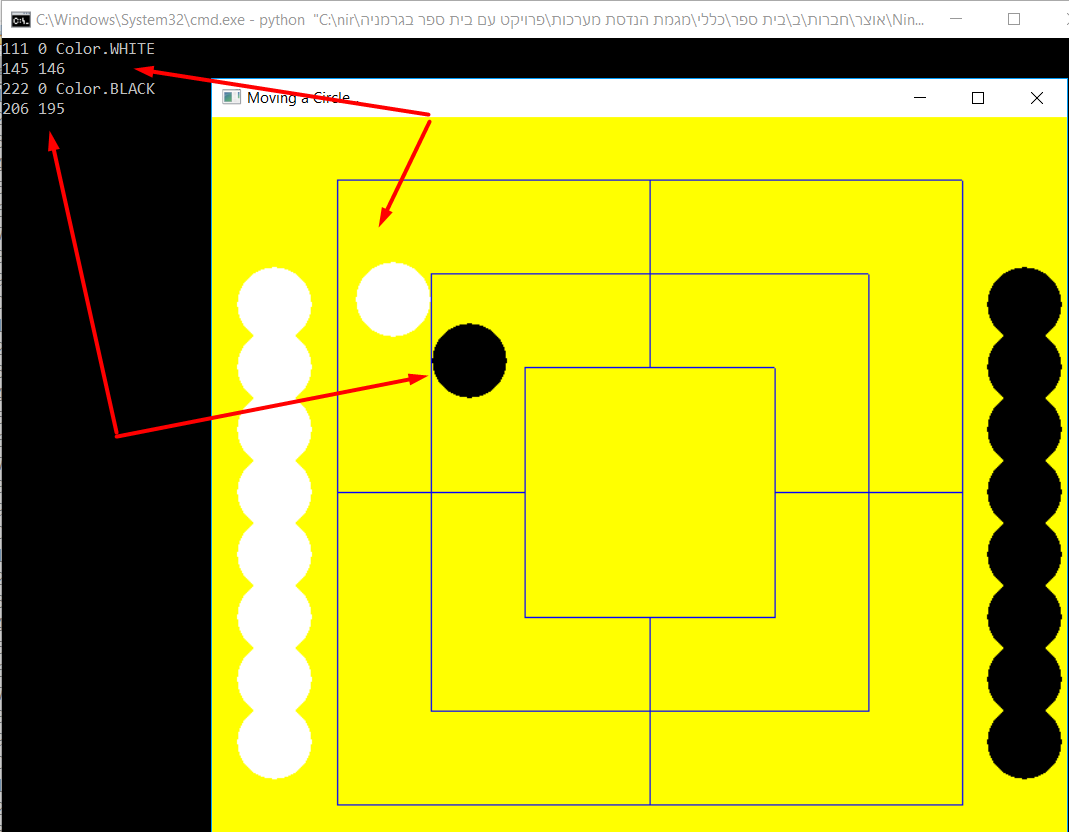
After drawing the board, I want to add drag&drop to the coins. I look a lot at internet and in books for small code that shows how to do it.

Finally I [found this](http://wxpython-users.1045709.n5.nabble.com/Drag-and-drop-circles-td5714000.html) code. It draws circle and let the user to drag and drop it

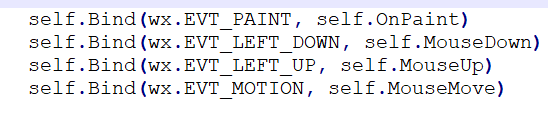


I took the ideas from that code and adopt it.

When you drop the coin you can see the x,y



The code handle 4 events



The MyPanel class is responsible for drawing. It also hold more information about the coin

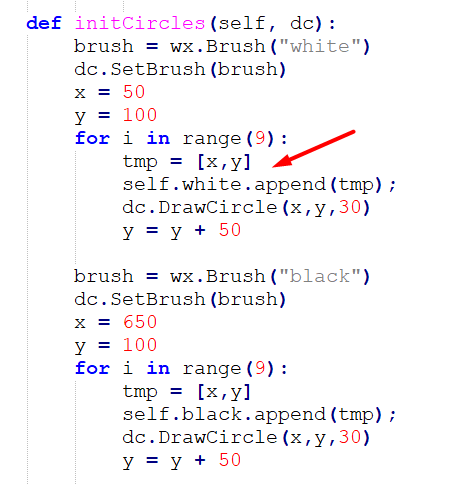
From this example and on , we do not just draw the coin(circle) , we build a lists , coins list, members 0 to 8 , we know each coin position. When we move coin[4] to x=450 y=300, we know it. We keep it becasuse in the evt\_paint we shall draw each coin in the right place

There are two lists of coins



When you draw the circles you add to list the position of each circle. This is very important since we need to redraw in each EVT\_PAINT the whole pictures and we need to know where are the circles was before the paint event.

In the following code the circles are print at the side of the board, at initial position

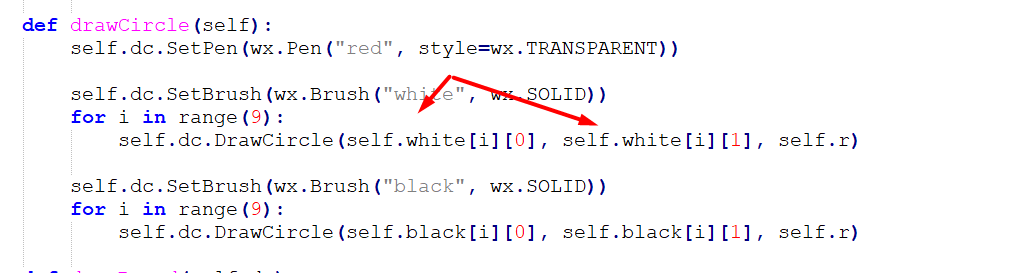


In the next function – drawCircle, the circle are drawn at the exact position that they was before the evt\_paint arose

To draw a simple circle you just call this function



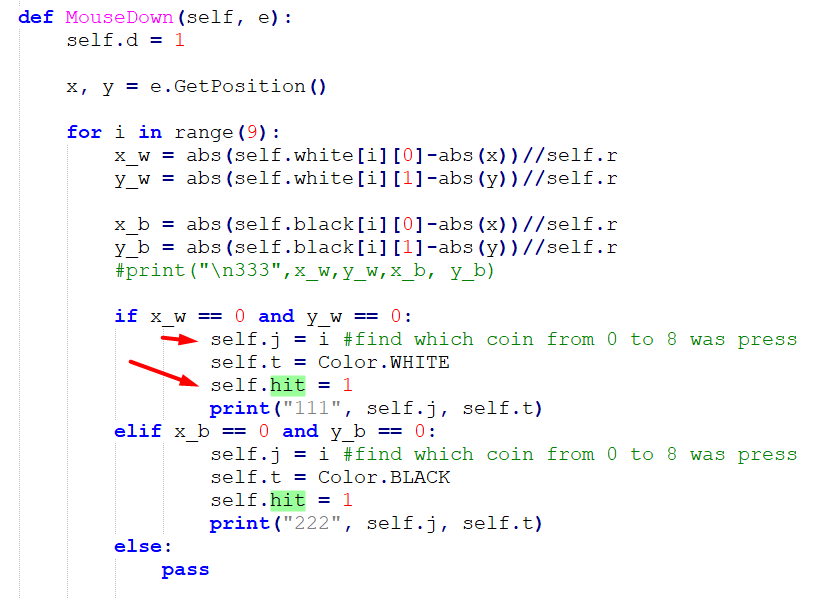
But to draw the circle at the place it was, you need to take the position from the self.white / self.black list



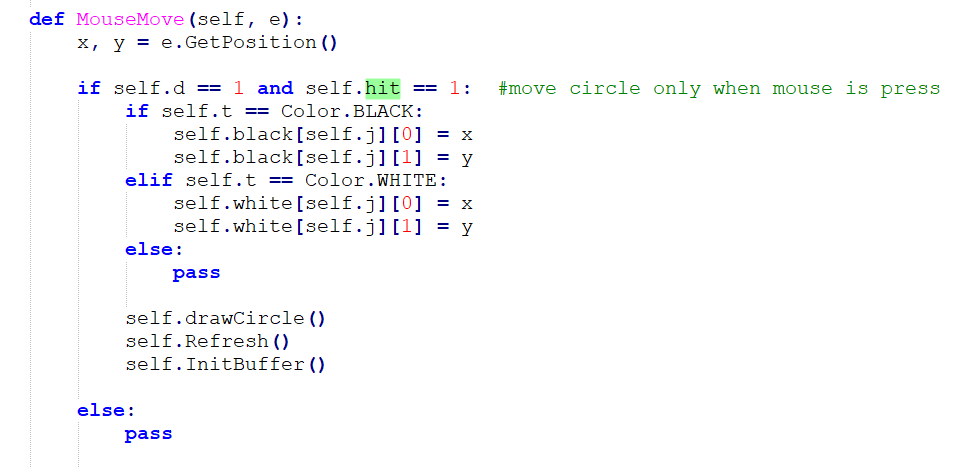
How we know on which coin we press ?, we need to take the x,y coordinate of the pressed mouse and check if it hits one of the coins

In the variable self.t we save which color was clicked and in self.j the number from 0 to 8 of the coin.

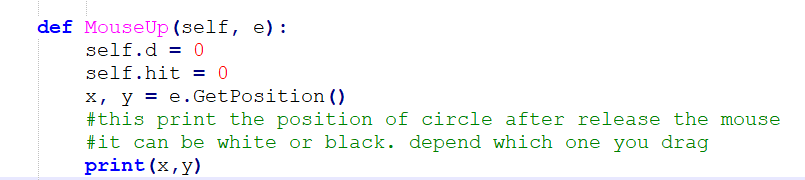
We also set self.d = 1 that we begin to move the coin



Now the mouse is down and the user move the mouse to new position, we need to accompanies the mouse movement with the circle. So we draw the circle while we get mouse movement events.



When the user release the mouse we just print where it was release



**Example 5 – move the coin to (x,y)**

import wx

from threading import Thread

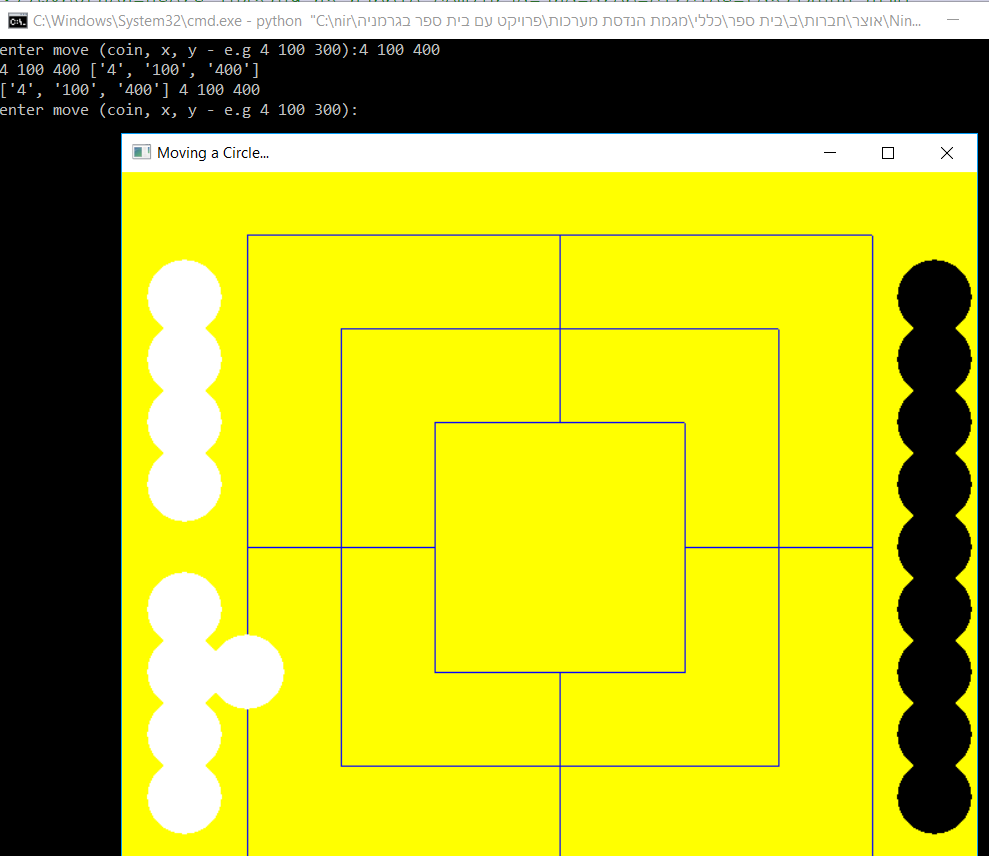
from wx.lib.pubsub import pub

The next thing to do is to make the coin move according to the user command ( and not just because of drag & drop)

It means that now you have two option to move the coin 1) drag it 2)move it by command

I choose a simple way to do commands – from the console.

For example I pressed the command 4 100 400. It moves white coin number 4 ( there are from 0 to 8) to x=100 y=400

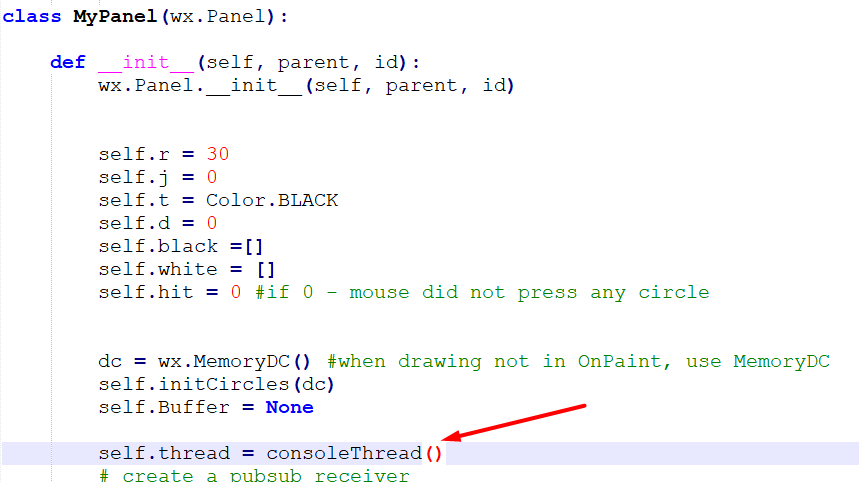


The main advance here is that the console is a spate thread !

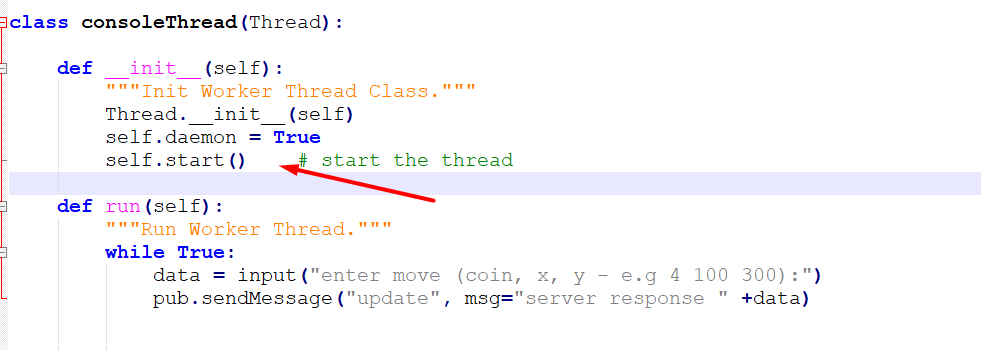
There is a thread that wait for user command and in case the user press something, this console thread send by a pub/sub mechanism message to WXPYTHON GUI.

The consoleThread is a class that inherit from Thread. In MyPanel.init we instantiate the consoleThread

And in the constructor of consoleThread it begins to run



And here in consoleThread constructor the thread begins to run



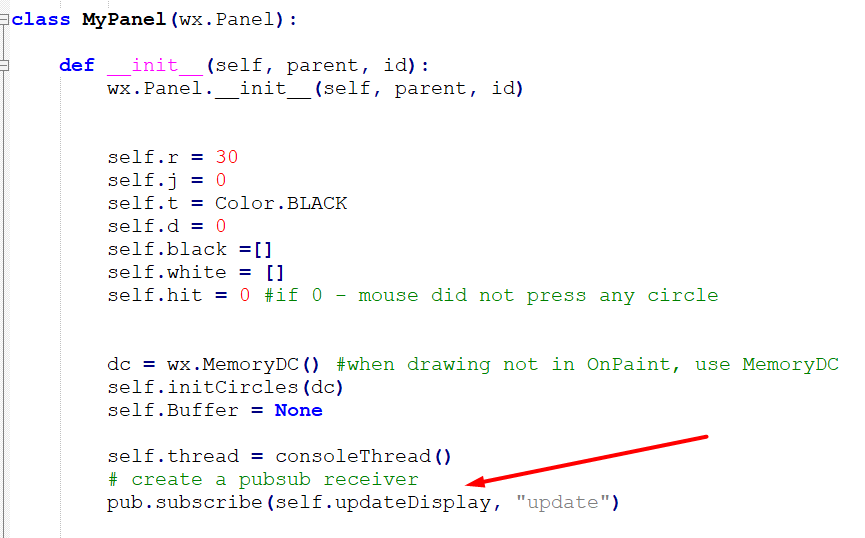
The interesting line is



This line publish the data get from user in the console, to the wx thread

Where the wx get the data?

First it need to subscribe to publication from console. It set the updateDisplay routine as the function that handles messages from console

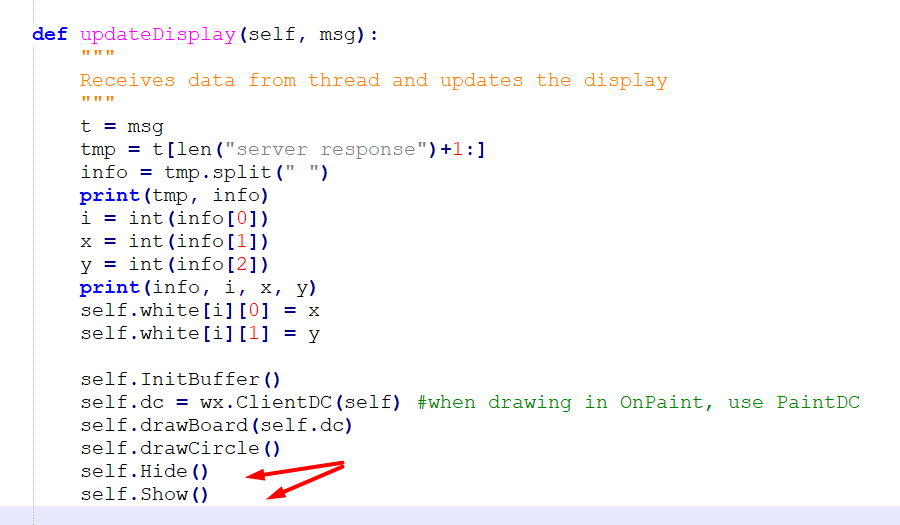


And here is the function that handle the message from consoleThread.

First it is need to parse the message, then we draw into some dc context ( this come from windows GDI)

All the picture item and then we need to refresh the screen.

We can not create EVT\_PAINT so we hide and show the window and this create the paint event that take the information from dc context and place it to the windwo



**Example 6 – graph db , move coin to station**

import wx

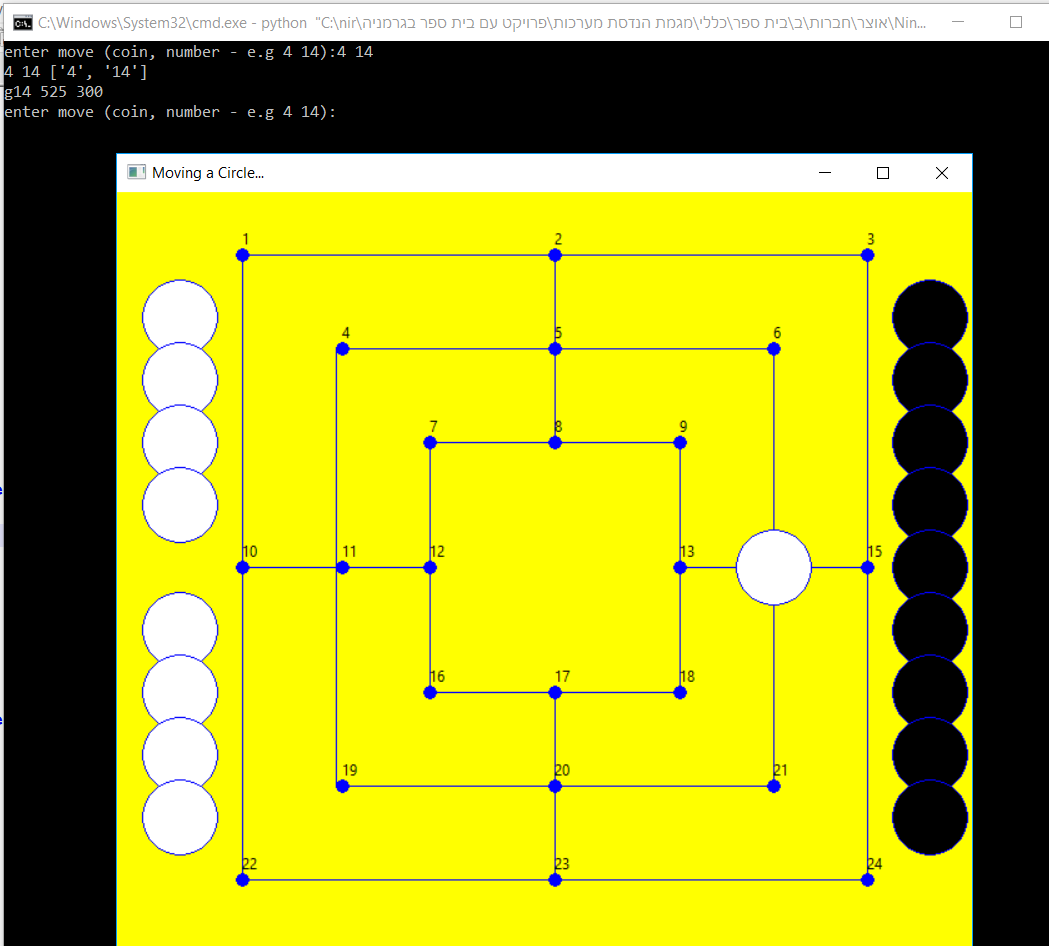
from threading import Thread

from wx.lib.pubsub import pub

import Enum

In this version I create a new file graph.py and it hold the information about the game, name of station coordinates and more.

Now in the console you can do the command 4 14 – it means coin 4 go to station 14

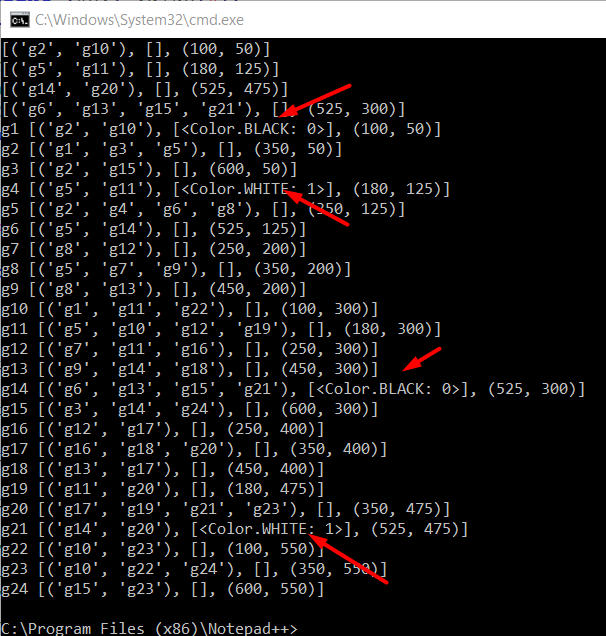


Here is the graph database

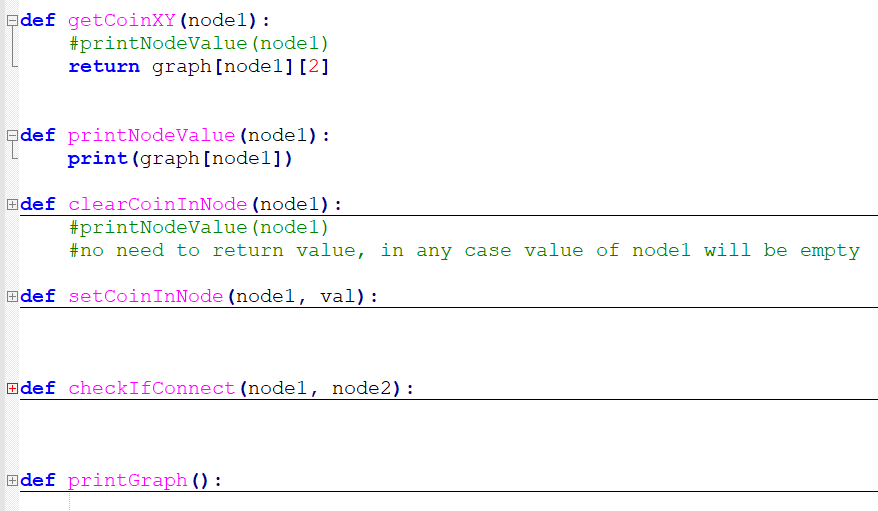
The empty square brackets [] are place holder. If it is empty ,then no coin is there



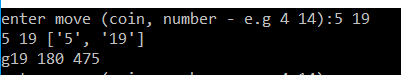
And if some coins move the [] will fill with the color of the coin



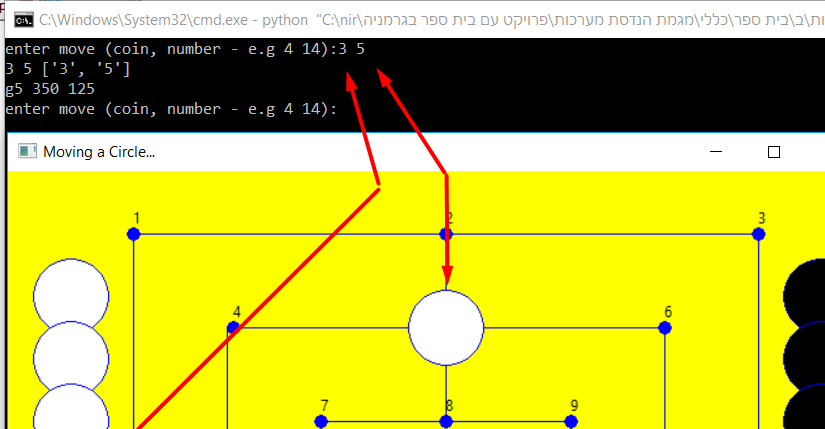
In the graph file there are some utility method to put coin in graph db, to remove from graph db etc



So if the user want to move coin 5 to station 19 he will write and a white coin will move to x=180 y=475

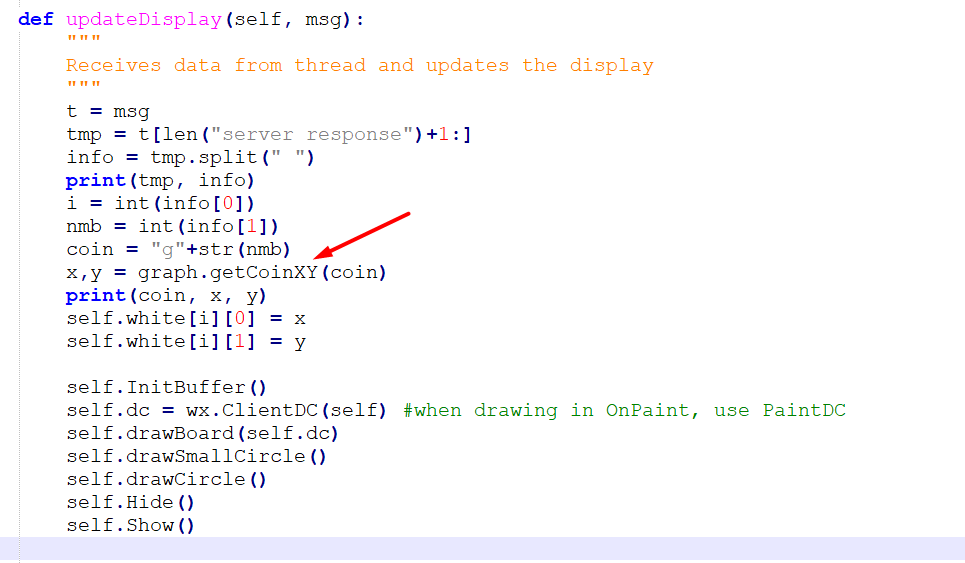


Or this



Here is the code that accept the message from console, get from graph db the coordinates of the

Needed station.



**Example 7 – hit on station**

import wx

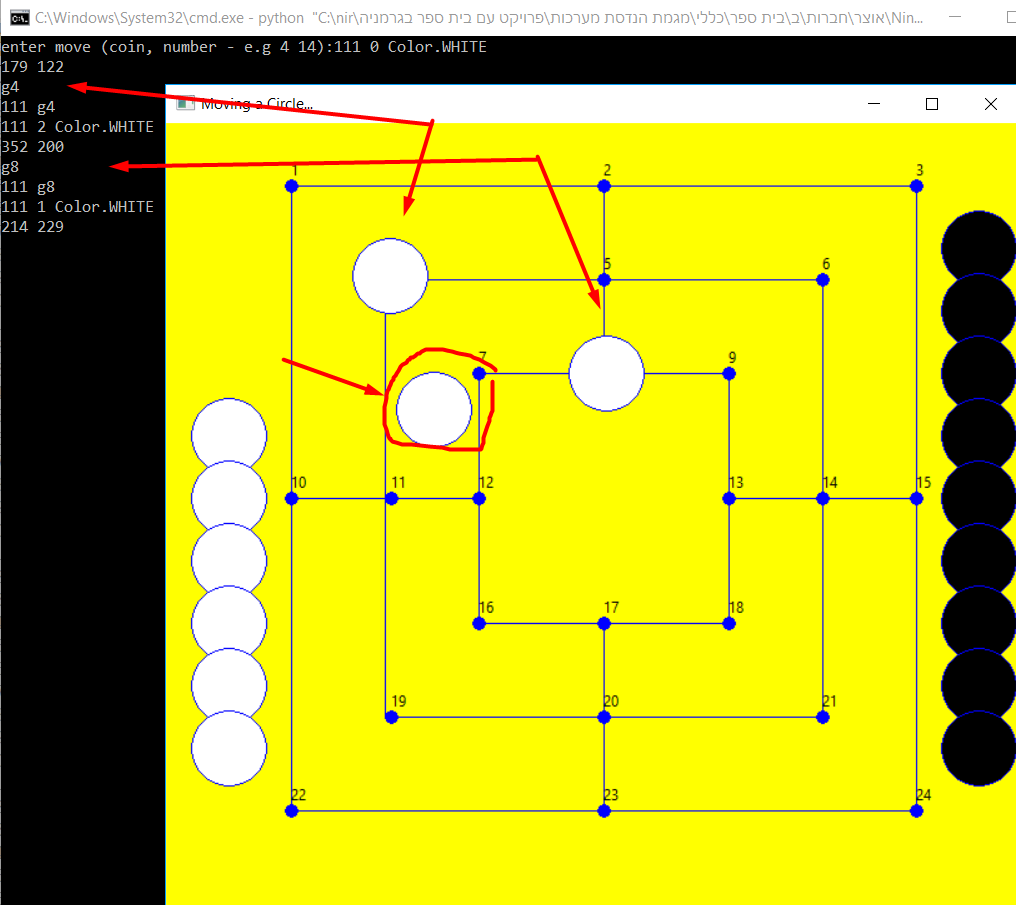
from threading import Thread

from wx.lib.pubsub import pub

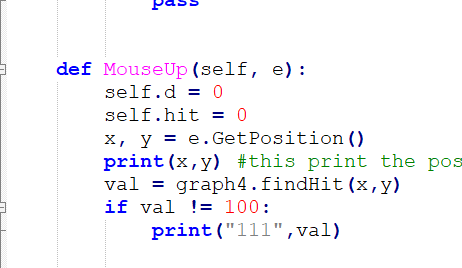
import Enum

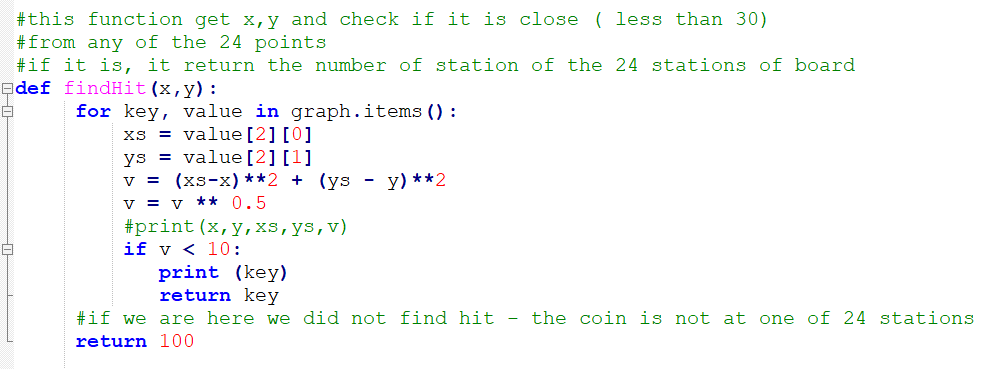
In this example there is test if when the user dragged and dropped a coin, it is exactly on one of 24 stations

Here you see that two coins are set on exact stations 4 and 8, but the third coin is not on any station, so there is no indication about it



Here we show how we check if there is a hit





**Example 8 – client console - server**

import threading

import socket

import sys

import wx

from threading import Thread

from wx.lib.pubsub import pub

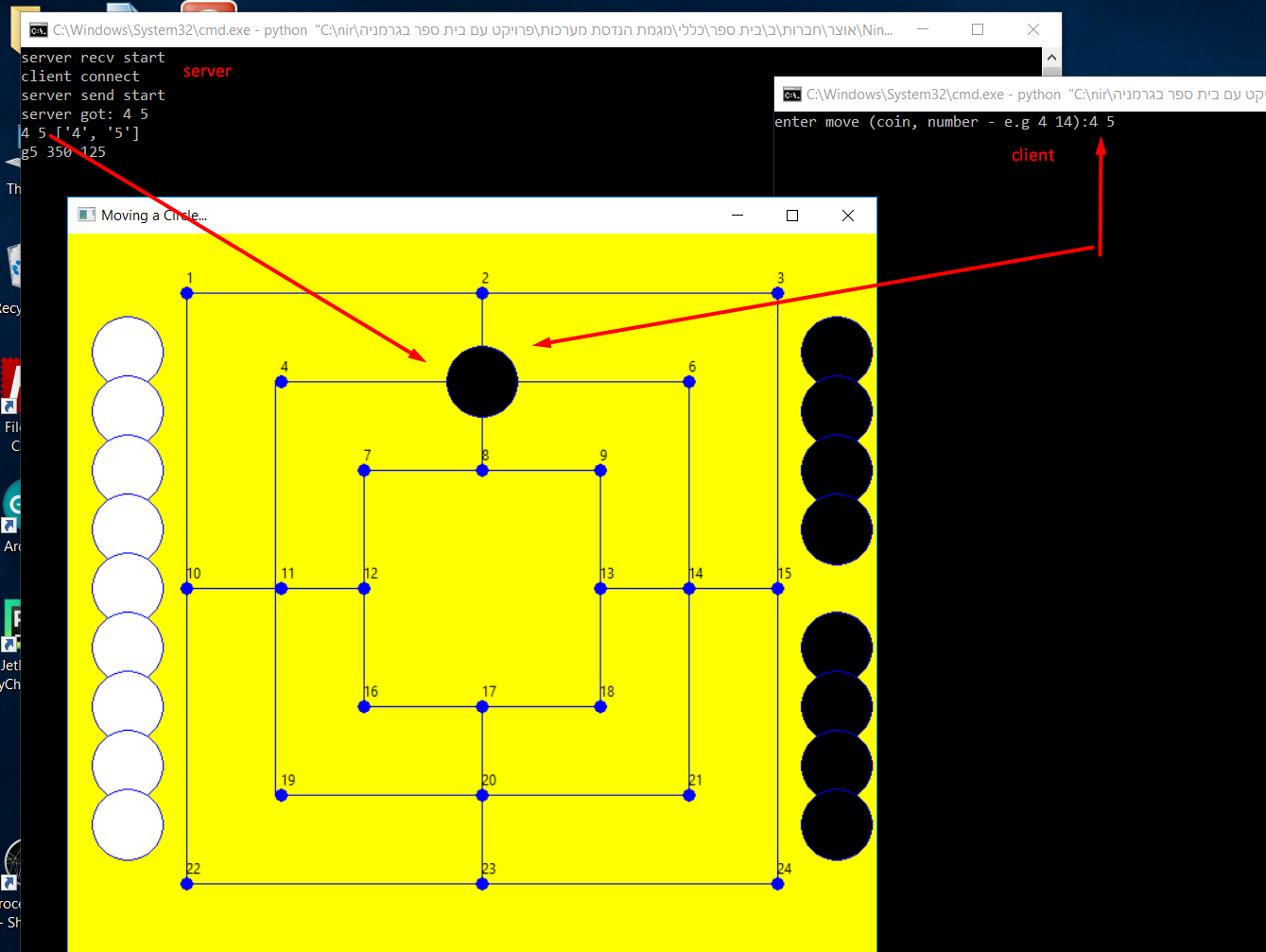
from time import sleep

import Queue

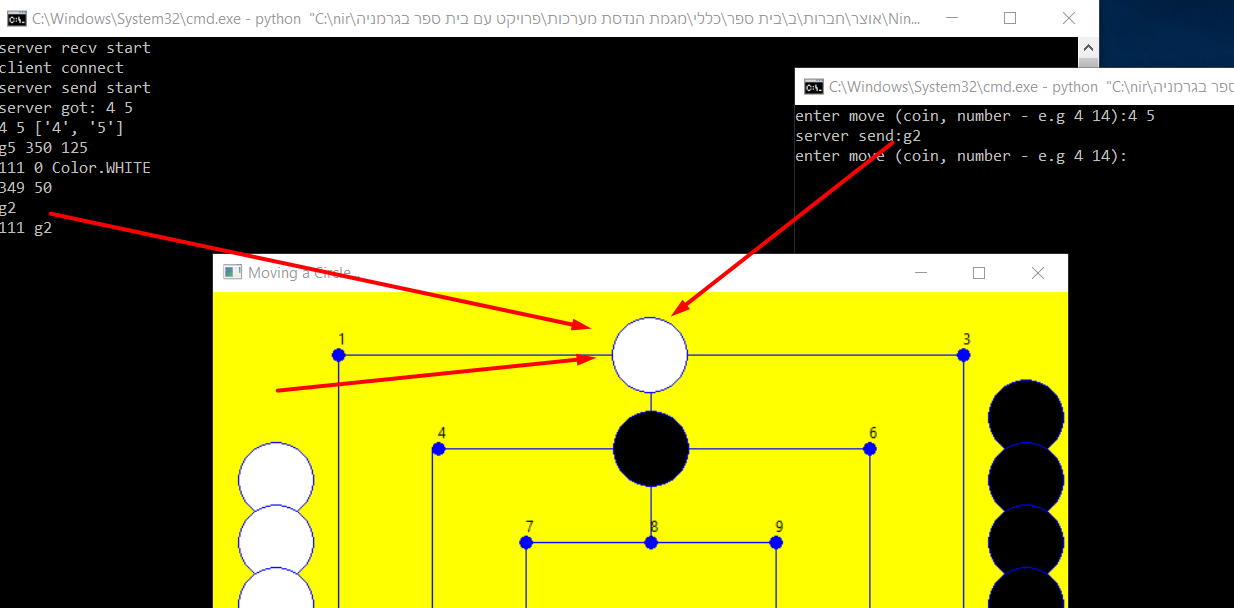
from enum import Enum

In this example I add socket communication.

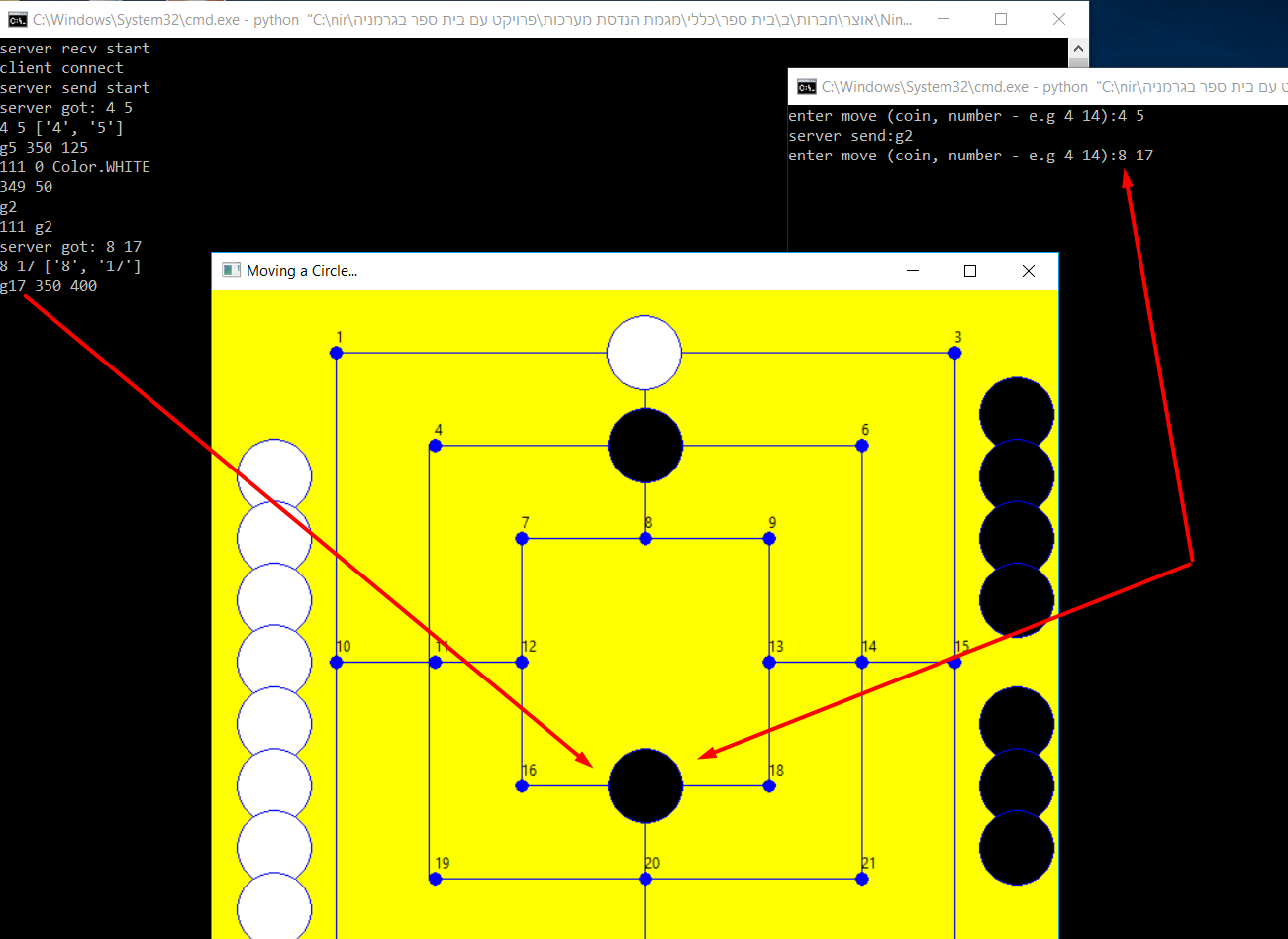
In the client side there is console that can accept commands, send it to server, the server move the client coin (black) to the station. (you can see that black coin number 4 was move to g5)



.Then the use at server side move white coin number 0 to g2 , and send this information to client.



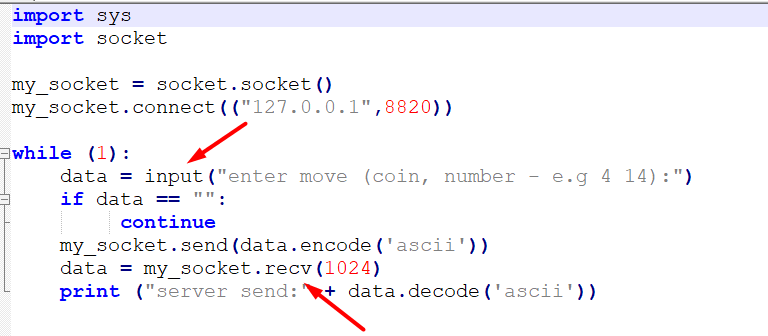
Now the user at client can do the next command. It begin to be a game…



Here is the client code. It wait for a command from user, then it send it to the server and then wait for server response

Pay attention to this – input() and recv() re blocking commands, it means that the software wait on this command until some input arrive .

And what If we want to wait on both together ?



This is the server code.

In the server we use three (3) threads !

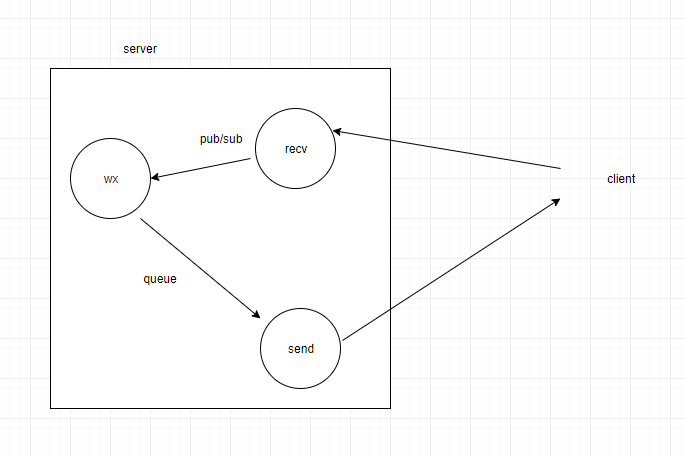
1 for the wxpython gui

2 for the send message to client

3 for recv message to client

You can see the method of communication – between client and server it is socket (in the future it will be mqtt)

Between recv to wx it is pub/sub ( we already see it in previous example) and between wx and send there is message queue



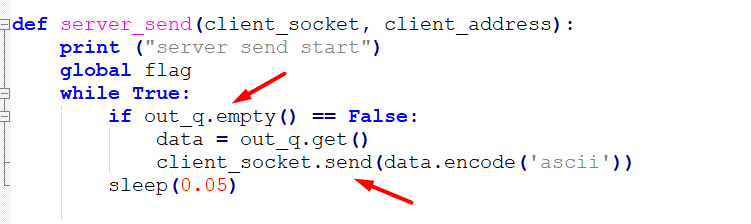
Here is the recv thread

In the recv thread we create the send thread.



Here is the send thread

It wait on the message queue and once message get from wx thread, it send to the client



**Example 9**

import threading

import socket

import sys

import wx

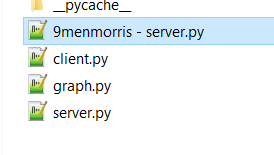
from threading import Thread

from wx.lib.pubsub import pub

from time import sleep

import Queue

This is like example8. The difference is just that I put the send thread of server in a separate file



**Example 10 – put 12**

import threading

import socket

import sys

import wx

from threading import Thread

from wx.lib.pubsub import pub

from time import sleep

import Queue

The new feature in this code is that the command in client was change from 4 14 to put 14

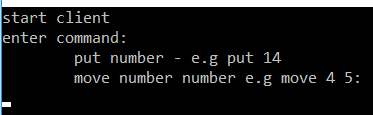
It means take the next coin and put it in station 14.

The client does not know – and should not know – the number of coint. This information is itenal just for repainting

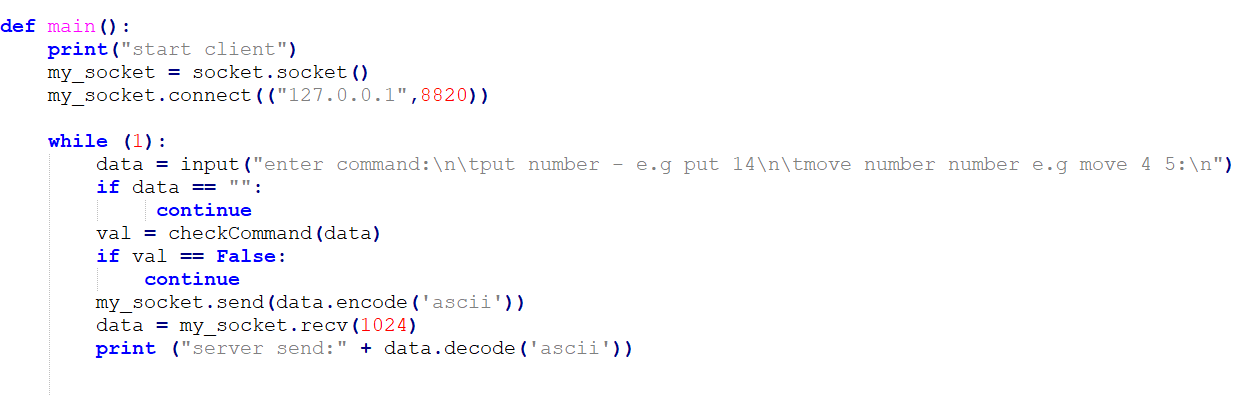
And have no other meaning so you just send the command put some coin on station 14

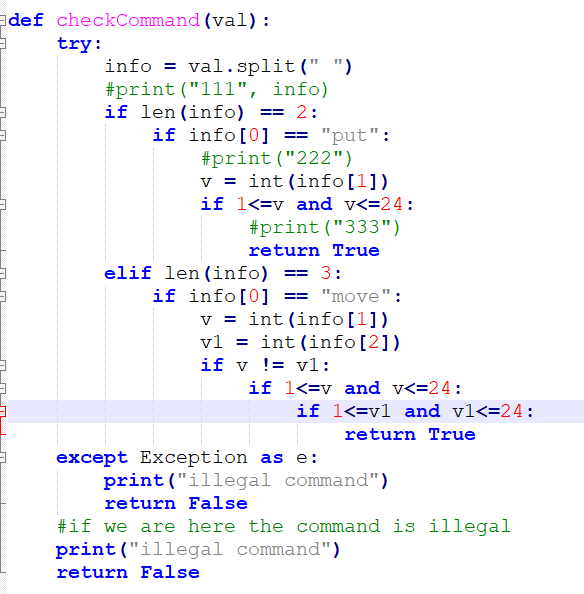
This is the menu in client

Only the put works (the move does not implemented yet)



And this is the client code





In the server , it is needed to make the coins as a pool. It will be at next example

**Example 11 – pool of coins**

import threading

import socket

import sys

import wx

from threading import Thread

from wx.lib.pubsub import pub

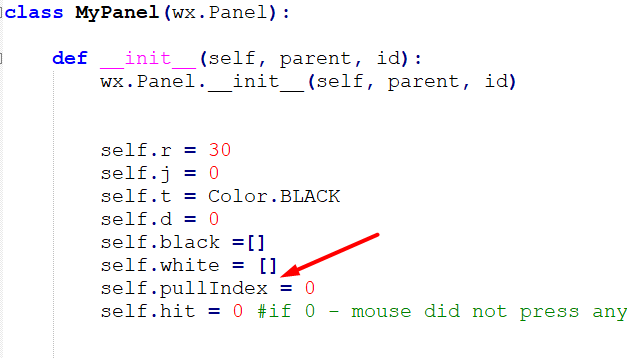
from time import sleep

import Queue

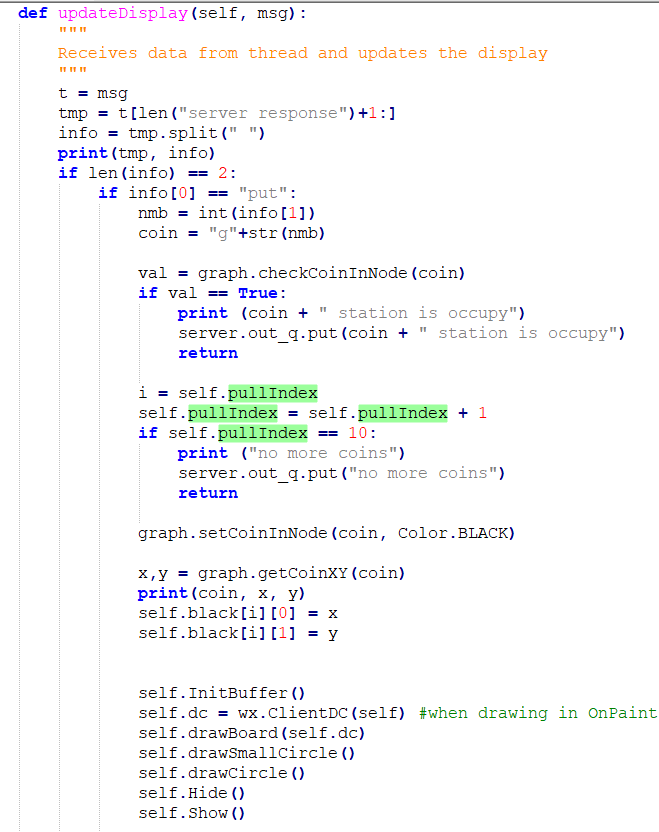
As write in previous example, the server now manage the coins as a pool .

It get command like put 14, so it take the next coin in pull and move it to station 14

Look at pullIndex



And we change the pullIndex when we get put message and we need to update display



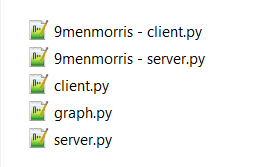
**Example 12 (example 1)– client is graphic too**

Example 12 is the code of example 1.

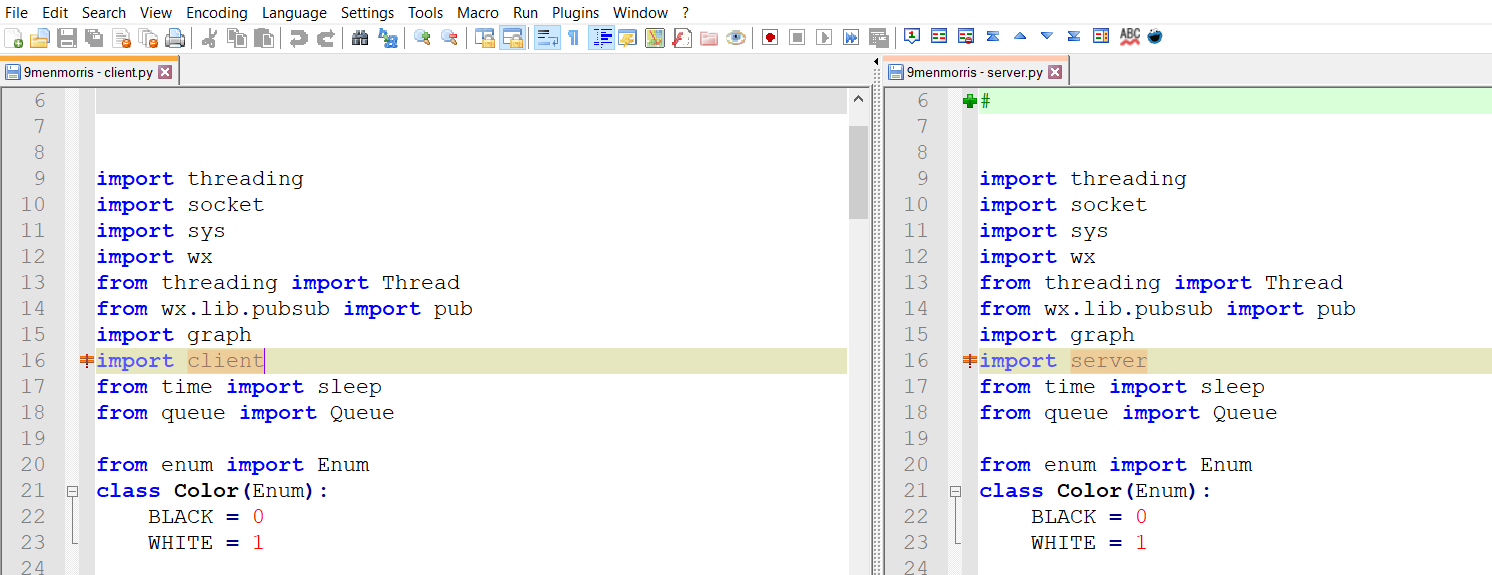
In the last examples, the client was console and the user write the commands there.

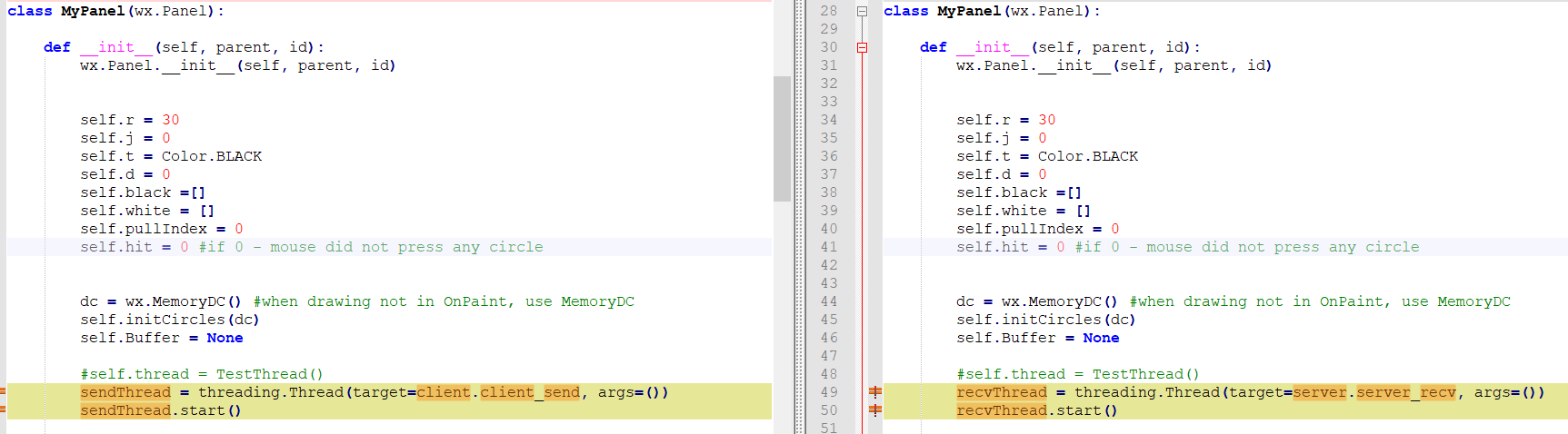
In this example the client become to be graphics too.

I took the code of the server and do small changes (the goal is that the same code will run the client and server – just to change the initial arguments – white/black, client/server, socket/mqtt.)

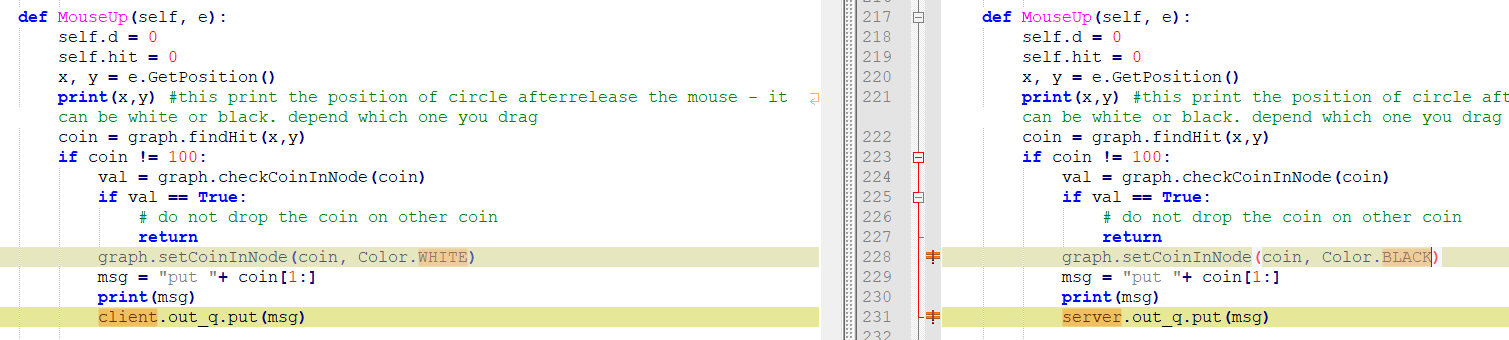


Here I compare the server and client files (use compare plugin of notepad++)









The client and serve code are different.

**Example 13**

import threading

import socket

import sys

import wx

from threading import Thread

from wx.lib.pubsub import pub

import graph

import comm

from time import sleep

from queue import Queue

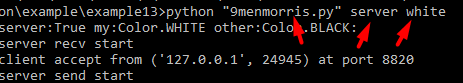
import argparse

from enum import Enum

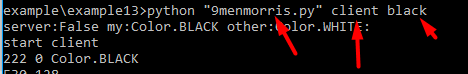
In this example I put the server code and client code in same file comm.py

And combine the 9menmorris-server.py and 9menmorris-client.py into single file 9memmorris.py

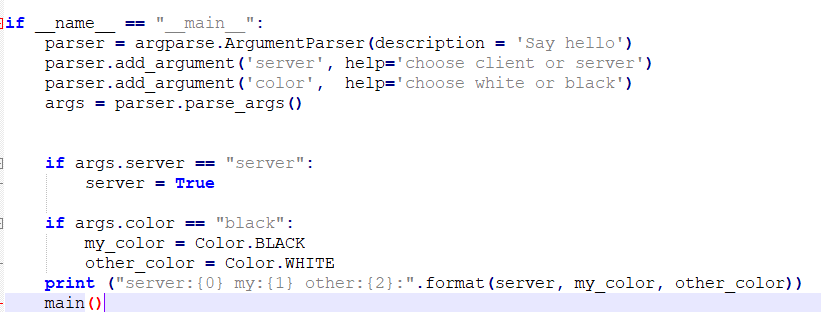
When you run the code in the server side and want to be white you do:



And in the client side



The code to force the user to choose side and color is



There is no code now that check that both sides did not choose to be server or in the same color.

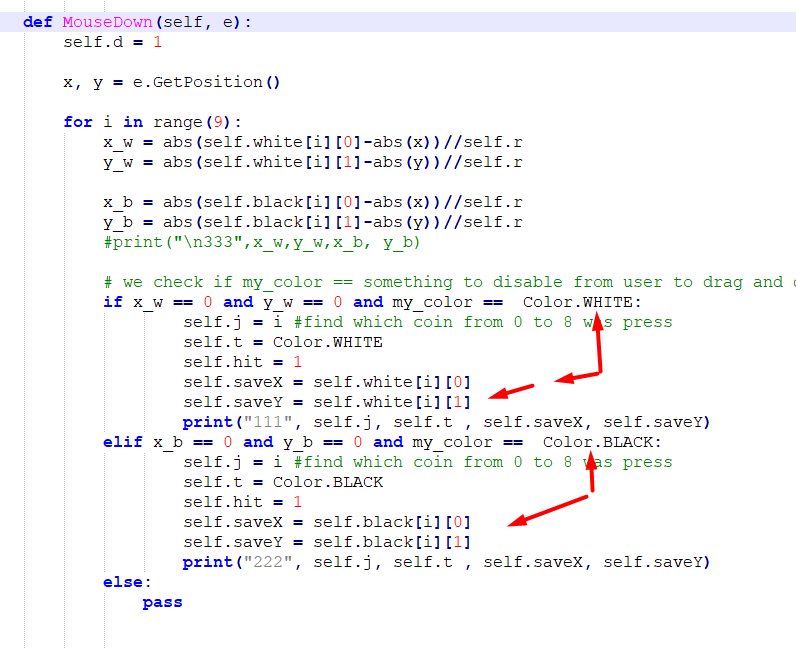
**Example 14**

In this code we prohibit to drag and drop the other side coin.

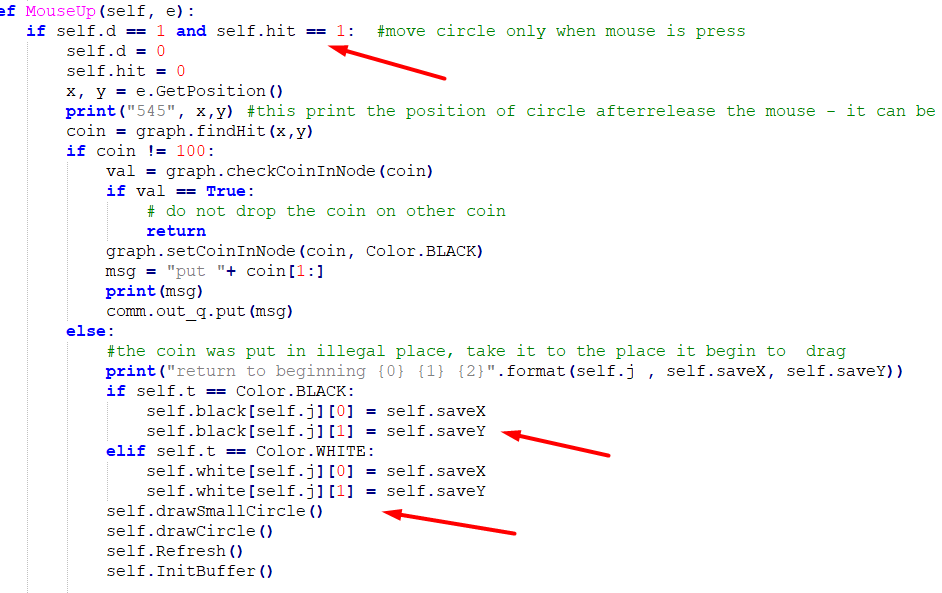
Means that if we white, we can not move the black by out mouse

Another feature is if you drag and drop the coin not in station, then it will return to previous location.

So each time the user press on his color coin, we save that place in saveX and saveY



and in case the user release the coin not in station, we move it to previous location



**Example 15**

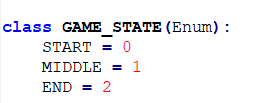
In this example I add logic to the software.

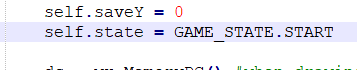
Morris game is divide to 3 states

Beginning – in this state the two sides just put the 9 coins on boards

Middle – two sides can move coins to adjacent nodes

End – only 3 coins are left, you can move your coin to any free station



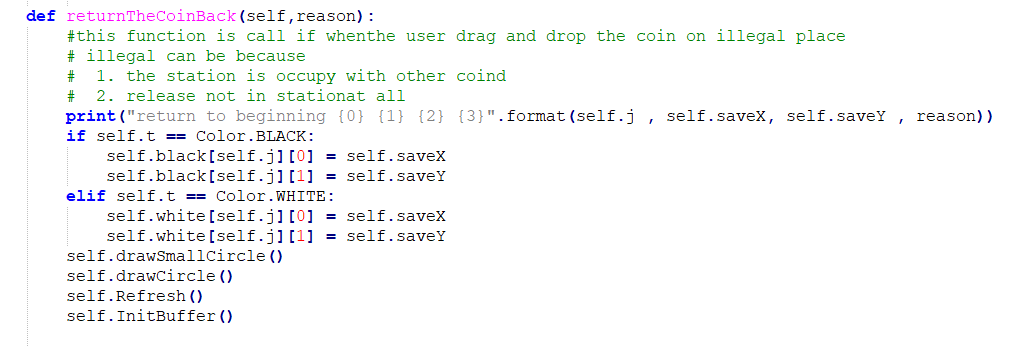


Now we add this state . we do not use it yet.

In this state we check if drag and drop movement is possible.

If the drag drop is not possible then we return the coin to its previous place.

I add a new routine for this , and the argument 'reason' describe why it was return



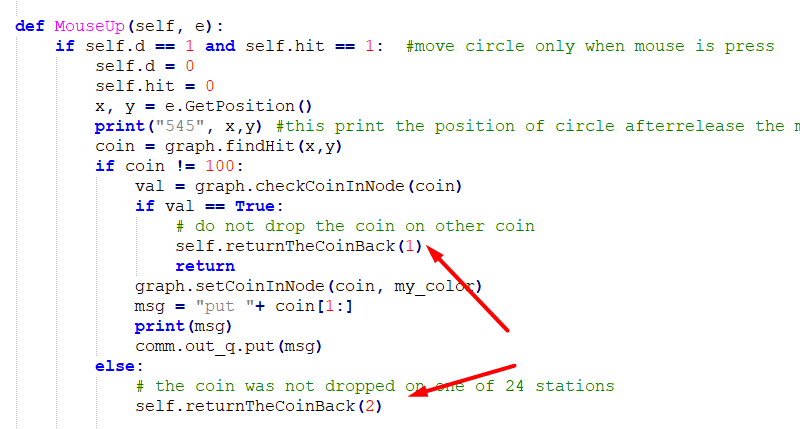
Important issues Is this the check if position is possible occur at the the code that you run in your location.

You do not check if the command that you get from other side is valid ! you trust the other side that it does the validity check and if the move in his side is not legal, thenit does not send you message.

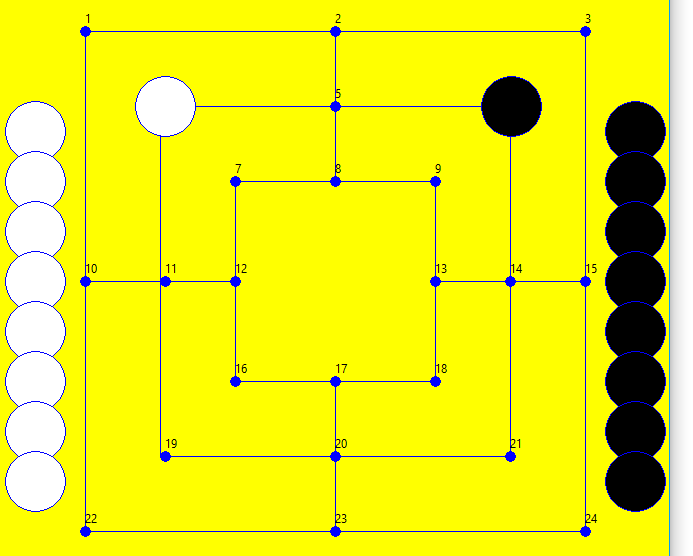
Again – each side do tests for drag and drop and in case the test fails, no movement occur and of course no message I sent to other side

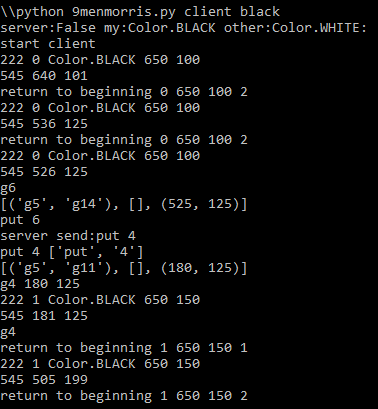
Again – you do not check commands from other side. You trust it.

It is like the code that you run in ASPX. You do not check the information from client side since it is the role of HTML/JAVASCRIPT to check it



The black begin then the white then the black try to move again but it once try to put it on other coin and once put it not in station





**Example 16**

In some of the examples the code use pull of coins for the put commands – "put 4" – was put the next coin in station 4.

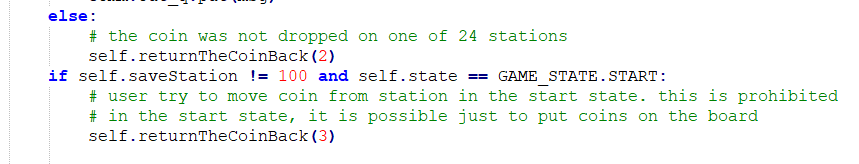
Since both sides have same information, I change the command and send "put 4 22" – it means put coin 4 to station 24

This command is send after release the drag and drop coin.

**Example 17**

It is not permit to move coin from station to station in the first state of game.

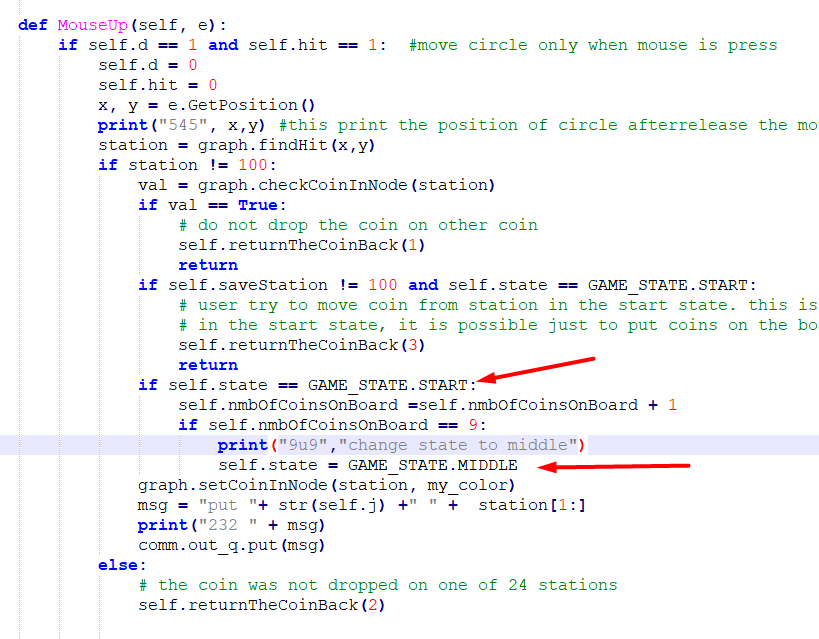
So if the user try to do it, then the coin will be return to previous location





The reason is 3

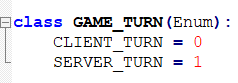
So just after put all the 9 coins on board , the game change state and it is possible to move

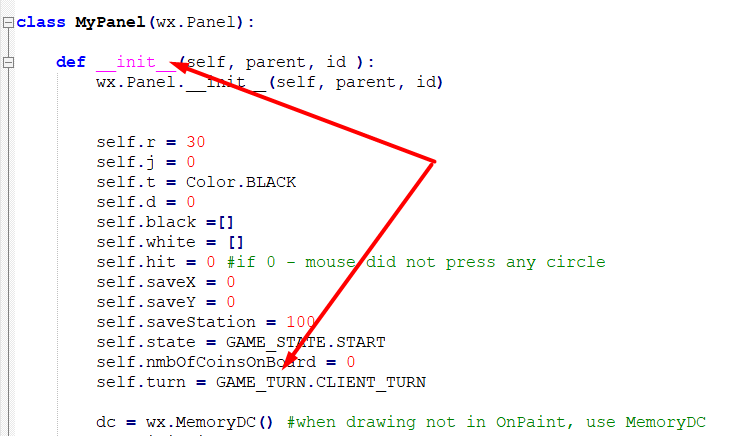


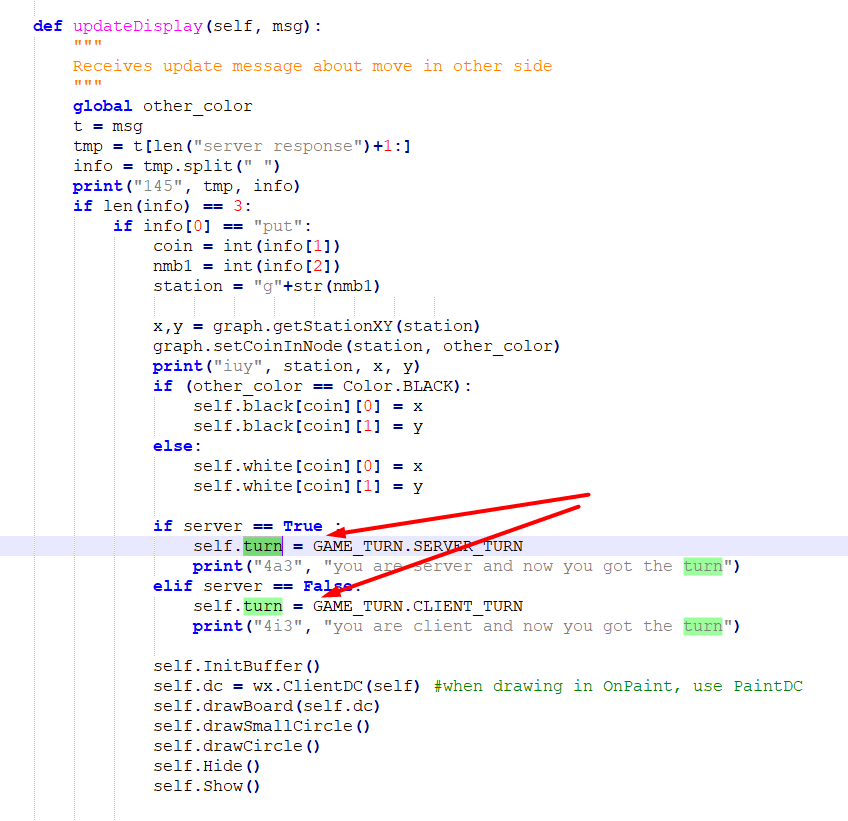
I add another feature – turn

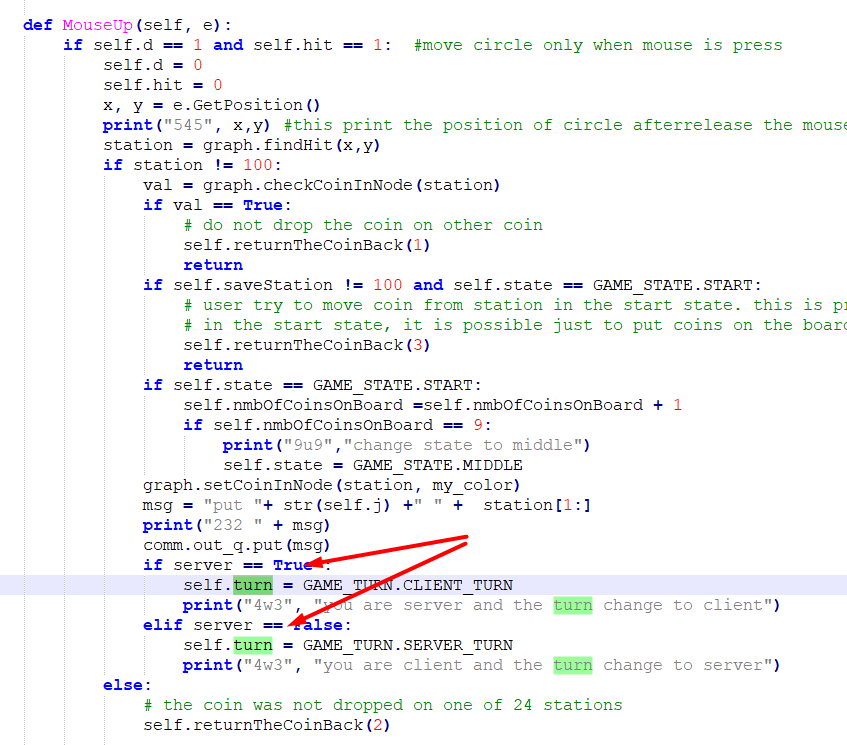
The game begin in the client side, he played and give the server to play, he responds and the client get the turn again

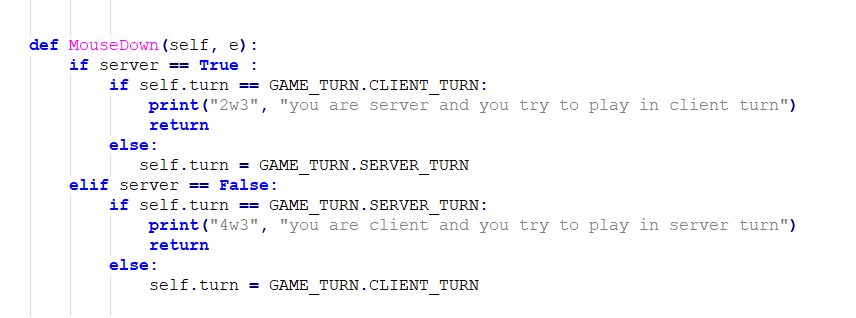
It means that we play each time in our turn

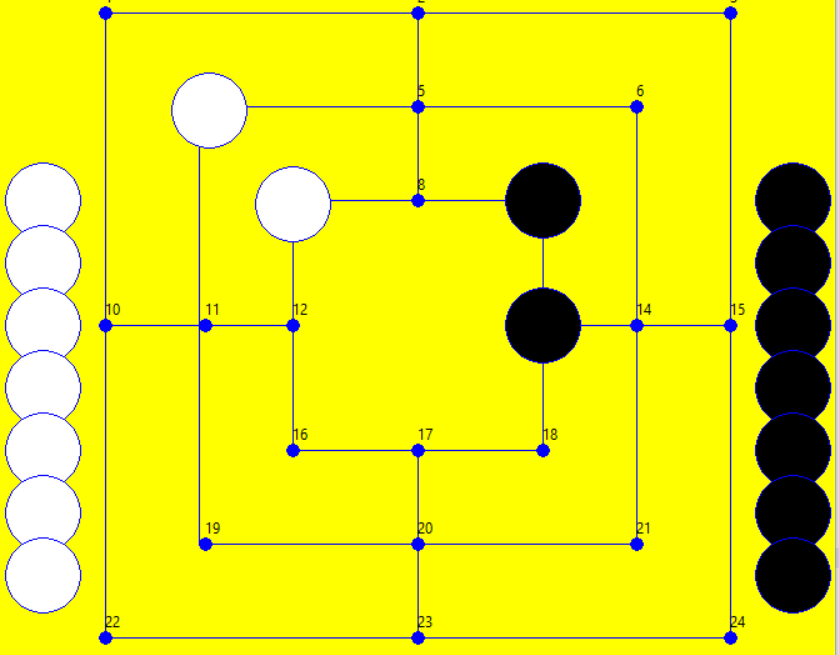


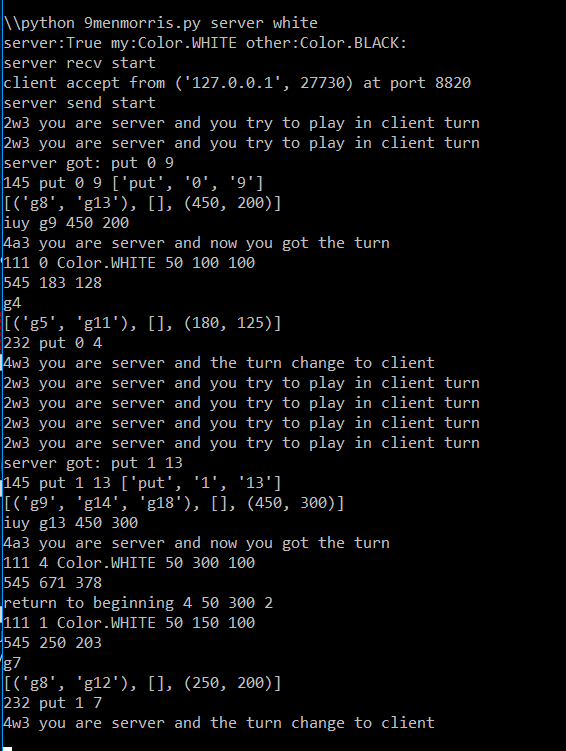


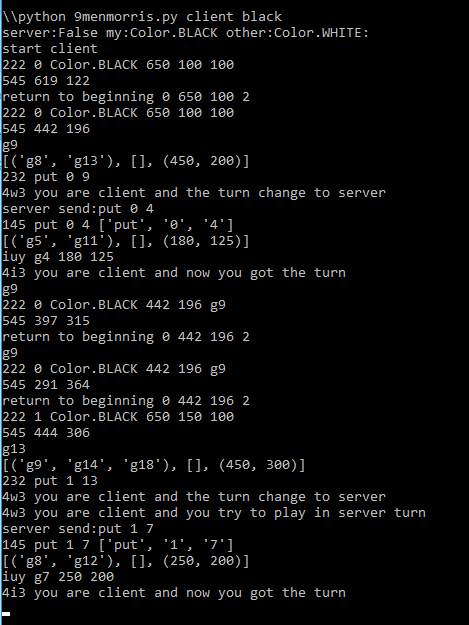






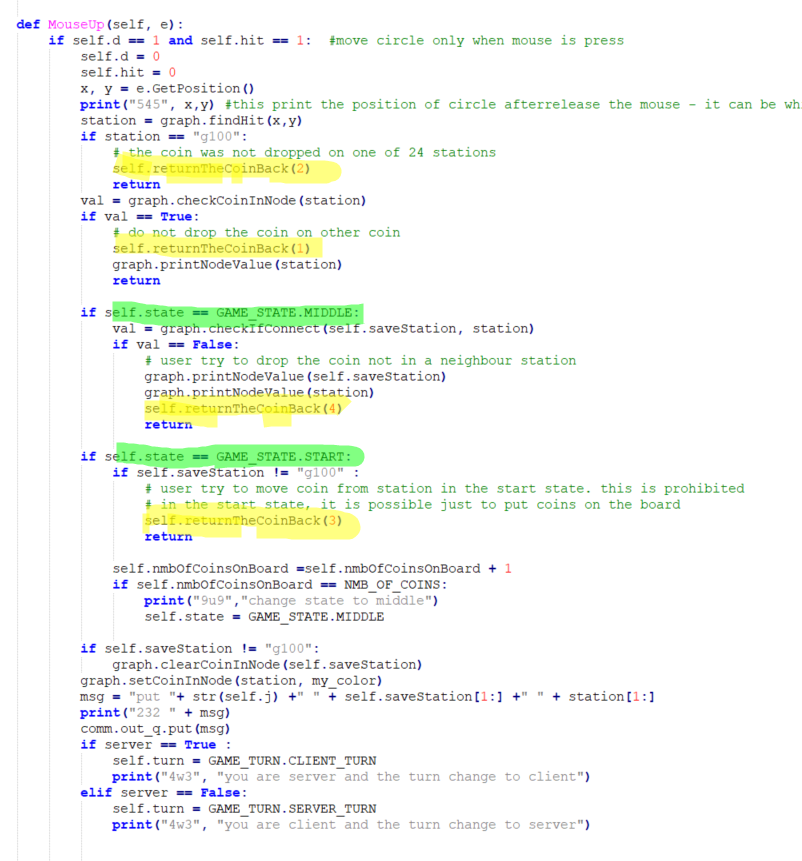






**Example 18**

Now I add logic for the middle game. In this state, all the coins are on board, and now it is possible to move coin from station to neighbor station . if the user try to move to other place it will return to previous position.



As I write above the graph db in both side is identical.

I change the message from one side to the other 'put <coin> <previous station> <next station>

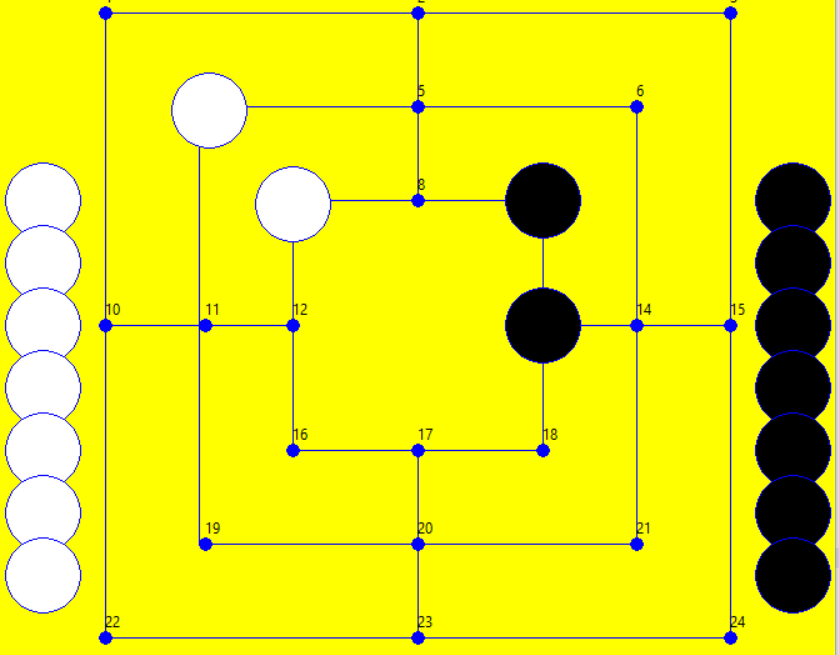
For example

Put 4 g6 g8

**Example 19**

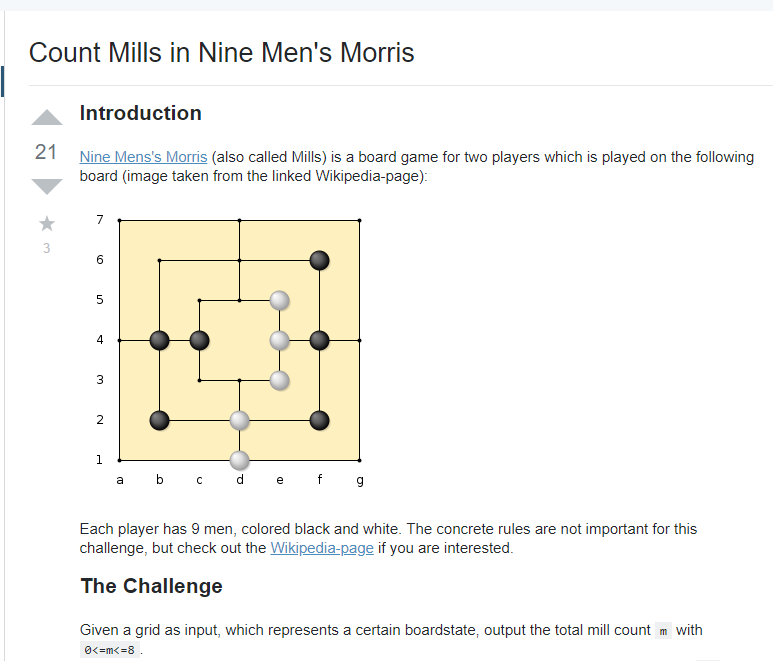
In this example we add the feature of take coin of your opponent in case you create a mill ( 3 coins in line – horizontal line or vertical line)

For example 3 white on stations 4,5,6 is a mill

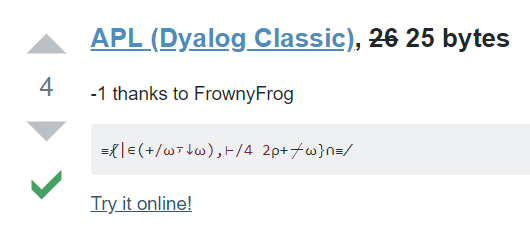


Look on this challenge, the solution is amazing.

<https://codegolf.stackexchange.com/questions/73412/count-mills-in-nine-mens-morris>



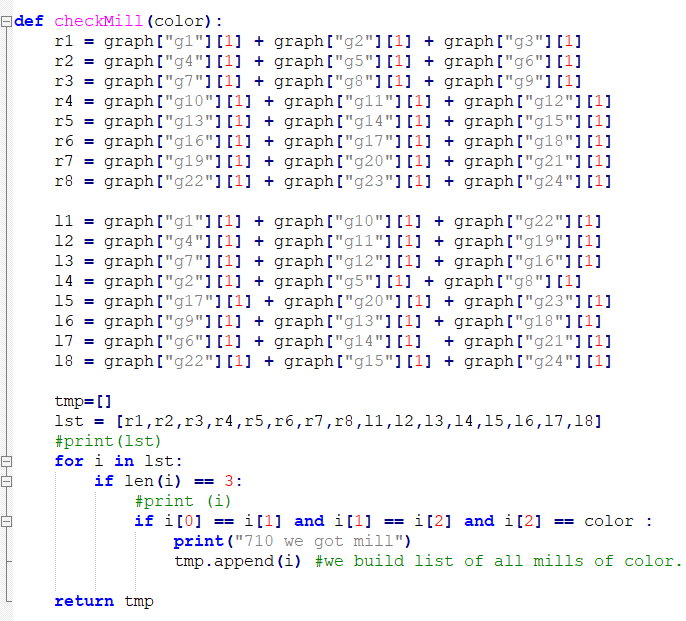
Here is the solution



This is my [reaction to solution](https://www.mojilala.com/stickers-emojis/what-the-fuck-emojis)…..



So I add new code in grap.py to get all the existing mill of some color



There could be few mills of the same color.

The user can take one if his opponents just in the moment he create mill.

In the next turn that the user run, it is possible that he mills did not change so, the user can not

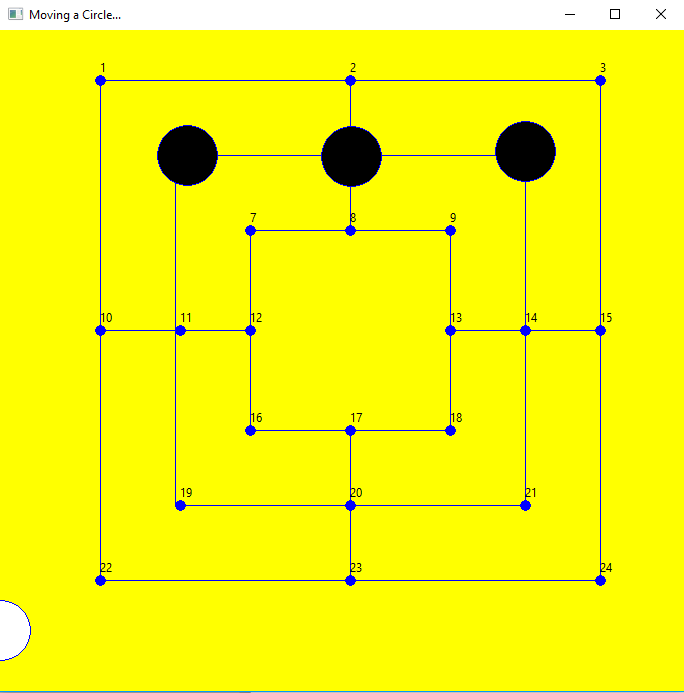
Take again coin from his rival.

Again – only when a mill is create , it is possible to take coin from rival.

One of the strategies is if you got mill in the next turn broke it and in the next next turn build it again…and in the next next turn you can take coin again from rival.

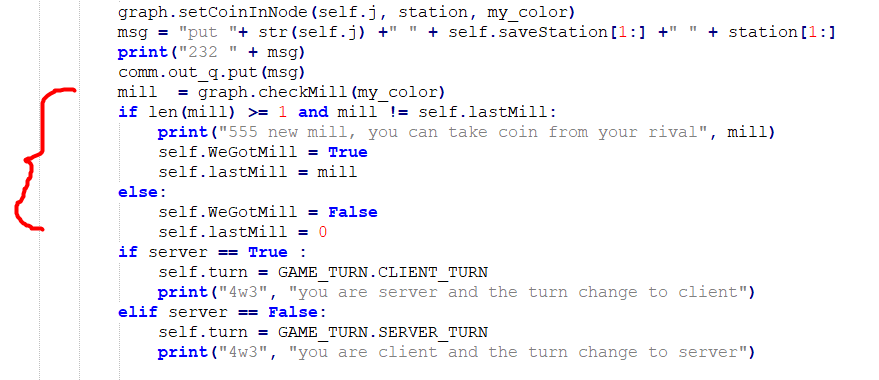
Here is the game after the black remove 3 whites (for testing, I work with 3 coins , it is faster)

The white are move to bottom left (x=0, y=600)



Here Is the code.

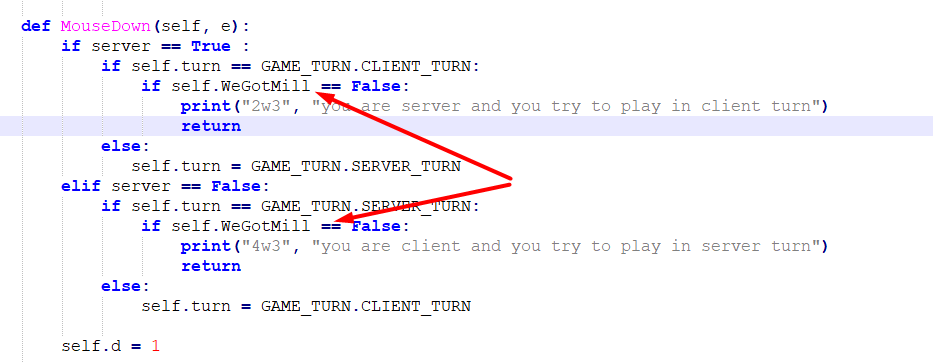
In the mouseUp ( when we place the new coin on board or when we move coin from station to station) we check if we create new mill (it is important that the last move create a new mill. If there was a mill and the last move did not create a new mill – this is not omportaant. Just last move that create new mill)



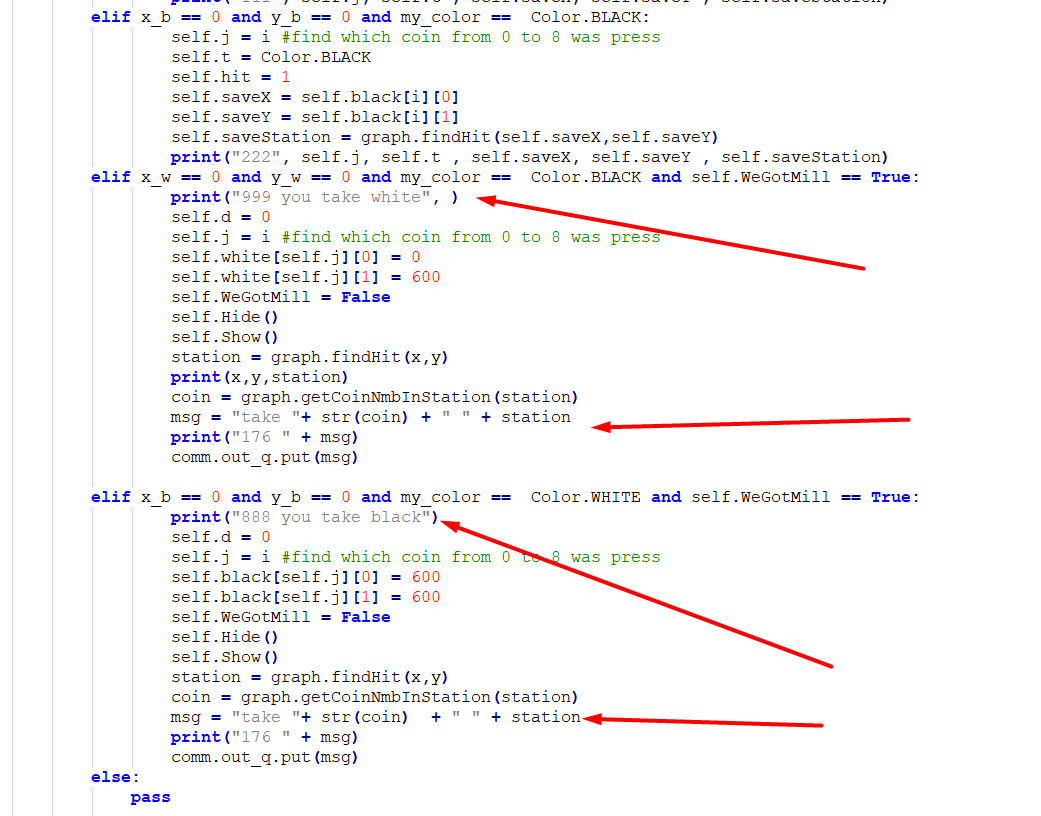
Now you shall take one of the coins of your rival ( ###todo you may not take from a mill of your rival if he had other coins not in a mill – I did not implement it , yet)

So now we go to the code of moudeDown.

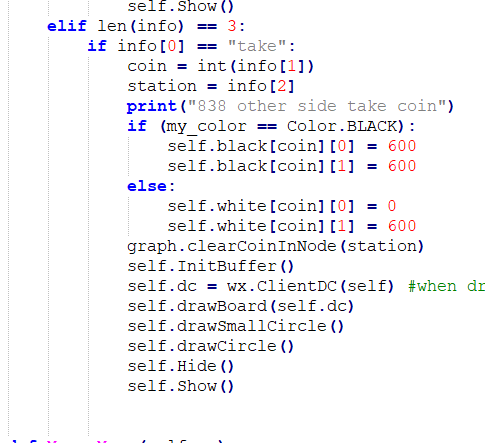
Normally after you finish a move, the turn go to the other side, but you finish the move and you need to take a coin of your rival, so you do moudeDown event. It is legal for you just to take coin



And now we take the coin and we send message to other side to remove the coin from board



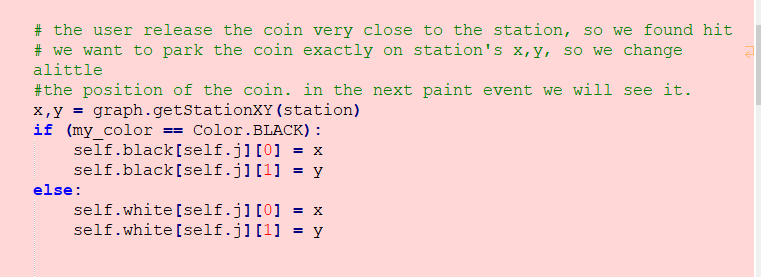
Here in updateDisplay we handle the new message take <coin> <station> e.g take 3 g8



**Example 20**

In this example I did some test, find few problems, so in order to see the problem, I comment few print ( too many print hide what happen and add 'printGraph' in important areas)

I did small adjustment to the coin – when some one release it on the station, it was not exactly on the station but within a range of 30 (findHit). What I add is move it to the exact location



**Example 21**

import threading

import socket

import sys

import wx

from threading import Thread

from wx.lib.pubsub import pub

import graph

import comm

from time import sleep

from queue import Queue

import argparse

import pickle

In order to make the test easy, after each change, I send the whole graph to other side to compare it.

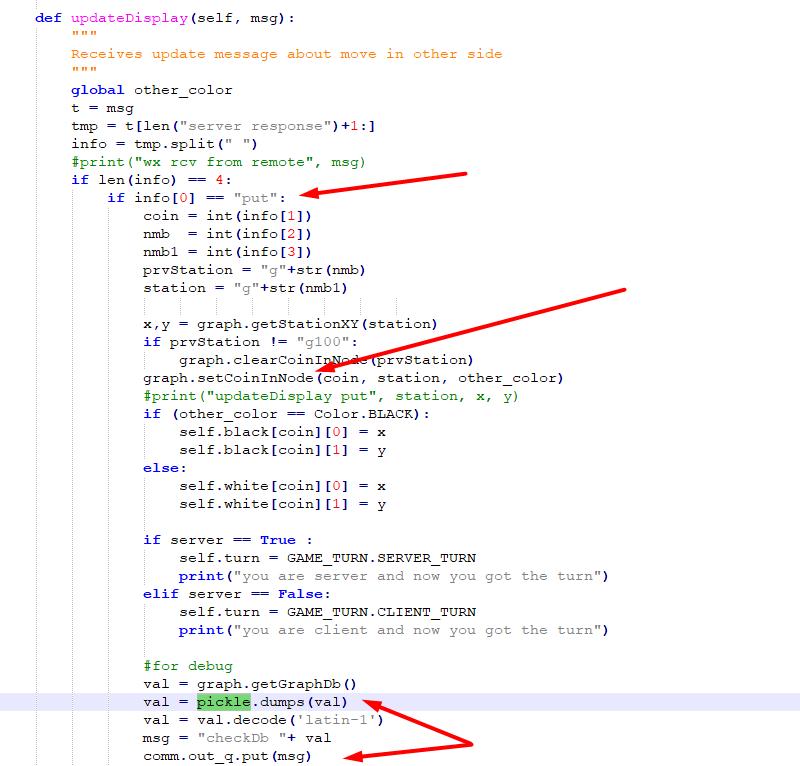
For example after the client do some move and update his grahDb, it send the move to the server, the server reflect the client move and update his graphDb. Then the server send the graphDb to compare.

Then the server do some move, it update his graphDb, and send message to the client.

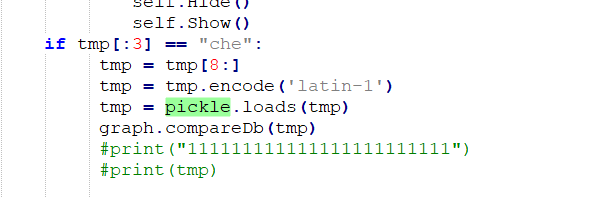
The client reflect the server move on his board and update the graphDb. Then the client send to server the graphdb for compare.

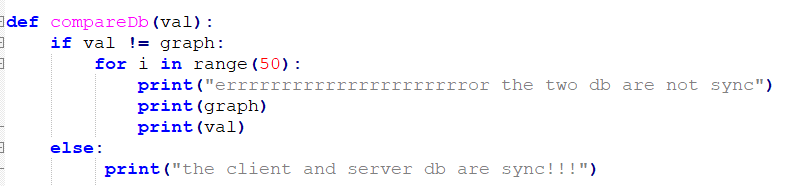
This is the new code

When we got 'put' command from remote, we change the board and then send 'checkDb' command with our graph toremote site. In remote site there will be comparison



In remote site when we got 'checkDb' command



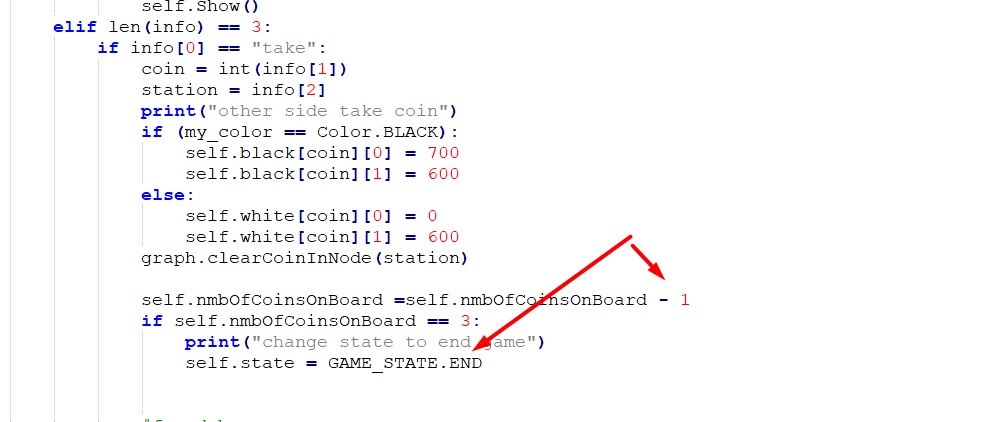


I comment out more prints

**Example 22**

In this example I add the last stage of game – when there are only 3 coins and each coin can move to any place

When we got 'take' command from remote we do



#todo – there is no test that check if one of player can not move at all. In such case he lose.

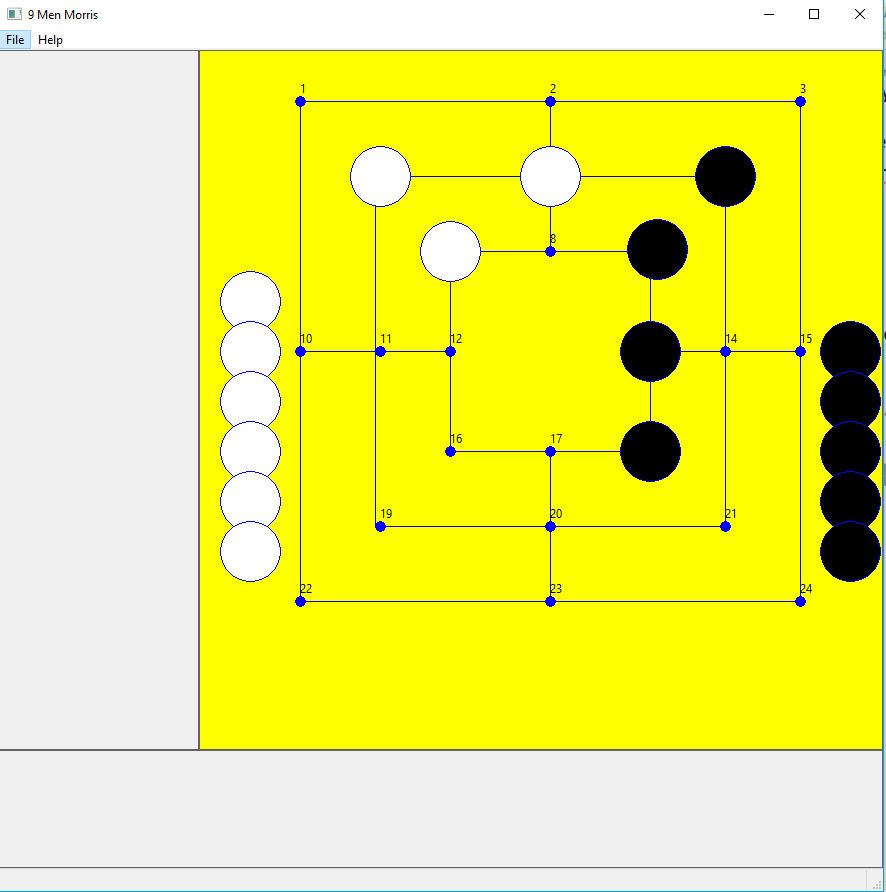
I also did some change in the logic of checkMill

I see that there is some bug, I try to move a coin to empty station but it did not move.

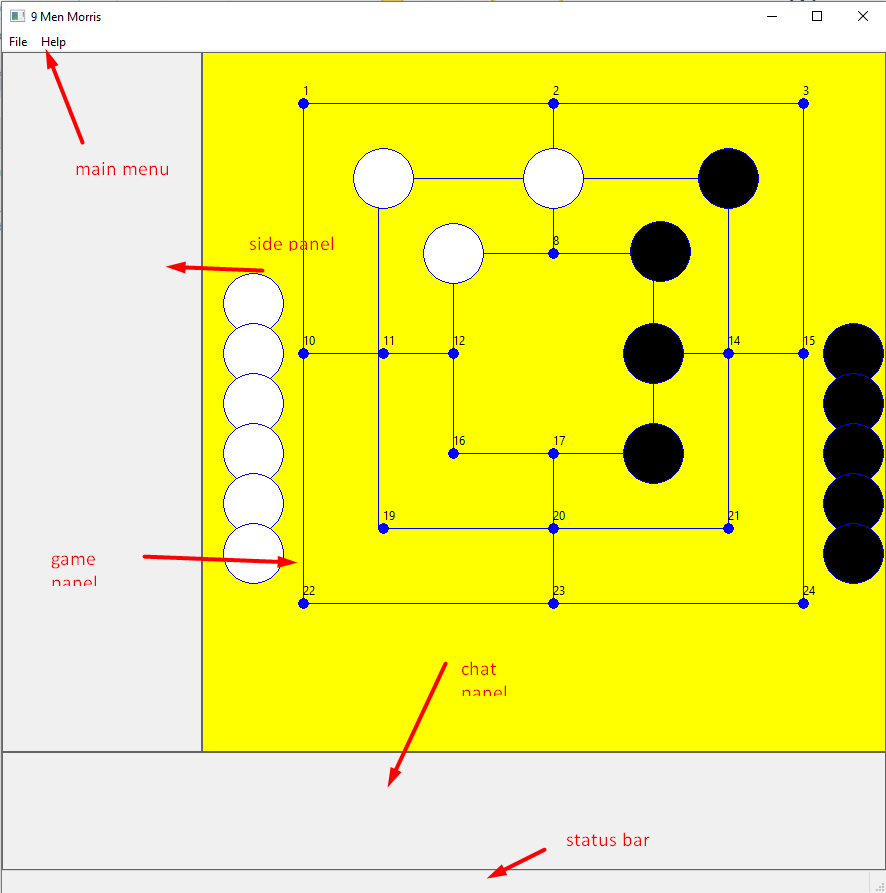
I need to add some option to see debug information in real time. It I will add some menu to the game, then in the menu I can press on some button and this will printGraph or something like this.

**Example 23**

Until now we got single window for the game. We want to put the game in some panel like this



This window is divide to few areas



How to do it?

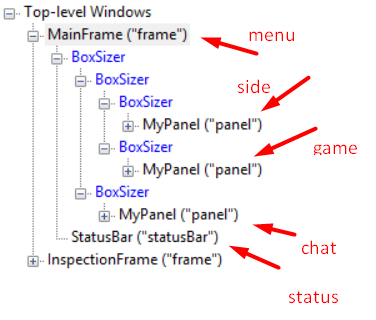
There is an example that it just create few panels



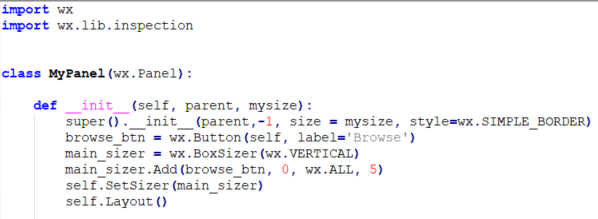
To create this we need to create some hierarch of window in window. This is how it looks like – you create frame and in the frame you put panel in the panel you put boxsizer (main). In the boxsizer you put two horizontal boxsizers (row and row1) in row you put two vertical boxsizer(col1 and col2) in row1 you put panel(chat)

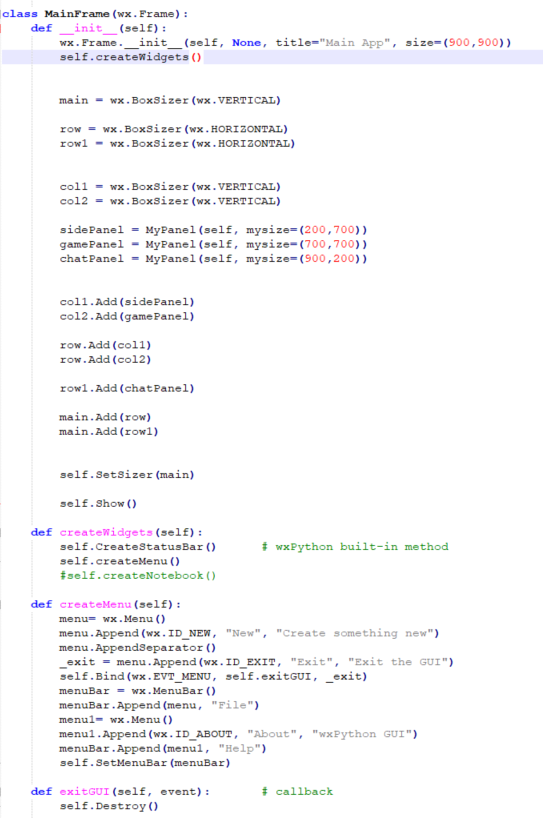
In col1 you put panel(sidepanel) and in col2 you put the game panel…

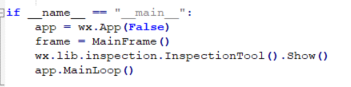
( there is wx inspection tool that shows it)



Here is the code

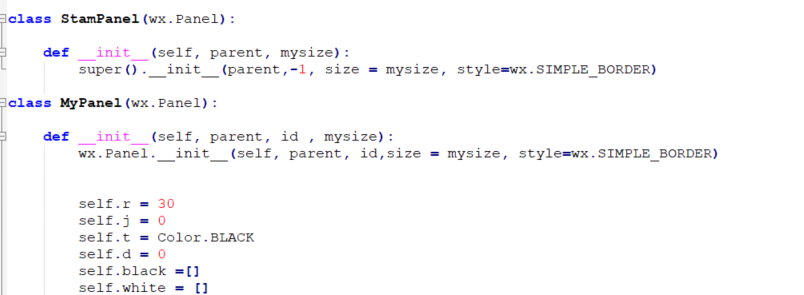


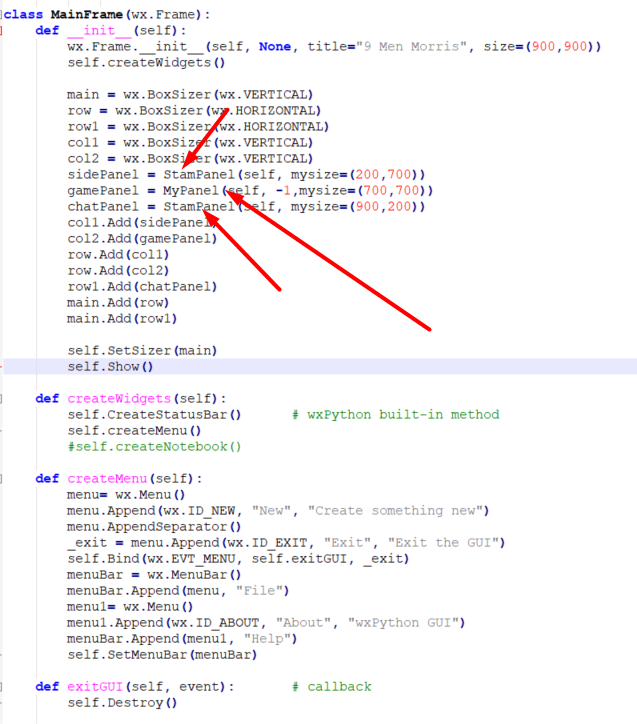


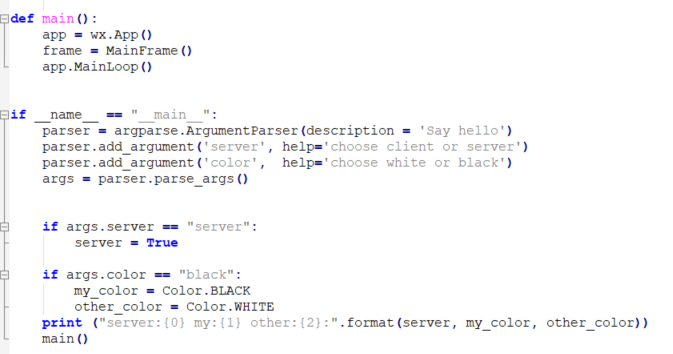


**Example 24**

We took the frame from example 23 and put our game in the game panel.

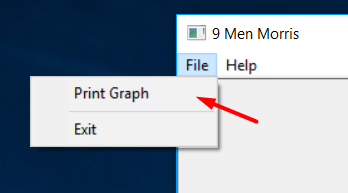




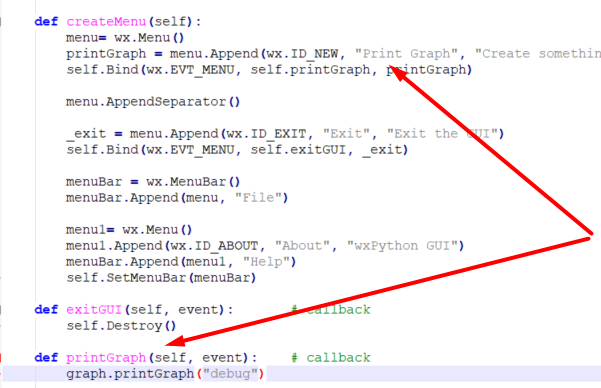


**Example 25**

In this example I add debug option to the menu.



And here is the implementation



**Example 26**

In this example I will write important message to chat panel

Message like change state from start -> middle ->end

Or you can take coin from your rival

#todo - In the status bar I will write the information like who got the turn now your turn / rival turn

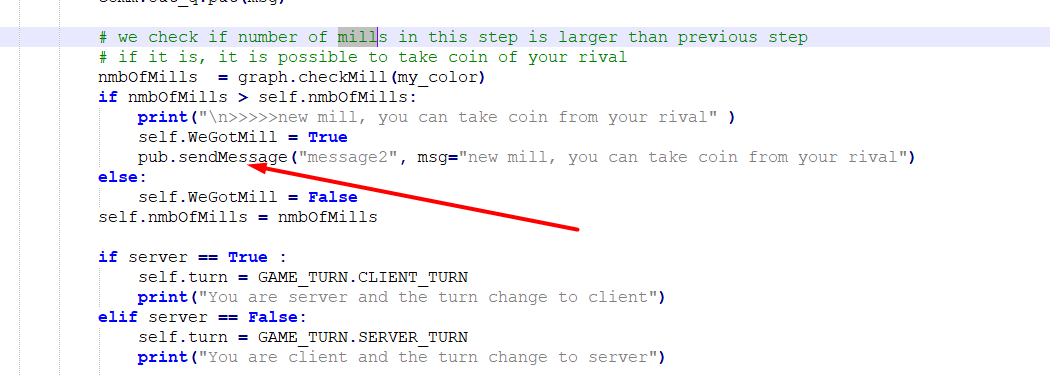
To communicate from one class MyPanel to other class ChatPanel I see that one of the recommended communication methods is PUB/SUB .

So the chat panel is the subscriber and it wait for messages on some topic

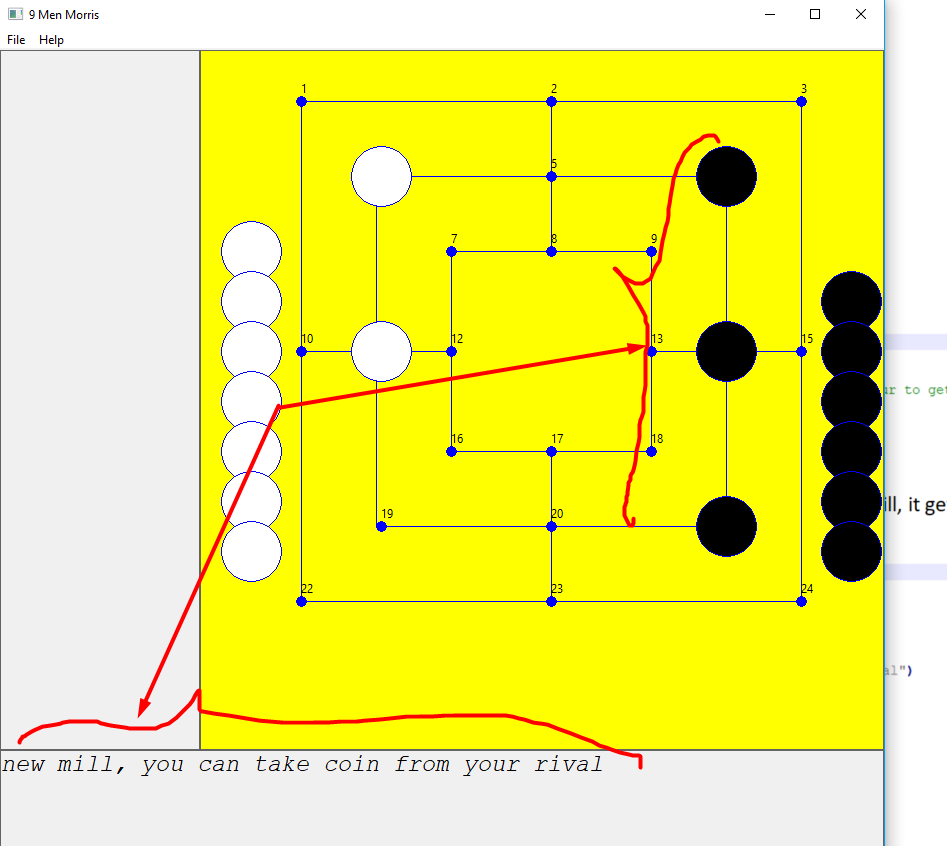
If it get message it writeMsg



And in the whole code I publish messages – for example when the user build a mill, it get message – you got a mill



And this is the board

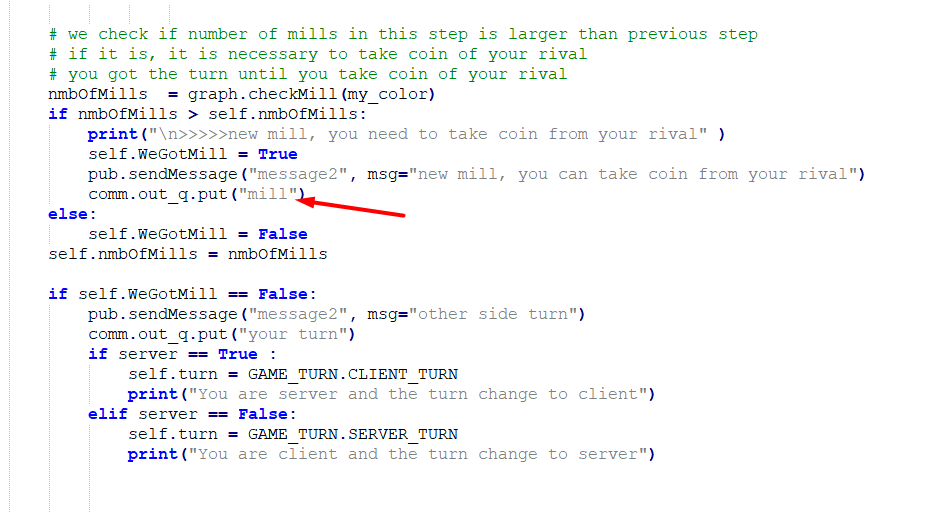


**Example 27**

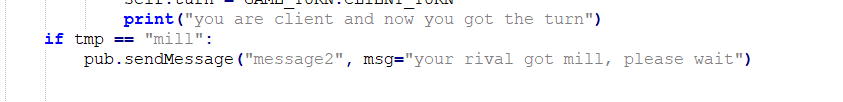
In this example I add more messages to chatpanel

I add two message to client/server – 'mill' and 'your turn'

In case of of getting mill, it is just for charPanel

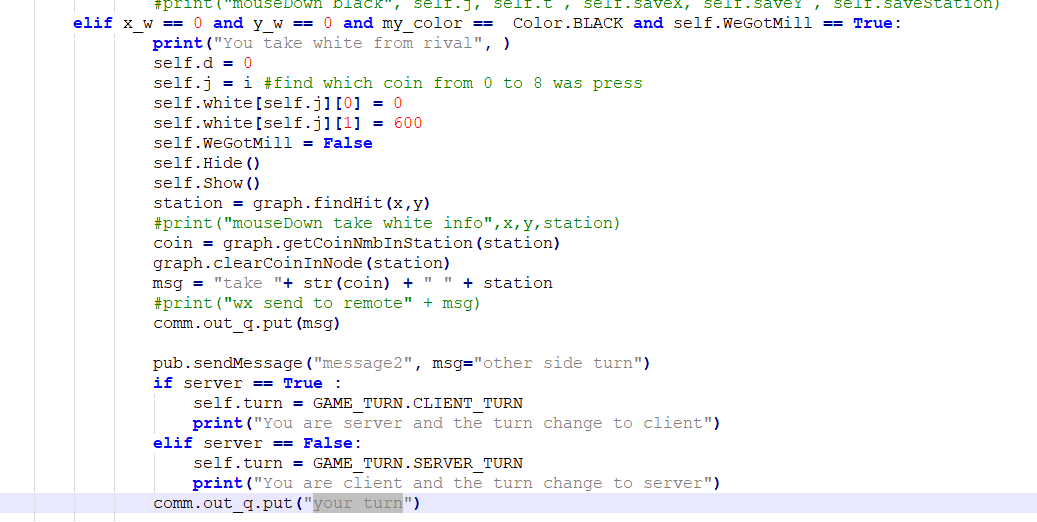


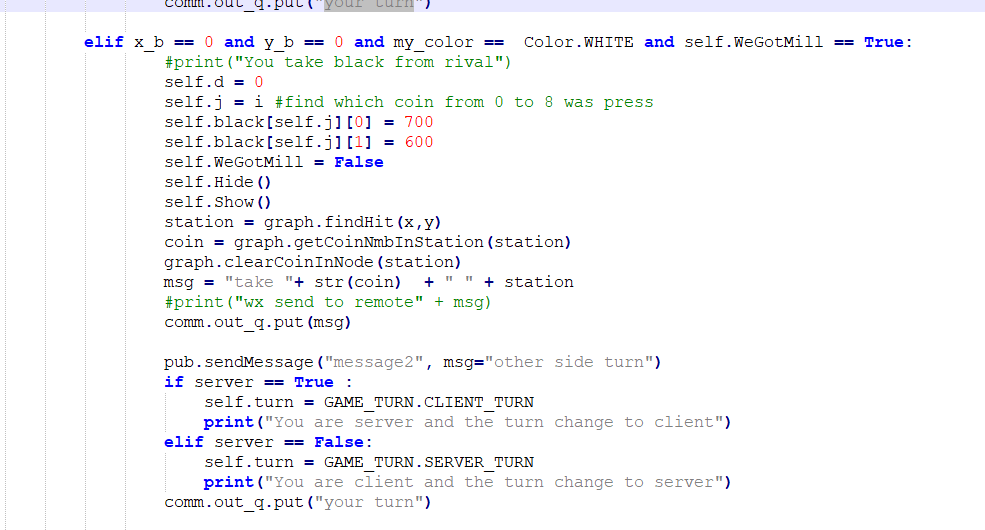
And here we handle the message



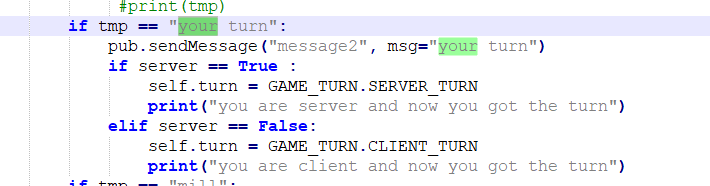
Now you send explicitly message that it is your turn







This is in the receiver



**Example 28**

import paho.mqtt.client as paho

In this example we do small test with mqtt – we publish message and then we read it.(we check the retain option of the broker – save last message of topic)

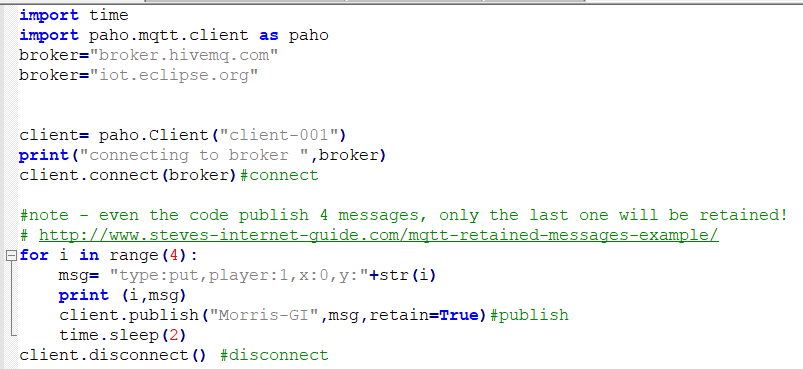
We use broker in internet

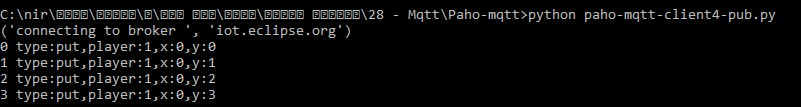


I read about paho-mqtt in this site

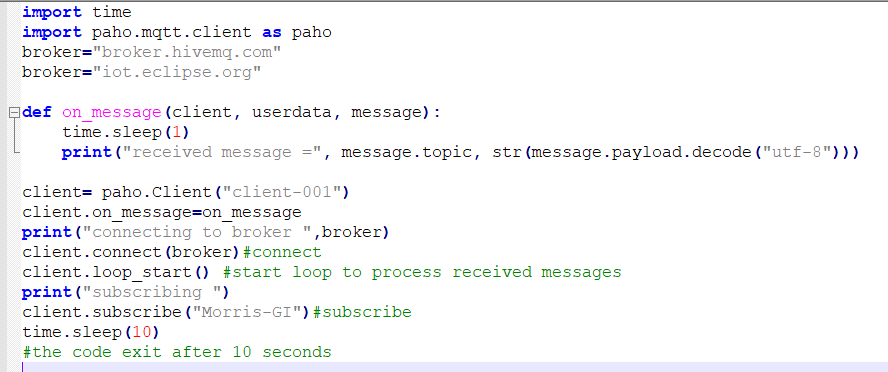
<http://www.steves-internet-guide.com/python-mqtt-publish-subscribe/>

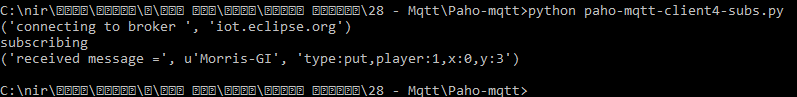
Publisher





Subscriber





**Example 29**

Now it is the time to integrate the mqtt to the game. Use mqtt or sockets.

I have a file name comm.py that implements the socket communication.

I save it as comm\_org.py

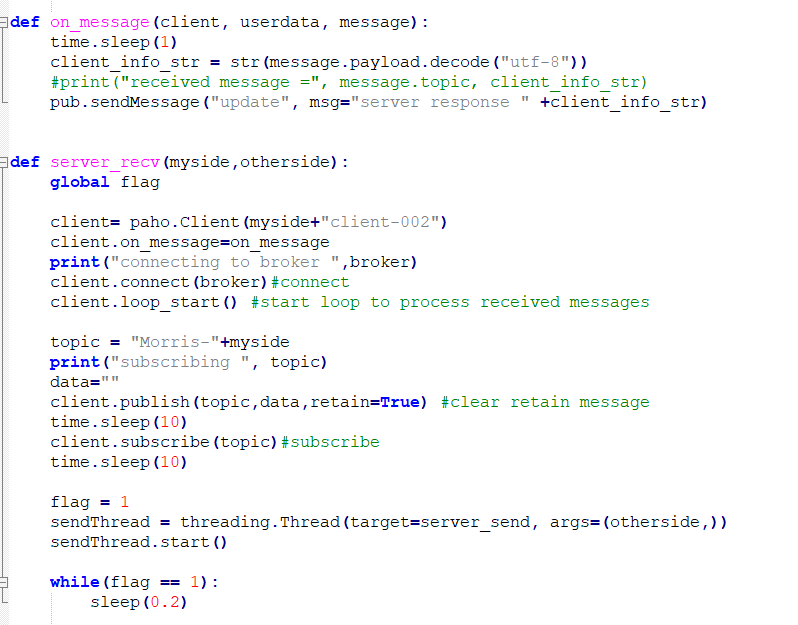
Now the file comm.py holds the communication level. It is mqtt. The code here is based on comm\_org.py comm with adaptation to mqtt.

There is a thread to publish mqtt message. It wait on out\_q (internal messages from wx), once he got message like put/take etc, it publish it

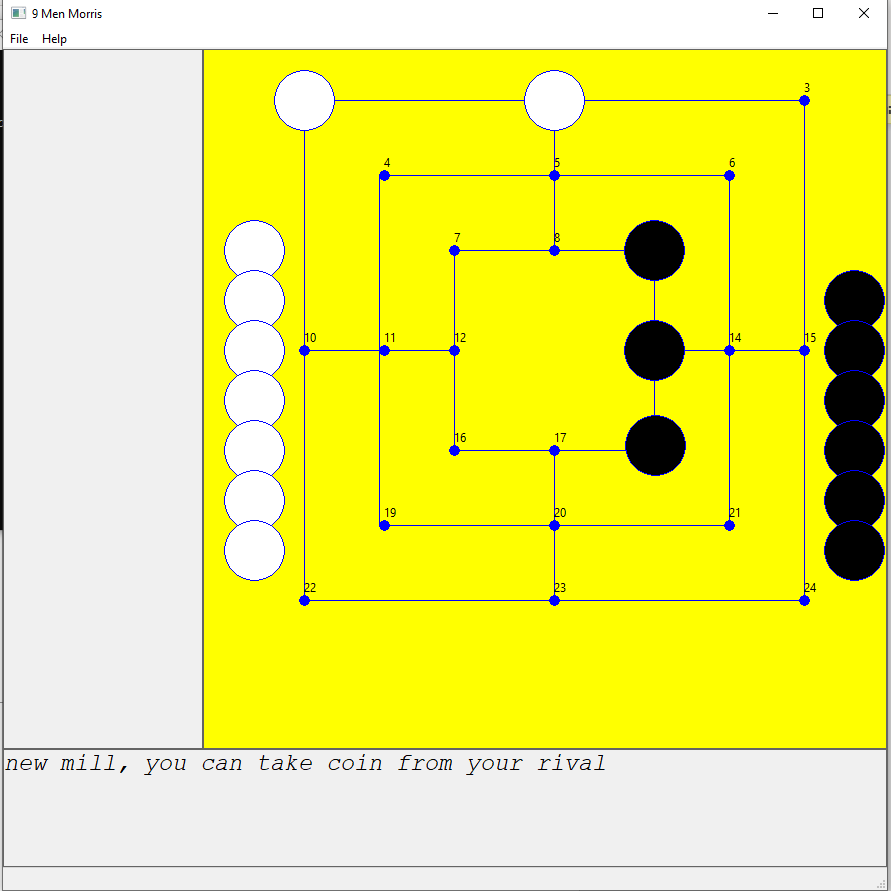
There are two mqtt topics – morris-client and morris-server

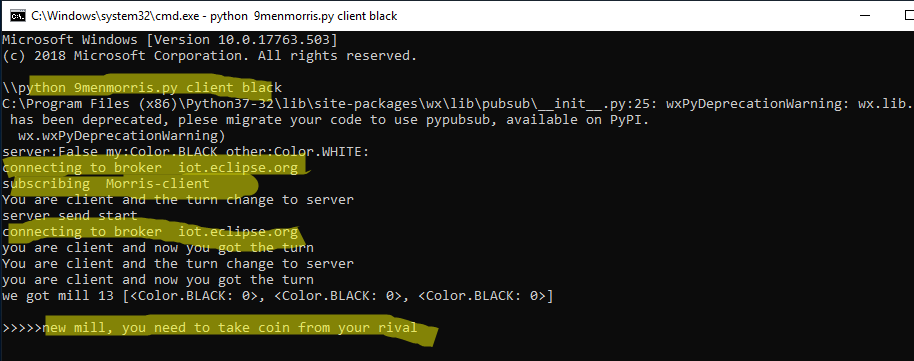


There is another ,it is for receiving mesaages.

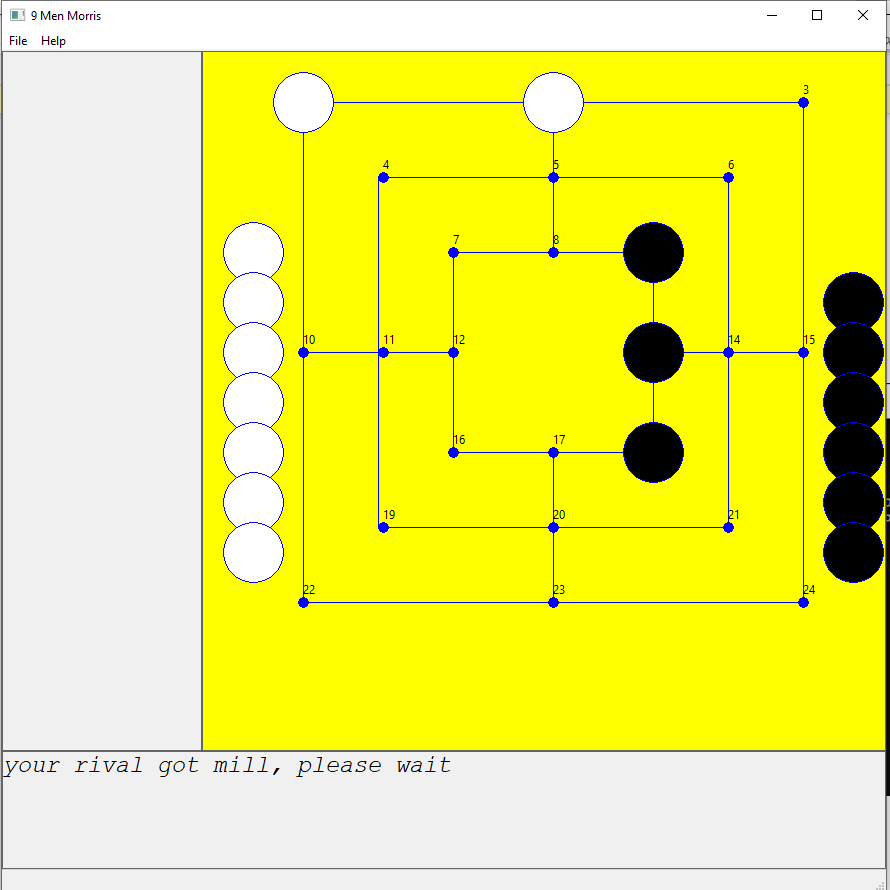


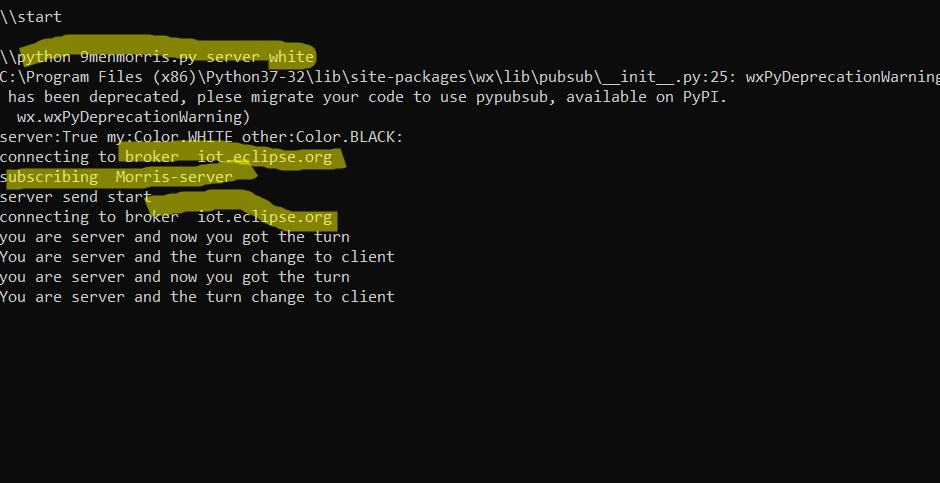
Here is the client side, it move 3 coins and now got mill



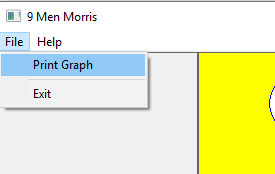


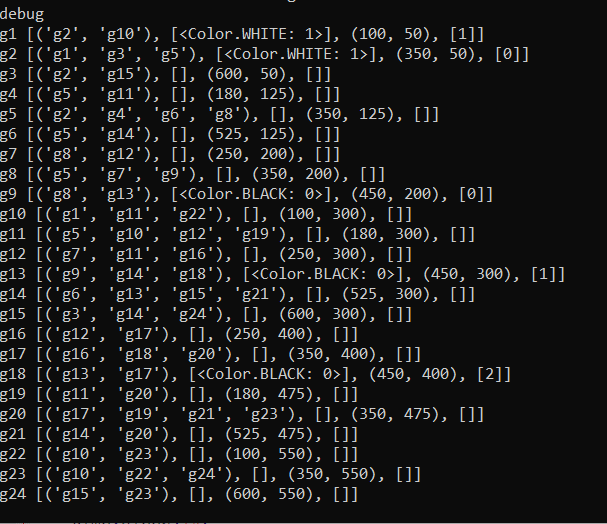
And here is the server side





I can print the graph db for debug





#todo – I see that if I stop send messages from side to side and then continue, the last message is lost.

We should use mqtt better – like tcp message that does not lose.