Compared the two new and old files and commented where they differ, edited the old file to match the new where code is not different.

If increasing iteration number decreases rate of good moves, C might need to be changed.

Make the actual game now.

The non class game program is done, make with class now and decide where to put the root creation, game tree, and other statements.

Lambda is like a mini function:

print(sorted(game\_tree[1], key=**lambda** node: node.value[1], reverse=**True**))

Here, the function sorted takes a thing to iterate, game\_tree[1]

It then takes a key value, this is the value that is compared to other values and is used in iteration.

The key value can take 1 input so that the value to be iterated (the key value) is returned.

The lambda function takes in a variable input node, and returns node.value[1], which will be compared to other node.value[1]’s to sort the game\_tree.

Reverse= True causes the sorted iterator to be returned in descending other instead of the default ascending order.

**def** \_\_repr\_\_(self):  
 *# return '{}, {}'.format(self.value) # self.parent is repr'd here* **return** str(self.value)

**def** hello(self):  
 print(**'yes'**)

the repr(self) of this class defines how it is represented, so that if it is added in list z.

print(z) will return the str(self.value) for that instance.

However, z[0].hello() can still be called, and will still print ‘yes’.

There is a difference in the turn values for the actual simulated game, and each simulation of the game in

**def** simulate(self, selected\_node):

All the turn values need to be dealt with.

Figure out all use cases

Use turn finding function instead of counting.

On turn=3 in Game, the monte carlo class ignores the root node update.

And figure out the order of the algorithm and be clear on this.

Try a different order.

Check each part of the Monte Carlo class separately, aswell as the Game class.

Fix errors etc..

Make it better

Implement difficulty modes

Finalise

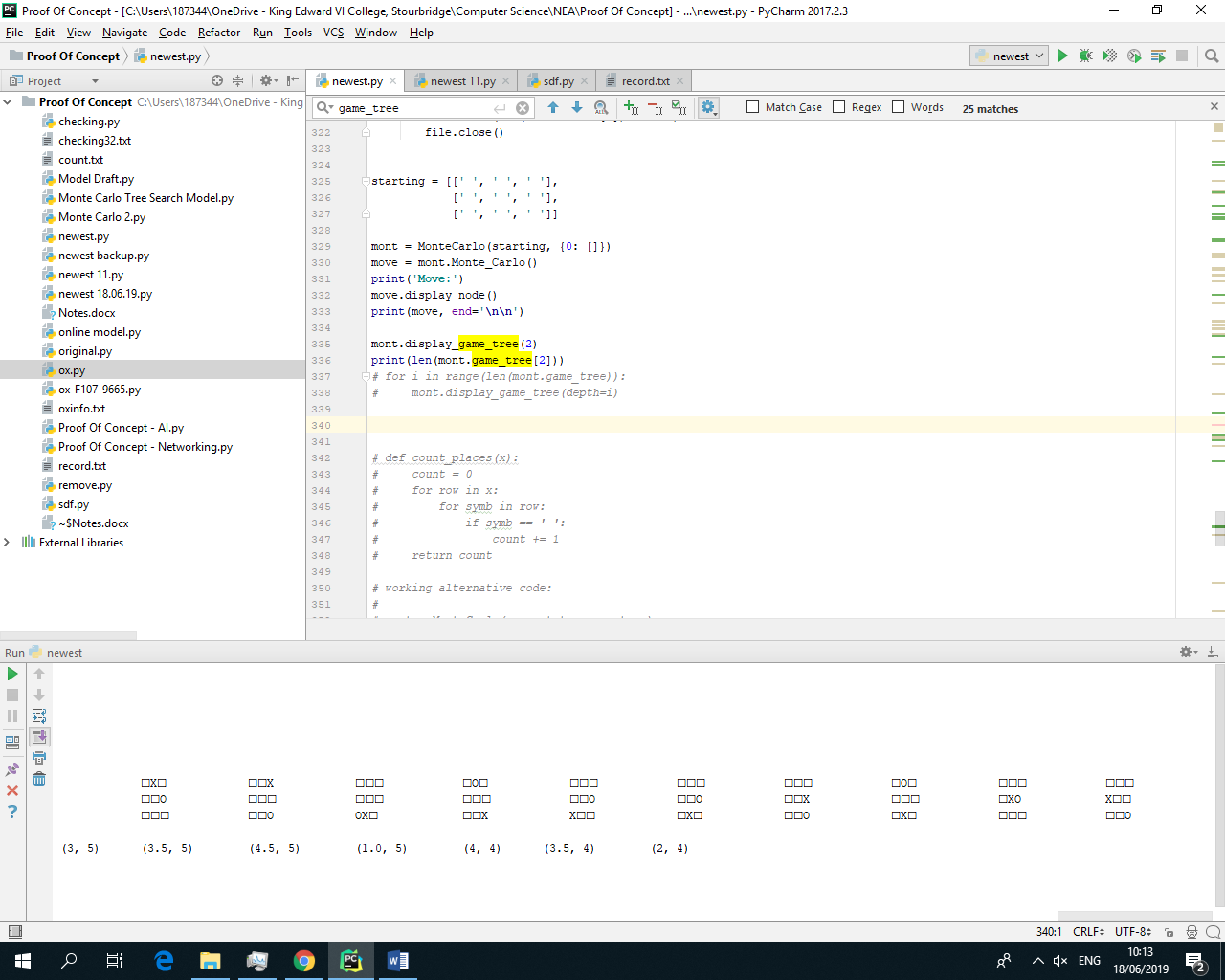
Do the ultimate Tic Tac toe

Game tree is used to find the node at 1 with the highest UCT value.

Game Tree is stored in the Monte Carlo class.

The selecting, expanding, simulating, and back propagating uses parent/child lists in the Node class.

Is the Game Tree updated when the node instance is updated?



At game\_tree depth=2

Produces 80 different possible nodes.

Some nodes do not have UCT values?

How to visualize game tree?

Check each part of Monte Carlo class and see if it works properly together

Check and make best algorithm possible.

Value for c?