Joel Trainer
Assignment 6:

Initially I started off by making a copy of the calc code and I made a change so that it could send the packet with 9 numbers instead of 2 and checked it worked in the python and p4 code.

In the python code

```
class P4calc(Packet):
    name = "P4calc"
    fields_desc = [ StrFixedLenField("P", "P", length=1),
                    StrFixedLenField("Four", "4", length=1),
                    XByteField("version", 0x01),
                    StrFixedLenField("op", "+", length=1),
                    IntField("operand_a", 0),
                    IntField("operand_b", 0),
                    IntField("operand_c", 0),
                    IntField("operand_d", 0),
                    IntField("operand_e", 0),
                    IntField("operand_f", 0),
                    IntField("operand_g", 0),
                    IntField("operand_h", 0),
                    IntField("operand_i", 0),
                    IntField("result", 0xDEADBABE)]
```

```
 try:
     pkt = Ether(dst='00:04:00:00:00:00', type=0x1234) / P4calc(op='+',
                                         operand_a=int(place_a),
                                         operand_b=int(place_b),
                                         operand_c=int(place_c),
                                         operand_d=int(place_d),
                                         operand_e=int(place_e),
                                         operand_f=int(place_f),
                                         operand_g=int(place_g),
                                         operand_h=int(place_h),
                                         operand_i=int(place_i))
```

And on the p4 code

```
header p4calc_t {
    bit<8> p;
    bit<8> four;
    bit<8> ver;
    bit<8>  op;
    bit<32> operand_a;
    bit<32> operand_b;
    bit<32> operand_c;
    bit<32> operand_d;
    bit<32> operand_e;
    bit<32> operand_f;
    bit<32> operand_g;
    bit<32> operand_h;
    bit<32> operand_i;
    bit<32> res;
}
```

I then changed the conditions of a valid input from the user.

```
s = input('board:\n'+str(place_a)+' '+str(place_b)+' '+str(place_c)+'\n'+str(place_d)+' '+str(place_e)+'
if s == "quit":
    break
print(s)
valid = 0
if s == '1':
    if place_a == 0:
        valid = 1
        place_a = 1
    else:
        valid = 0
elif s == '2':
    if place_b == 0:
        valid = 1
        place_b = 1
    else:
        valid = 0
elif s == '3':
    if place_c == 0:
        valid = 1
        place_c = 1
    else:
        valid = 0
```

```
if valid == 1:

    try:
        pkt = Ether(dst='00:04:00:00:00:00', type=0x1234)


else:
    print("invalid move")
```

The code now checks if the input is just a 1-9, if so it sends the packet, else it will say invalid move.

Next I needed to make it send back a packet with the computer's move.

```
/*Make player 2 move*/
if (hdr.p4calc.operand_a == 0) {
send_back(1);
}
```

Using the same send back code as the calc code it sends back 1 as a result, if 1 was an available move.

```
if p4calc:
    if str(p4calc.result) == '1':
        place_a = 2
```

Now when the python code receives the packet it looks at result and changes the board.

```
board:
0 0 0
0 0 0
0 0 0
 > 5
5
board:
2 0 0
0 1 0
0 0 0
 >
```

I then extended this code but for all the board positions so now the game can be played until the board is full.

```
/*Make player 2 move*/
else if (hdr.p4calc.operand_a == 0) {
send_back(1);
}
else if (hdr.p4calc.operand_b == 0) {
send_back(2);
}
else if (hdr.p4calc.operand_c == 0) {
send_back(3);
}
else if (hdr.p4calc.operand_d == 0) {
send_back(4);
}
else if (hdr.p4calc.operand_e == 0) {
send_back(5);
}
else if (hdr.p4calc.operand_f == 0) {
send_back(6);
}
else if (hdr.p4calc.operand_g == 0) {
send_back(7);
}
else if (hdr.p4calc.operand_h == 0) {
send_back(8);
}
else {
send_back(9);
}
```

I then had to add a code to check if the player has won.

```
/* Check if player 1 wins*/
if (hdr.p4calc.operand_a == 1 && hdr.p4calc.operand_b == 1 && hdr.p4calc.operand_c == 1) {
send_back(10);
}
else if (hdr.p4calc.operand_d == 1 && hdr.p4calc.operand_e ==1 && hdr.p4calc.operand_f == 1) {
send_back(10);
}
else if (hdr.p4calc.operand_g == 1 && hdr.p4calc.operand_h ==1 && hdr.p4calc.operand_i == 1) {
send_back(10);
}
else if (hdr.p4calc.operand_a == 1 && hdr.p4calc.operand_d ==1 && hdr.p4calc.operand_g == 1) {
send_back(10);
}
else if (hdr.p4calc.operand_b == 1 && hdr.p4calc.operand_e ==1 && hdr.p4calc.operand_h == 1) {
send_back(10);
}
else if (hdr.p4calc.operand_c == 1 && hdr.p4calc.operand_f ==1 && hdr.p4calc.operand_i == 1) {
send_back(10);
}
else if (hdr.p4calc.operand_a == 1 && hdr.p4calc.operand_e ==1 && hdr.p4calc.operand_i == 1) {
send_back(10);
}
else if (hdr.p4calc.operand_c == 1 && hdr.p4calc.operand_e ==1 && hdr.p4calc.operand_g == 1) {
send_back(10);
}
```

This sends back a result 10

```
elif str(p4calc.result) == '10':
    print('you win')
    break
```

So in the python I added this to show that you won.

I then added code to check if the computer could win in the next move and would return a win.

```
/*Check if player 2 wins*/
else if (hdr.p4calc.operand_a == 0 && hdr.p4calc.operand_b == 2 && hdr.p4calc.operand_c == 2) {
send_back(11);
}
else if (hdr.p4calc.operand_a == 2 && hdr.p4calc.operand_b == 0 && hdr.p4calc.operand_c == 2) {
send_back(11);
}
else if (hdr.p4calc.operand_a == 2 && hdr.p4calc.operand_b == 2 && hdr.p4calc.operand_c == 0) {
send_back(11);
}
else if (hdr.p4calc.operand_d == 0 && hdr.p4calc.operand_e ==2 && hdr.p4calc.operand_f == 2) {
send_back(11);
}
else if (hdr.p4calc.operand_d == 2 && hdr.p4calc.operand_e ==0 && hdr.p4calc.operand_f == 2) {
send_back(11);
}
else if (hdr.p4calc.operand_d == 2 && hdr.p4calc.operand_e ==2 && hdr.p4calc.operand_f == 0) {
send_back(11);
}
else if (hdr.p4calc.operand_g == 0 && hdr.p4calc.operand_h ==2 && hdr.p4calc.operand_i == 2) {
send_back(11);
}
else if (hdr.p4calc.operand_g == 2 && hdr.p4calc.operand_h ==0 && hdr.p4calc.operand_i == 2) {
send_back(11);
}
```

```
elif str(p4calc.result) == '11':
    print('you lose')
```

This now worked as a functional game, but it didn’t show the final board and it would just keep asking you for a move if it ended at a draw. To solve this I made new more streamlined versions of the code.

```
/*Check if player 2 can win in one move*/
else if (hdr.p4calc.operand_a == 0 && hdr.p4calc.operand_b == 2 && hdr.p4calc.operand_c == 2) {
hdr.p4calc.operand_a = 2;
send_back(11);
}
else if (hdr.p4calc.operand_a == 2 && hdr.p4calc.operand_b == 0 && hdr.p4calc.operand_c == 2) {
hdr.p4calc.operand_b = 2;
send_back(11);
}
```

It now also sends back with updated board numbers

```
/*if no available moves then it sends back a draw*/
else {
send_back(12);
}
```

I added in a draw result at the end if the computer couldn’t make a valid move.

```
/*Otherwise make random player 2 move*/
/*First choice is middle*/
else if (hdr.p4calc.operand_e == 0) {
hdr.p4calc.operand_e = 2;
send_back(5);
}
/*next choice is corners*/
else if (hdr.p4calc.operand_a == 0) {
hdr.p4calc.operand_a = 2;
send_back(1);
}
```

I also made it so the computer would pick moves starting with the middle and then the corners rather than just going from the top left in order.

```
resp = srp1(pkt, iface=iface,timeout=5, verbose=False)
if resp:
    p4calc=resp[P4calc]
    if p4calc:
    #replace board with new board received from the packet.
        place_a = p4calc.operand_a
        place_b = p4calc.operand_b
        place_c = p4calc.operand_c
        place_d = p4calc.operand_d
        place_e = p4calc.operand_e
        place_f = p4calc.operand_f
        place_g = p4calc.operand_g
        place_h = p4calc.operand_h
        place_i = p4calc.operand_i

        #check if received a game ending result
        if str(p4calc.result) == '10':
            print('you win')
            break
        elif str(p4calc.result) == '11':
            print('you lose')
            break
        elif str(p4calc.result) == '12':
            print('draw')
            break
```

For the python code I now made it so that the packet sends back the board and it updates all its values, and then checks if a gaming ending result was received.

Also made some quality of life changes. For example explaining the rules.

```
#explanation of how to play
print('On your turn type the position you want to play corresponding to these numbers:')
print('1 2 3')
print('4 5 6')
print('7 8 9')
print('your moves will be represented by 1 and the computer moves by 2')
```

And showing the board at the end

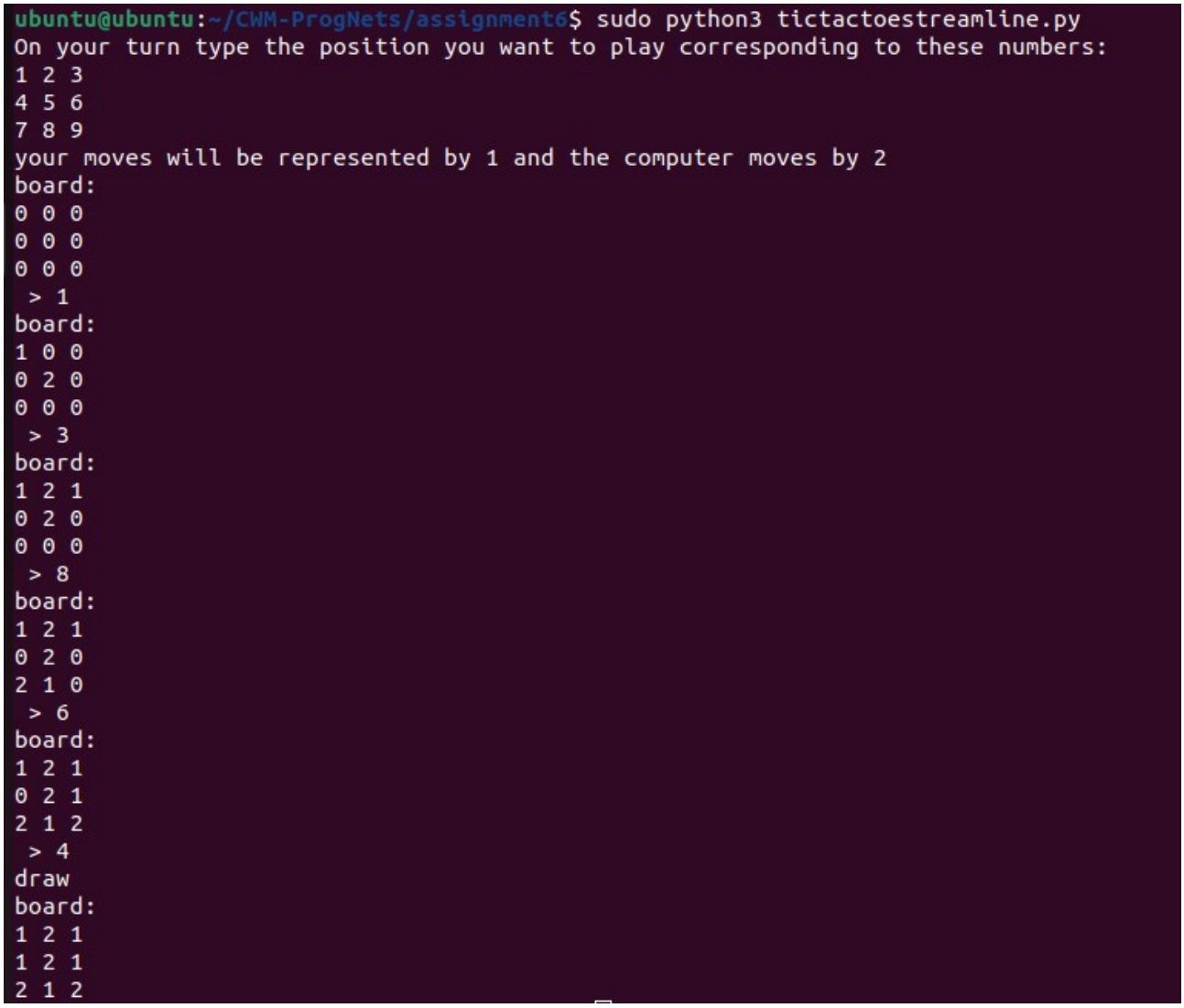
```
def display_board(a,b,c,d,e,f,g,h,i):
    print('board:')
    print(str(a)+' '+str(b)+' '+str(c))
    print(str(d)+' '+str(e)+' '+str(f))
    print(str(g)+' '+str(h)+' '+str(i))

#if it breaks out of the loop it shows the final board
display_board(place_a,place_b,place_c,place_d,place_e,place_f,place_g,place_h,place_i)
```

```
ubuntu@ubuntu:~/CWM-ProgNets/assignment6$ sudo python3 tictactoestreamline.py
On your turn type the position you want to play corresponding to these numbers:
1 2 3
4 5 6
7 8 9
your moves will be represented by 1 and the computer moves by 2
board:
0 0 0
0 0 0
0 0 0
 > 5
5
board:
2 0 0
0 1 0
0 0 0
 > 3
3
board:
2 0 1
0 1 0
2 0 0
 > 4
4
board:
2 0 1
1 1 0
2 0 2
 > 6
6
you win
board:
2 0 1
1 1 1
2 0 2
ubuntu@ubuntu:~/CWM-ProgNets/assignment6$
```

This is how the game ran in the streamlined version.

I then improved the computer's choice of move further by making it block if the user is one move away from winning

```
/*Check if player 1 can win in one move and blocks*/
else if (hdr.p4calc.operand_a == 0 && hdr.p4calc.operand_b == 1 && hdr.p4calc.operand_c == 1) {
hdr.p4calc.operand_a = 2;
send_back(1);
}
else if (hdr.p4calc.operand_a == 1 && hdr.p4calc.operand_b == 0 && hdr.p4calc.operand_c == 1) {
hdr.p4calc.operand_b = 2;
send_back(2);
}
else if (hdr.p4calc.operand_a == 1 && hdr.p4calc.operand_b == 1 && hdr.p4calc.operand_c == 0) {
hdr.p4calc.operand_c = 2;
send_back(3);
}
else if (hdr.p4calc.operand_d == 0 && hdr.p4calc.operand_e ==1 && hdr.p4calc.operand_f == 1) {
hdr.p4calc.operand_d = 2;
send_back(4);
}
```



The code now fully works to play tic tac toe.