```python
"""
============================== TicTacPro ==============================
FILE: achievements.py
MODIFIED: 07/12/2015
STATUS: Complete
FILE DESCRIPTION:
    The achievements.py file is used for loading and saving the achievements statistics
    to a binary file for a more permantant as it means the variables can be tranferred between instances of the game runnin
    rather than being local to each instance of the program being run.
USAGE:
    The game runs the achievem
"""

from main import *
from logic import *
import pickle

def achievereset():
    #resets the acheivements file
    f = open("achievements.pickle", "wb")
    stats = [0,0,0,0,0]
    pickle.dump(stats,f)
    f.close()

def achievements(res):
    #loads the stats from a local file
    f = open("achievements.pickle","rb")
    stats = pickle.load(f)
    f.close()
    #1-games played, 2-games won 3-games lost 4 - games draw 5 - level
    if res == "won":
        #ups the stats
        stats[0]+=1
        stats[1]+=1
        stats[4]+=1
        print("You are level " + str(stats[4]))
    elif res == "lost":
        if int(stats[4]) >1 :
            stats[0]+=1
            stats[2]+=1
            stats[4]-=0.5
            print("You are level "+ str(stats[4]))
        if int(stats[4]) <= 1:
            stats[0]+=1
            print("You are still on level 1!")
    elif res == "draw":
        stats[0]+=1
        stats[3]+=1
        stats[4]+=0.5
        print("You are level " + str(stats[4]))

    f = open("achievements.pickle","wb")
    pickle.dump(stats, f)
    f.close()

def achievementsview():
    #clears the last event and sets up the window for the viewing the achievements
    pygame.event.clear()
    screen=pygame.display.set_mode((610, 650))
    achievementsmenu=pygame.image.load("images/menu/settings/achievementsmenu.png")
    screen.blit(achievementsmenu,(0,0))
    #loads the achievements
    f = open("achievements.pickle","rb")
    stats = pickle.load(f)
    f.close()
    #prints them out in the corresponding locations
    print("Games played ....................................... " + str(stats[0]),70,140)
    print("Games won ........................................... " + str(stats[1]),70,200)
    print("Games lost ............................................ " + str(stats[2]),70,260)
    print("Games drawn ....................................... " + str(stats[3]),70,320)
    print("You are level " + str(stats[4]),200,380)

    q=False
    while not q:
        pygame.display.flip()
        ev = pygame.event.get()
        for event in ev:
            if event.type == pygame.MOUSEBUTTONUP:
                #checks for the back button being pressed
                pos = pygame.mouse.get_pos()
                if pos[0] in range(23,204) and pos[1] in range(540,600):
                    return None

if __name__=="__main__":
    achievements("won")
```

```
###-------------NEWFILE

from main import *
from logic import *
import random

def AIwinNext(used): #This code checks if the computer can win the next move
    for i in range(len(used)):
        if used[i] != "x" and used[i] != "o":
            oldi = used[i]
            used[i] = "o"
            isgamewoncheck = gamewon(used)
            used[i] = oldi
            if isgamewoncheck[0]:
                if oldi == 7:
                    return (10,10)
                elif oldi == 8:
                    return (210,10)
                elif oldi == 9:
                    return (410,10)
                elif oldi == 4:
                    return (10,210)
                elif oldi == 5:
                    return (210,210)
                elif oldi == 6:
                    return (410,210)
                elif oldi == 1:
                    return (10,410)
                elif oldi == 2:
                    return (210,410)
                elif oldi == 3:
                    return (410,410)
    return None

def AIblockNext(used):
    for i in range(len(used)):
        if used[i] != "x" and used[i] != "o":
            oldi = used[i]
            used[i] = "x"
            isgamewoncheck = gamewon(used)
            used[i] = oldi
            if isgamewoncheck[0]:
                if oldi == 7:
                    return (10,10)
                elif oldi == 8:
                    return (210,10)
                elif oldi == 9:
                    return (410,10)
                elif oldi == 4:
                    return (10,210)
                elif oldi == 5:
                    return (210,210)
                elif oldi == 6:
                    return (410,210)
                elif oldi == 1:
                    return (10,410)
                elif oldi == 2:
                    return (210,410)
                elif oldi == 3:
                    return (410,410)
    return None

def AItakeCorner(used):
    corners = [0, 2, 6, 8]
    random.shuffle(corners)
    for i in corners:
        if used[i] != "x" and used[i] != "o":
            if used[i] == 7:
                return (10,10)
            elif used[i] == 9:
                return (410,10)
            elif used[i] == 1:
                return (10,410)
            elif used[i] == 3:
                return (410,410)
    return None

def AItakeCentre(used):
    if used[4] != "x" and used[4] != "o":
        return (210,210)

def AItakeRandom(used):
    unused = []
    for i in used:
        if i != "x" and i != "o":
            unused.append(i)
    square = random.choice(unused)
    if square == 7:
        return (10,10)
    elif square == 8:
        return (210,10)
    elif square == 9:
        return (410,10)
    elif square == 4:
        return (10,210)
    elif square == 5:
        return (210,210)
    elif square == 6:
        return (410,210)
    elif square == 1:
        return (10,410)
    elif square == 2:
        return (210,410)
    elif square == 3:
        return (410,410)

###-------------NEWFILE

"""
============================== TicTacPro ==============================
FILE: chat.py
MODIFIED: 27/11/2015
STATUS: Complete
FILE DESCRIPTION:
    The chat.py file contains all the functions relating to displaying the chat and
    input and the chat list.

USAGE:
This is a client chat display and input management function.
"""
from main import *
from logic import *

def main():
    screen=pygame.display.set_mode((900, 650))
    pygame.draw.rect(screen, (0,0,0), [610, 0, 300, 700], 0)
    chat=[]
    displaystring=""
    turn=True
    while True:
        #how i get mouse clicks this is event driven programming
        ev = pygame.event.get()
        for event in ev:
            if event.type == pygame.QUIT:
                raise SystemExit
            elif event.type == pygame.KEYDOWN:
                displaystring,chat=chatinput(event.key,displaystring,chat)

def chatinput(eventkey,displaystring,chat):
    #this function runs when a key is presed, and decides what to do with it

    if eventkey in range(97,123) or eventkey in range(48,58):
        #these ranges are a-z and 0-9, as it is the ascii value it is simple to
        #convert the number to a letter or number
        displaystring+=str(chr(eventkey))

    elif eventkey == 13:
        #when the enter key is pressed, sends the message to the server and also
        #it clears the display string and adds it to the chat list, then draws the chat
        if displaystring!="":
            chat.insert(0,("You",displaystring))
            sendmsg=displaystring
        displaystring=""
        drawlist(chat)

    elif eventkey == 32:
        #whhen the space key is pressed a blank space in inserted into the displaystring
        displaystring+=" "

    elif eventkey == 8:
        #when the backspace key is pressed the last letter in the displaystring is removed
        displaystring=displaystring[:-1]

    #display the current input message
    showinput(displaystring)
    return displaystring,chat

def showinput(string):
    string=(": "+string)
    #covers up any previous stuff on the screen
    pygame.draw.rect(screen, (0,0,0), [610, 600, 300, 700], 0)
    #draws texts on screen
    font = pygame.font.Font(None, 16)
    text = font.render(string, 4, (255, 255, 255))
    screen.blit(text, (620,610))
    #updates screen to show new changes
    pygame.display.flip()

def drawlist(chat):
    #set the bottom height of the screen
    y=540
    #deletes the oldest message in the chat after it gets to 14 messages
    if len(chat)>14: del chat[-1]
    pygame.draw.rect(screen, (0,0,0), [610, 0, 300, 700], 0)
    for each in chat:
        #generates the output string
        each=(str(each[0]) +": " + str(each[1]))
        #decides how to handle printing out the statement, if it will fit on one line or not
        if len(each)<47:
            #prints the string out at the y coord
            font = pygame.font.Font(None, 16)
            chatstuff = font.render(each, 4, (255, 255, 255))
            screen.blit(chatstuff, (620,y))
            #moves the y coord down to accomodate the next message
            y-=40

        elif len(each)>120:
            #word limit
            print("Too long to display!")

        else:
            #smaller recursive loop like the one above for multi line messages
            y2=y
            y-=40
            while len(each)!=0:
                #takes one lines worth of code from the each string
                line=each[:40]
                each=each[40:]
                font = pygame.font.Font(None, 16)
                chatstuff = font.render(line, 4, (255, 255, 255))
                screen.blit(chatstuff, (620,y2))
                #moves it down slightly lower to give a multiline effect
                y2+=10

        pygame.display.flip()
if __name__=="__main__":
    main()



###-------------NEWFILE

"""
============================== TicTacPro ==============================
FILE: firstgo.py
MODIFIED: 07/12/2015
STATUS: Complete
FILE DESCRIPTION:
 The firstgo.py file is used for deciding who the first person to make a move
 is by asking an educational question. It is used only for offline single-player
 games and online games.
"""

from main import *
from logic import *
import random

def askquestion():
    questions = [
["Who is now the American President?","Barrack Obama", "Sean Paul", "Michael Jordan", "Skepta", 1],
["How many cheeks do you have?","2", "0", "3", "4", 4],
["What goes up and never comes down?","Age", "Football", "Plane", "Rain", 1],
["What is the square root of 144 equal?","12.5", "12", "1", "2", 2],
["Who made Microsoft?","Bill Smith", "Bill Phil", "Bill Gates", "Bill Paul", 3],
["What is half of 200?","150", "120", "100", "102", 3],
["Which of the following is a soap (TV Programme)?","Power Rangers", "Eastenders", "BBC News", "MTV Base", 2],
["How many legs does a spider have?","4", "6", "8", "10", 3],
["What is the tallest animal in the world?","The giraffe", "Crocodile", "Bear", "Fox", 1]
                  ] # Structure of list: 0 - question, 1 - option 1, 2 - option 2, 3 - option 3, 4 - option 4, 5 - index of
    question = random.choice(questions)
    pygame.event.clear() # Remove all events in list. Prevents conflicts.
    screen=pygame.display.set_mode((610, 650))
    questionmenu=pygame.image.load("images/menu/questionmenu.png")
    screen.blit(questionmenu,(0,0))
    print(question[0],80,100) # Display question.
    print(question[1],100,300) # Display answers.
    print(question[2],350,300)
    print(question[3],110,440)
    print(question[4],350,440)
    q=False
    choice = 0
    while not q:
        pygame.display.flip()
        ev = pygame.event.get()
        for event in ev:
            if event.type == pygame.MOUSEBUTTONUP:

                pos = pygame.mouse.get_pos()

                if pos[0] in range(85,275) and pos[1] in range(275,365): # Detect which answer is selected.
                    choice=1

                elif pos[0] in range(335,275) and pos[1] in range(525,365):
                    choice=2

                elif pos[0] in range(85,410) and pos[1] in range(275,500):
                    choice=3

                elif pos[0] in range(308,410) and pos[1] in range(525,500):
                    choice=4

                if choice:
                 if choice==question[5]: # Compare the user's answer to the correct answer.
                     print("Correct! You go first!")
                     time.sleep(2)
                     return True
                 else:
                     print("That's not correct! You go second.")
                     time.sleep(2)
                     return False
            elif event.type == pygame.QUIT:
                quitgame()

if __name__=="__main__":
    askquestion()


###-------------NEWFILE

"""
============================== TicTacPro ==============================
FILE: logic.py
MODIFIED: 08/12/2015
STATUS: Complete
FILE DESCRIPTION:
    The logic.py file is responsible for the game logic and behavour. It includes
    functions used to draw the board, validate and draw player moves, determine game
    outcome and quit the game.
USAGE:
    This file is not meant to be run independently. It serves as the codebase for the
    aforementioned functions and is used by nearly all other game files. Should the file
    be run on its own, it will prompt the user to run the TicTacProLauncher.py file.
"""

from main import *
from settings import *

def quitgame():
    """
    FUNCTION NAME: quitgame()
    PARAMETERS: 0
    FUNCTION DESCRIPTION:
        Excecutes the correct procedure for closing the program.
    """
    pygame.quit()
    sys.exit()

def gamewon(used):
    """
    FUNCTION NAME: gamewon()
    PARAMETERS: 1
            used (list; mandatory): the list representation of the board, containing either numbers 1-9, "x" or "o"
    FUNCTION DESCRIPTION:
        Decides whether the game is won taking in the used table which indicates the current game state
        and checking if there are three marks of the same kind in a row. The function return a tuple
        containing a boolean, corresponding to the state of the game, as its first value and an integer,
        corresponding to the win combination, as the second.
        """
    if used[0]==used[1] and used[1]==used[2]: return (True,1)
    elif used[3]==used[4] and used[4]==used[5]: return (True,2)
    elif used[6]==used[7] and used[7]==used[8]: return (True,3)
    elif used[0]==used[3] and used[3]==used[6]: return (True,4)
```

```python
    elif used[1]==used[4] and used[4]==used[7]: return (True,5)
    elif used[2]==used[5] and used[5]==used[8]: return (True,6)
    elif used[0]==used[4] and used[4]==used[8]: return (True,7)
    elif used[2]==used[4] and used[4]==used[6]: return (True,8)
    else: return (False,999)

def validmove(pos,used):
    """
    FUNCTION NAME: validmove()
    PARAMETERS: 2
                pos (tuple; mandatory): a tuple containing the coordinate of the user mouse click (0-620, 0-690)
                used (list; mandatory): the list representation of the board, containing either numbers 1-9, "x" or "o"
    FUNCTION DESCRIPTION:
        Checks the mouse position coordinates against the board "hitboxes" and returns a boolean
        defining whether the click was in a valid location of a box.
    """
    rickcheck(pos) # Rickageddon check.

    if pos[0] in range(10,200) and pos[1] in range(10,200): sqr=0
    elif pos[0] in range(210,400) and pos[1] in range(10,200): sqr=1
    elif pos[0] in range(410,600) and pos[1] in range(10,200): sqr=2
    elif pos[0] in range(10,200) and pos[1] in range(210,400): sqr=3
    elif pos[0] in range(210,400) and pos[1] in range(210,400): sqr=4
    elif pos[0] in range(410,600) and pos[1] in range(210,400): sqr=5
    elif pos[0] in range(10,200) and pos[1] in range(410,600): sqr=6
    elif pos[0] in range(210,400) and pos[1] in range(410,600): sqr=7
    elif pos[0] in range(410,600) and pos[1] in range(410,600): sqr=8
    else:
        return False

    if used[sqr] not in ["o","x"]:
            return True
    else:
        return False

def rickcheck(pos):
    """
    FUNCTION NAME: rickcheck()
    PARAMETERS: 1
                pos (tuple; mandatory): a tuple containing the coordinate of the user mouse click (0-620, 0-690)
    FUNCTION DESCRIPTION:
        Defines a super special move that rolls the screen!
    """
    if pos[0] in range(580,610) and pos[1] in range(610,640):
        cena=pygame.image.load("images/misc/cena.png")
        pygame.image.save(screen,"images/misc/temp.png")
        pygame.mixer.music.load("music/special.mp3")
        pygame.mixer.music.play(-1)
        posit=(-200)
        num=10
        while num<40:
            if posit>500:
                posit=(-200)
            screen.blit(rick,(posit,-50))
            screen.blit(rick,(posit+num,50))
            screen.blit(rick,(posit+num*2,150))
            screen.blit(rick,(posit+num*3,250))
            screen.blit(rick,(posit+num*4,350))
            if num>15:
                screen.blit(rick,(posit,-50))
                screen.blit(rick,(50,posit+num))
                screen.blit(rick,(150,posit+num*2))
                screen.blit(rick,(250,posit+num*3))
                screen.blit(rick,(350,posit+num*4))
            pygame.display.flip()
            num+=0.5
            posit+=20
            time.sleep(0.2)
            ev = pygame.event.get()
            for event in ev:
                if event.type == pygame.MOUSEBUTTONUP:
                    pos = pygame.mouse.get_pos()
                    screen.blit(rick,(pos[0]-50,pos[1]-50))
                    pygame.display.flip()
        pygame.mixer.music.pause()
        random.play()
        time.sleep(1.6)
        screen.blit(cena,(-350,-50))
        pygame.display.flip()
        time.sleep(4.4)
        pygame.mixer.music.unpause()

        temp=pygame.image.load("images/misc/temp.png")
        screen.blit(pygame.image.load("images/misc/temp.png"),(0,0))
        pygame.display.flip()
```

```python
def drawbox(turn,pos,used,images):
    """
    FUNCTION NAME: drawbox()
    PARAMETERS: 4
                turn (boolean; mandatory): a boolean value that determines whose turn it is; True for "x"; False for "o"
                pos (tuple; mandatory): a tuple containing the coordinate of a mouse click (0-620, 0-690)
                used (list; mandatory): the list representation of the board, containing either numbers 1-9, "x" or "o"
                images (list; mandatory): a list that stores the game's current image set for the board to use
    FUNCTION DESCRIPTION:
        Draws the game board and redraws it every time it is generated. It can handle different
        image sets using the images list passed in as a parameter.
    """
    if pos[0] in range(10,200) and pos[1] in range(10,200): # Top left box (7).
        if turn:
            screen.blit(images[1],(10,10))
            used[0]="x"
        else:
            screen.blit(images[2],(10,10))
            used[0]="o"
    elif pos[0] in range(210,400) and pos[1] in range(10,200): # Top center box (8).
        if turn:
            screen.blit(images[1],(210,10))
            used[1]="x"
        else:
            screen.blit(images[2],(210,10))
            used[1]="o"
    elif pos[0] in range(410,600) and pos[1] in range(10,200): # Top right box (9).
        if turn:
            screen.blit(images[1],(410,10))
            used[2]="x"
        else:
            screen.blit(images[2],(410,10))
            used[2]="o"
    elif pos[0] in range(10,200) and pos[1] in range(210,400): # Middle left box (4).
        if turn:
            screen.blit(images[1],(10,210))
            used[3]="x"
        else:
            screen.blit(images[2],(10,210))
            used[3]="o"
    elif pos[0] in range(210,400) and pos[1] in range(210,400): # Middle center box (5).
        if turn:
            screen.blit(images[1],(210,210))
            used[4]="x"
        else:
            screen.blit(images[2],(210,210))
            used[4]="o"
    elif pos[0] in range(410,600) and pos[1] in range(210,400): # Middle right box (6).
        if turn:
            screen.blit(images[1],(410,210))
            used[5]="x"
        else:
            screen.blit(images[2],(410,210))
            used[5]="o"
    elif pos[0] in range(10,200) and pos[1] in range(410,600): # Bottom left box (1).
        if turn:
            screen.blit(images[1],(10,410))
            used[6]="x"
        else:
            screen.blit(images[2],(10,410))
            used[6]="o"
    elif pos[0] in range(210,400) and pos[1] in range(410,600): # Bottom center box (2).
        if turn:
            screen.blit(images[1],(210,410))
            used[7]="x"
        else:
            screen.blit(images[2],(210,410))
            used[7]="o"
    elif pos[0] in range(410,600) and pos[1] in range(410,600): # Bottom right box (3).
        if turn:
            screen.blit(images[1],(410,410))
            used[8]="x"
        else:
            screen.blit(images[2],(410,410))
            used[8]="o"

    turn=not turn
    pygame.display.flip()
    return used

def drawline(wincombo,turn,images):
    """
    FUNCTION NAME: drawline()
    PARAMETERS: 3
                wincombo (integer; mandatory): an integer that represents what combination has been achieved
                turn (boolean; mandatory): a boolean value that determines whose turn it is; True for "x"; False for "o"
            images (list; mandatory): a list that stores the game's current image set for the board to use
    FUNCTION DESCRIPTION:
        When the game is won, this function creates the effect of the flashing winning boxes.
    """
    xw=images[3]
    ow=images[4]
    x=images[1]
    o=images[2]
    for count in range(0,6):    # Creates the flashing effect according to the win pattern.
        if count/2==count//2:
            winsound.play()
            if wincombo==1:
                if turn:screen.blit(xw,(10,10)),screen.blit(xw,(210,10)),screen.blit(xw,(410,10))
                else:screen.blit(ow,(10,10)),screen.blit(ow,(210,10)),screen.blit(ow,(410,10))

            elif wincombo==2:
                if turn:screen.blit(xw,(10,210)),screen.blit(xw,(210,210)),screen.blit(xw,(410,210))
                else:screen.blit(ow,(10,210)),screen.blit(ow,(210,210)),screen.blit(ow,(410,210))

            elif wincombo==3:
                if turn:screen.blit(xw,(10,410)),screen.blit(xw,(210,410)),screen.blit(xw,(410,410))
                else:screen.blit(ow,(10,410)),screen.blit(ow,(210,410)),screen.blit(ow,(410,410))

            elif wincombo==4:
                if turn:screen.blit(xw,(10,10)),screen.blit(xw,(10,210)),screen.blit(xw,(10,410))
                else:screen.blit(ow,(10,10)),screen.blit(ow,(10,210)),screen.blit(ow,(10,410))

            elif wincombo==5:
                if turn:screen.blit(xw,(210,10)),screen.blit(xw,(210,210)),screen.blit(xw,(210,410))
                else:screen.blit(ow,(210,10)),screen.blit(ow,(210,210)),screen.blit(ow,(210,410))

            elif wincombo==6:
                if turn:screen.blit(xw,(410,10)),screen.blit(xw,(410,210)),screen.blit(xw,(410,410))
                else:screen.blit(ow,(410,10)),screen.blit(ow,(410,210)),screen.blit(ow,(410,410))

            elif wincombo==7:
                if turn:screen.blit(xw,(10,10)),screen.blit(xw,(210,210)),screen.blit(xw,(410,410))
                else:screen.blit(ow,(10,10)),screen.blit(ow,(210,210)),screen.blit(ow,(410,410))

            elif wincombo==8:
                if turn:screen.blit(xw,(410,10)),screen.blit(xw,(210,210)),screen.blit(xw,(10,410))
                else:screen.blit(ow,(410,10)),screen.blit(ow,(210,210)),screen.blit(ow,(10,410))
        else:
            if wincombo==1:
                if turn:screen.blit(x,(10,10)),screen.blit(x,(210,10)),screen.blit(x,(410,10))
                else:screen.blit(o,(10,10)),screen.blit(o,(210,10)),screen.blit(o,(410,10))

            elif wincombo==2:
                if turn:screen.blit(x,(10,210)),screen.blit(x,(210,210)),screen.blit(x,(410,210))
                else:screen.blit(o,(10,210)),screen.blit(o,(210,210)),screen.blit(o,(410,210))

            elif wincombo==3:
                if turn:screen.blit(x,(10,410)),screen.blit(x,(210,410)),screen.blit(x,(410,410))
                else:screen.blit(o,(10,410)),screen.blit(o,(210,410)),screen.blit(o,(410,410))

            elif wincombo==4:
                if turn:screen.blit(x,(10,10)),screen.blit(x,(10,210)),screen.blit(x,(10,410))
                else:screen.blit(o,(10,10)),screen.blit(o,(10,210)),screen.blit(o,(10,410))

            elif wincombo==5:
                if turn:screen.blit(x,(210,10)),screen.blit(x,(210,210)),screen.blit(x,(210,410))
                else:screen.blit(o,(210,10)),screen.blit(o,(210,210)),screen.blit(o,(210,410))

            elif wincombo==6:
                if turn:screen.blit(x,(410,10)),screen.blit(x,(410,210)),screen.blit(x,(410,410))
                else:screen.blit(o,(410,10)),screen.blit(o,(410,210)),screen.blit(o,(410,410))

            elif wincombo==7:
                if turn:screen.blit(x,(10,10)),screen.blit(x,(210,210)),screen.blit(x,(410,410))
                else:screen.blit(o,(10,10)),screen.blit(o,(210,210)),screen.blit(o,(410,410))

            elif wincombo==8:
                if turn:screen.blit(x,(410,10)),screen.blit(x,(210,210)),screen.blit(x,(10,410))
                else:screen.blit(o,(410,10)),screen.blit(o,(210,210)),screen.blit(o,(10,410))
        pygame.display.flip()
        time.sleep(0.5)

if __name__=="__main__":
    print("Please launch the game from the TicTacProLauncher.py file.")


###-------------NEWFILE

"""
============================== TicTacPro ==============================
FILE: Main.py
```

```
MODIFIED: 15/11/2015
STATUS: Complete
FILE DESCRIPTION:
The Main.py file is the central state for the game, initilises the game and
all it's images and features, it also launches the splash screen when booted.
"""

#installing all the modules required for this game
try:
    module="Time"
    import time
    module="Sys"
    import sys
    module="Pygame"
    import pygame
    module="Math"
    import math
    module="os"
    import os
#if none of these work then it will throw an error and tell you what you need
except:
    print("Error - You don't have the required modules installed!")
    print("Please install Module '{0}'! ".format(module))
    for count in range(0,50000000):
        pass
    raise SystemExit

pygame.init()
#game wide initalistion, starts all the relevant aspects to the game and loads the standard images
pygame.display.set_icon(pygame.image.load("images/misc/icon.png"))
screen=pygame.display.set_mode((610, 650))
clock = pygame.time.Clock()
images=[
pygame.image.load("images/classic/board.png"),
pygame.image.load('images/classic/x.png'),
pygame.image.load('images/classic/o.png'),
pygame.image.load("images/classic/xwon.png"),
pygame.image.load("images/classic/owin.png")
]
rick=pygame.image.load("images/misc/rick.png")
pygame.display.set_caption("TicTacPro Game","TicTacPro")
#all sound files from sounddogs.com royalty free and some editied by me
pygame.mixer.music.load("music/harder.mp3")
clicksound=pygame.mixer.Sound("music/fx/click.wav")
winsound=pygame.mixer.Sound("music/fx/win.wav")
losersound=pygame.mixer.Sound("music/fx/loser.wav")
nosound=pygame.mixer.Sound("music/fx/no.wav")
random=pygame.mixer.Sound("music/fx/random.wav")
mainmenuimg=pygame.image.load("images/menu/mainmenu.png")

#imports each file so the functions can be called and run in this program
try:
    from offline2p import *
    from offline1p import *
    from online import *
    from achievements import *
    from settings import *
    from firstgo import *
    from settings import *
    from logic import *
#if any files are missing will let you know what is missing
except ImportError:
    print("Error - You're missing game files!")
    print("Please download zip file again!")
    for count in range(0,50000000):
        pass
    raise SystemExit

def mainmenu(images, host="no", port=0):
    """Runs the main menu, it opens the main menu, and allows you to access the rest of the game from here"""
    q = False
    difficulty="medium"
    while not q:
        pygame.event.clear()
        screen=pygame.display.set_mode((610, 650))
        screen.blit(mainmenuimg,(0,0))
        pygame.draw.rect(screen, (64,0,64), (0,610,650,50), 0)
        pygame.display.flip()
        ev = pygame.event.get()
        #if an event happens such as a keypress or mouse click it will run through this code
        for event in ev:
            if event.type == pygame.QUIT:
                quitgame()
            if event.type == pygame.MOUSEBUTTONUP:
                pos = pygame.mouse.get_pos()
                pygame.display.flip()
                #checks whether the mouse click was on a button, which i have defined by coordinates
                if pos[0] in range(137,560) and pos[1] in range(132,180):
                    if  host == "no" and port == 0:
                        print("Online game play is disabled.")
                        time.sleep(1)
                    else:
                        online(images, host, port)
                elif pos[0] in range(50,485) and pos[1] in range(210,275):
                    #runs the code that asks who goes first
                    whosturn=askquestion()
                    #launches the offline 1p state
                    offline1p(difficulty,whosturn,images)

                elif pos[0] in range(50,485) and pos[1] in range(290,350):
                    offline2p(images)

                elif pos[0] in range(50,485) and pos[1] in range(370,430):
                    possimages=settingmenu()
                    #if the stlye is changed then this code will run, changing what images the program uses
                    if possimages[0]!=None:
                        difficulty=possimages[0]
                    elif possimages[1]!=None:
                        images=possimages[1]
                elif pos[0] in range(50,485) and pos[1] in range(450,515):
                    achievementsview()

                elif pos[0] in range(540,595) and pos[1] in range(10,50):
                    #quits the game and leaves the loop
                    quitgame()
                    q=True

if __name__=="__main__":
    #this generates the splash screen for the program, this is run after the loading code as splash screens are traditional
    pygame.display.set_icon(pygame.image.load("images/misc/icon.png"))
    os.environ['SDL_VIDEO_CENTERED'] = '1'
    sega=pygame.mixer.Sound("music/fx/sega.wav")
    sega.play()
    screen = pygame.display.set_mode((200,200),pygame.NOFRAME)
    pygame.Surface([640,480], pygame.SRCALPHA, 32)
    splash=pygame.image.load("images/misc/splash.png")
    screen.blit(splash,(-30,-200))
    pygame.display.flip()
    time.sleep(2)

    pygame.display.set_icon(pygame.image.load("images/misc/icon.png"))
    screen=pygame.display.set_mode((610, 650))
    if len(sys.argv) > 1: # If the game was started through the launcher, pass the host and port from the arguments variabl
        mainmenu(images, sys.argv[1], int(sys.argv[2]))
    else: # If the game was started by opening the main.py file, show the following message and exit the game.
        #print("Start the game using the TicTacProLauncher.py file.")
        #debug
        mainmenu(images, "no", 0)
        time.sleep(5)


###-------------NEWFILE

"""
============================== TicTacPro ==============================
FILE: offline1p.py
MODIFIED: 07/12/2015
STATUS: Complete
FILE DESCRIPTION:
The offline1p.py file is the one-player state that uses AI for deciding where
the computer should play according to the game difficulty.
"""

from main import *
from logic import *
from achievements import *
from AI import *
import random, time

def offline1p(difficulty, turn, images):
    """
    FUNCTION NAME: offline1p
    PARAMETERS: 3
                difficulty (string; mandatory): determines the gameplay of the computer using AI; the allowed values are "e
                turn (boolean; mandatory): determines who makes the first move; if True, the user goes first; if False, the
                images (list; mandatory): passes in the list of images used to depict the board; this list is determined by
    FUNCTION DESCRIPTION:
        This is the main function that runs the offline single-player state of the game.
        It begins by defining the game window, drawing the empty game board and playing
        the background music. It then runs the main loop which is responsible for giving
        turns to the two players, validating those moves and deciding whether the game
        has been won or not.
        For the computer's moves, the loop goes through a sequence of functions that decide
        the most appropriate move according to the difficulty level. Look at AI.py for more
        details.
    """
    screen=pygame.display.set_mode((610, 650))
    pygame.mixer.music.play(-1)
    screen.blit(images[0],(0,0))
    used=[7,8,9,4,5,6,1,2,3]    # Defines the list of squares used to hold the players' moves.
    pygame.display.flip()
    count=0 # Initiates a count of the moves made in this instance of the game.
    isgamewon=(False,9)
    while count<9 and not isgamewon[0]: # Main loop.
        ev = pygame.event.get()
        for event in ev:
            print("")   # If there is an old message in the status bar, remove it.
            if turn:    # User's move.
                if event.type == pygame.MOUSEBUTTONUP:
                    pos = pygame.mouse.get_pos()
                    clicksound.play()
                    valid=validmove(pos,used)
                    if valid:
                        drawbox(turn,pos,used,images)
                        isgamewon=gamewon(used)
                        count+=1
                        if isgamewon[0]:
                            drawline(isgamewon[1],turn,images)
                            break
                        turn=not turn
                    else:
                        print("That is not a valid move.")
                        nosound.play()
            elif not turn:  # Computer's move.
                if difficulty == "easy":
                    pos = AItakeRandom(used)

                elif difficulty == "medium":
                    pos = None
                    while pos is None:
                        pos = AIwinNext(used)
                        if pos is not None:
                            break
                        pos = AIblockNext(used)
                        if pos is not None:
                            break
                        pos = AItakeRandom(used)

                elif difficulty == "hard":
                    pos = None
                    while pos is None:
                        pos = AIwinNext(used)
                        if pos is not None:
                            break
                        pos = AIblockNext(used)
                        if pos is not None:
                            break
                        pos = AItakeCorner(used)
                        if pos is not None:
                            break
                        pos = AItakeCentre(used)
                        if pos is not None:
                            break
                        pos = AItakeRandom(used)

                valid=validmove(pos,used)
                if valid:
                    clicksound.play()
                    drawbox(turn,pos,used,images)
                    isgamewon=gamewon(used)
                    count+=1
                    if isgamewon[0]:
                        drawline(isgamewon[1],turn,images)
                        break
                    turn=not turn
                else:
                    pass

            if event.type == pygame.QUIT:   # Allows closing the game window with the X button.
                quitgame()

    if isgamewon[0]:
        if turn:
            winner="1"
            print("The winner is Player {0}!".format(winner))
            time.sleep(2)
            achievements("won")
        else:
            winner="2"
            print("The winner is Player {0}!".format(winner))
            time.sleep(2)
            achievements("lost")
    else:
        losersound.play()
        print("The game is a draw!")
        time.sleep(2)
        achievements("draw")
    pygame.mixer.music.fadeout(2000)
    time.sleep(2)

if __name__=="__main__":
    offline1p("hard", True, images)


###-------------NEWFILE

"""
============================== TicTacPro ==============================
FILE: Offline2p.py
MODIFIED: 15/11/2015
STATUS: Complete
FILE DESCRIPTION:
the Offline2p.py file is the 2 player offline state of the Tic Tac Toe game,
allowing 2 players to play against offline on a single system.
"""

from main import *
from logic import *

def offline2p(images):
    pygame.event.clear()
    """runs the offline 2 player iteration of the game"""
    #sets up the screen for running the offline2p gamemode
    screen=pygame.display.set_mode((610, 650))
    pygame.mixer.music.play(-1)
    screen.blit(images[0],(0,0))
    used=[7,8,9,4,5,6,1,2,3]
    pygame.display.flip()
    count=0
    turn = True
    isgamewon=(False,9)
    while count<9 and not isgamewon[0]:
        if turn:
            print("It is Player 1's turn.")
        else:
            print("It is Player 2's turn.")
        ev = pygame.event.get()
        for event in ev:
            #checks where you click
            if event.type == pygame.MOUSEBUTTONDOWN:
                pos = pygame.mouse.get_pos()
                valid=validmove(pos,used)
                if valid:
                    #if the input is in a valid position then it will run this code
                    clicksound.play()
                    drawbox(turn,pos,used,images)
                    isgamewon=gamewon(used)
                    count+=1
                    #isgamewon is a tuple (bool,string of "player 1" or 2"
                    if isgamewon[0]:
                        drawline(isgamewon[1],turn,images)
                        break
                    turn=not turn
                else:
                    nosound.play()
                    print("That is not a valid move.")
            elif event.type == pygame.QUIT:
                quitgame()
    #this runs when the game is over, either the max number of moves have happened
    #or someone has won
    if isgamewon[0]:
        #works out who won, runs the corresponding code
        if turn:
            winner = "1"
            print("The winner is Player {0}!".format(winner))
        else:
            winner = "2"
            print("The winner is Player {0}!".format(winner))
    else:
        losersound.play()
        print("The game is a draw!")
        time.sleep(2)
        achievements("draw")
    pygame.mixer.music.fadeout(2000)
    time.sleep(2)

if __name__=="__main__":
    offline2p(images)


###-------------NEWFILE

"""
============================== TicTacPro ==============================
FILE: Online.py
MODIFIED: 15/11/2015
STATUS: Complete
FILE DESCRIPTION:
the Online.py file is the online version of the tictactoe game, it allows
communication between 2 machines through a server, which sends commands for
not only board positions but also the chat, which allows text communication
between the 2 clients


includes chat
"""
from main import *
from logic import *
from chat import *
from firstgo import *
import socket, select, pickle, logging

class GameClient():
    def __init__(self, host="localhost", port=12341):
        self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.client.connect((host, port))
        self.messages = []
        self.recvmessages = []

    def shutdown(self):
        self.client.shutdown(1)
        self.client.close()

    def send_message(self, msg):
        msg = pickle.dumps(msg)
        self.messages.append(msg)

    def recv_message(self):
        if self.recvmessages:
            return pickle.loads(self.recvmessages.pop(0))

    def poll(self):
        read, write, error = select.select ( [self.client], [self.client], [self.client], 0 )

        for conn in read:
            newmsg = self.client.recv(1024)
            if newmsg:
                self.recvmessages.append(newmsg)

        for conn in write:
            for i in self.messages:
                conn.send(i)
                self.messages.remove(i)

def online(images, host, port):
    used = [7, 8, 9, 4, 5, 6, 1, 2, 3]
    count = 0
    isgamewon = (False, 9)
    chat=[]
    displaystring=""
    try:
        gamecli = GameClient(host, port)

        myturn = None
        while True:
            if myturn is None:
                myturn = askquestion()
                gamecli.send_message(myturn)
            gamecli.poll() # Send and receive messages between opponents.
            newmsg = gamecli.recv_message()
            if myturn is not None and newmsg is not None:
                if myturn == newmsg:
                    print("Both players answered correctly. Try again.")
                    time.sleep(2)
                    myturn = None
                else:
                    turn = myturn
                    break
        screen = pygame.display.set_mode((900, 650))
        pygame.mixer.music.play(-1)
        screen.blit(images[0], (0, 0))
        pygame.display.flip()

        while True:
            ev = pygame.event.get()
            for event in ev:
                if turn and event.type == pygame.MOUSEBUTTONUP:
                    pos = pygame.mouse.get_pos()
                    clicksound.play()
                    valid = validmove(pos, used)
                    if valid:
                        gamecli.send_message(pos)
                        drawbox(turn, pos, used, images)
                        isgamewon = gamewon(used)
                        count += 1
                        if isgamewon[0]:
                            drawline(isgamewon[1], turn, images)
                            break
                        turn = not turn
                    else:
                        print("That is not a valid move.")
                        nosound.play()

                elif event.type == pygame.KEYDOWN:
                    if event.key == 13:
                        if displaystring!="":
                            chat.insert(0,("You",displaystring))
                            gamecli.send_message(displaystring)
                        displaystring=""
                        drawlist(chat)
                    else:
                        displaystring,chat=chatinput(event.key,displaystring,chat)

                elif event.type == pygame.QUIT:
                    quitgame()

            gamecli.poll() # Send and receive messages between opponents.
            newmsg = gamecli.recv_message()

            if not turn and isinstance(newmsg, tuple):
                drawbox(turn, newmsg, used, images)
                clicksound.play()
                isgamewon = gamewon(used)
                count += 1
                if isgamewon[0]:
                    drawline(isgamewon[1], turn, images)
                    break
                turn = not turn

            if isinstance(newmsg, str):
                chat.insert(0,("Opponent",newmsg))
                drawlist(chat)
            if count == 9:
                break

        if isgamewon:
            if turn:
                print("You won! Congratulations!")
                time.sleep(2)
                achievements("won")
            else:
                print("Better luck next time!")
                time.sleep(2)
                achievements("lost")
        else:
            losersound.play()
            print("The game is a draw!")
            time.sleep(2)
            achievements("draw")
        pygame.mixer.music.fadeout(2000)
        time.sleep(2)
    finally:
        gamecli.shutdown()

if __name__=="__main__":
    online(images, "localhost", 12341)


###-------------NEWFILE

"""
============================== TicTacPro ==============================
FILE: server.py
MODIFIED: 23/11/2015
STATUS: debug
FILE DESCRIPTION:
    The server.py file adds the functionality of playing multiplayer online games on a local network.
    Events during the execution of the file will be logged to events.log.
USAGE:
    The server.py file has to reside on the server computer to which clients will connect.
"""

import socket, select, logging

class servergo():
    """
    CLASS NAME: servergo()
    CLASS DESCRIPTION:
        Initializes the server needed to host an online multiplayer game of TicTacPro. Makes use of a
        TCP/IP sockets for the client-server connection. This server acts as a bridge between the
        two clients, i.e. sends information from one to the other. No actions are performed on the data
        while travelling to and from the server.
    """

    def __init__(self, ipaddr = "localhost", port = 12341):
        """
        FUNCTION NAME: __init__
        PARAMETERS: 2
                    ipaddr (string; optional; defaults to "localhost"): the server IP address which will host client connec
                    port (integer; optional; defaults to 12341): the socket port which clients will connect to
        FUNCTION DESCRIPTION:
            Initializes the server connection with the default parameters unless others are provided.
        """
        self.connections = []   # Declare a list that will hold all connections.
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Create an IPv4 TCP socket.
        self.server.bind((ipaddr, port))    # Bind the socket to the provided host and port.
        self.server.listen(2)   # Accept a maximum of 2 client connections.
        hostname=str(socket.gethostbyname(socket.gethostname()))
        print("TicTacPro server started on host " + hostname + " and port " + str(port))
        # Informs the user of the host IP address and the port number.
        #Clients must use this data to connect in order to play multiplayer online.
        print("")

    def shutdown(self):
        """
        FUNCTION NAME: shutdown()
        PARAMETERS: 0
        FUNCTION DESCRIPTION:
            Closes the connection to each client and the completely shuts down the server. No connection to the server can
        """
        for conn in self.connections:
            conn.close()
        self.server.shutdown(1)
        self.server.close()

    def play(self):
        """
        FUNCTION NAME: play()
        PARAMETERS: 0
        FUNCTION DESCRIPTION:
            Accepts new connections and appends them to the connection list (self.connections).
            Transfers information from one client to the other.
        """
        read, write, error = select.select(self.connections+[self.server], self.connections, self.connections, 0) # Get the

        for conn in read:
            if conn is self.server: # Handle new clients connecting to the server.
                c, addr = conn.accept()
                self.connections.append(c)
                print("New client connected: " + str(addr))
            else:
                data = conn.recv(1024)
                if not data: # If no data is received, the client has disconnected. Close the connection and end the online
                    conn.close()
                    self.shutdown()
                elif data: # If data is received, send it to the other client.
                    for client in self.connections:
                        if client != conn:
                            try:
                                client.send(data)
                            except Exception as e:
                                print("Could not send to " + str(conn))
                                logging.error("Could not send to " + str(conn) + " Exception details: " + str(e))

if __name__ == "__main__":
    logging.basicConfig(filename="events.log", format="%(asctime)s [server.py] %(message)s", datefmt="%Y/%m/%d %I:%M:%S %p"
    try:
        server = servergo()
        while True: # Keep server alive.
            server.play()
    except Exception as e:
        logging.exception(str(e)) # Log error to file.
    finally:
        server.shutdown()


###-------------NEWFILE

"""
============================== TicTacPro ==============================
FILE: Settings.py
MODIFIED: 15/11/2015
STATUS: Complete
FILE DESCRIPTION:
the Settings.py file is the state in which the program is changing visual
and audio settings about the game, such as the style or the music or the
difficulty.
"""
from main import *
from logic import *
#0-board
#1-x
#2-o
#3-xwin
#4-owin

def print(text,x=35,y=620):
    """
    FUNCTION NAME: print()
    PARAMETERS: 3
                text (string; mandatory): the output text intended to be shown to the user
                x (integer; optional): the X coordinate of the top left corner of the text output
                y (integer; optional): the Y coordinate of the top left corner of the text output
    FUNCTION DESCRIPTION:
        Allows drawing text anywhere on the screen via the coordinates of the top left
        corner of the text displayed. If no coordinates are passed in, it defaults to the bottom left of
        the screen, which is designated as the display bar section. Has to be defined here do
        to some unknown bug.
    """
    pygame.draw.rect(screen, (64,0,64), (0,610,650,50), 0)
    font = pygame.font.Font(None, 32)
    textg = font.render(str(text), 4, (255, 255, 255))
    screen.blit(textg, (x,y))
    pygame.display.flip()


def settingmenu():
    settingsmenu=pygame.image.load("images/menu/settings/settingsmenu.png")
    q=False
    print("UN")
    while not q:
        #display menu screen
        screen.blit(settingsmenu,(0,0))
        pygame.display.flip()
        ev = pygame.event.get()
        #EDP
        for event in ev:
            if event.type == pygame.MOUSEBUTTONUP:

                #pos is the coords for the mouse click
                pos = pygame.mouse.get_pos()
                #chooses which menu opens
                if pos[0] in range(70,260) and pos[1] in range(180,320):
                    images=stylemenu()
                elif pos[0] in range(350,540) and pos[1] in range(185,320):
                    songmenu()
                elif pos[0] in range(70,260) and pos[1] in range (380,510):
                    difficulty=difficultymenu()
                elif pos[0] in range(23,204) and pos[1] in range(540,600):
                    #return nothing and the game will continue to use whatever
                    #images are already loaded
                    try:
                        return (difficulty,images)

                    except:
                        try:
                            return (difficulty,None)

                        except:
                            try:
                                return (None,images)
                            except:
                                return (None,None)

def songmenu():
    songmenu=pygame.image.load("images/menu/settings/songmenu.png")
    screen.blit(songmenu,(0,0))
    pygame.display.flip()
    q=False
    while not q:
        ev = pygame.event.get()
        for event in ev:
            if event.type == pygame.MOUSEBUTTONUP:
                pos = pygame.mouse.get_pos()

                if pos[0] in range(70,260) and pos[1] in range(180,320):
                    print("Music Changed - Harder!")
                    pygame.mixer.music.load("music/harder.mp3")

                elif pos[0] in range(350,540) and pos[1] in range(180,320):
                    print("Music Changed - Lucky!")
                    pygame.mixer.music.load("music/lucky.mp3")

                elif pos[0] in range(70,260) and pos[1] in range (380,510):
                    print("Music Changed - Nostalgia!")
                    pygame.mixer.music.load("music/oldschool.mp3")

                elif pos[0] in range(350,540) and pos[1] in range(380,510):
                    print("Music Changed - Oh, Christmas Tree!")
                    pygame.mixer.music.load("music/christmas.mp3")

                elif pos[0] in range(23,204) and pos[1] in range(540,600): # Back button.
                    q=True

def difficultymenu():
    difficultymenuimg=pygame.image.load("images/menu/settings/difficultymenu.png")
    screen.blit(difficultymenuimg,(0,0))
    pygame.display.flip()
    q=False
    while not q:
        ev = pygame.event.get()
        for event in ev:
            if event.type == pygame.MOUSEBUTTONUP:
                pos = pygame.mouse.get_pos()
            #top left button
                if pos[0] in range(75,260) and pos[1] in range(190,320):
                    print("Difficulty set - Easy")
                    difficulty="easy"

                elif pos[0] in range(350,540) and pos[1] in range(185,320):
                    print("Difficulty set - Medium")
                    difficulty="medium"

                elif pos[0] in range(75,260) and pos[1] in range(370,510):
                    print("Difficulty set - Hard")
                    difficulty="hard"

                elif pos[0] in range(50,140) and pos[1] in range(560,590):
                    pass
                    try:
                        return difficulty
                    except:
                        return None
                elif event.type == pygame.QUIT:
                    quitgame()

def stylemenu():
    stylesmenu=pygame.image.load("images/menu/settings/stylemenu.png")
    screen.blit(stylesmenu,(0,0))
    pygame.display.flip()
    q=False
    while not q:
        ev = pygame.event.get()
        for event in ev:
            if event.type == pygame.MOUSEBUTTONUP:
                pos = pygame.mouse.get_pos()
            #top left button

                if pos[0] in range(30,200) and pos[1] in range(160,300):
                    print("Board Loaded - Classic Board")
                    #the game pulls images from the list, change the content
                    #of the list, the images used changes, for ease of use
                    images=[
                    pygame.image.load("images/classic/board.png"),
                    pygame.image.load('images/classic/x.png'),
                    pygame.image.load('images/classic/o.png'),
                    pygame.image.load("images/classic/xwon.png"),
                    pygame.image.load("images/classic/owin.png")
                    ]

                elif pos[0] in range(380,510) and pos[1] in range(160,300):
                    print("Board Loaded - Test Board")
                    images=[
                    pygame.image.load("images/test/board.png"),
                    pygame.image.load('images/test/x.png'),
                    pygame.image.load('images/test/o.png'),
                    pygame.image.load("images/test/xwon.png"),
                    pygame.image.load("images/test/owin.png")
                    ]

                elif pos[0] in range(30,200) and pos[1] in range(370,510):
                    print("Board Loaded - Christmas Board")
                    images=[
                    pygame.image.load("images/christmas/board.png"),
                    pygame.image.load('images/christmas/x.png'),
                    pygame.image.load('images/christmas/o.png'),
                    pygame.image.load("images/christmas/xwon.png"),
                    pygame.image.load("images/christmas/owin.png")
                    ]

                elif pos[0] in range(350,510) and pos[1] in range(370,510):
                    print("Board Loaded - Mushroom Board")
                    images=[
                    pygame.image.load("images/mushroom/board.png"),
                    pygame.image.load('images/mushroom/x.png'),
                    pygame.image.load('images/mushroom/o.png'),
                    pygame.image.load("images/mushroom/xwon.png"),
                    pygame.image.load("images/mushroom/owin.png")
                    ]

                elif pos[0] in range(23,204) and pos[1] in range(540,600):
                    try:
                        return images
                    except:
                        return None

if __name__=="__main__":
    settingmenu()


###-------------NEWFILE

"""
============================== TicTacPro ==============================
FILE: TicTacProLauncher.py
MODIFIED: 15/11/2015
STATUS: Complete
FILE DESCRIPTION:
    The TicTacToeLauncher.py file is the bootstrap for the game. This is the file
    that should be used to start the game. It prompts the user for a host name and
    port number port if he wishes to play the game online and starts the game with
    disabled online multiplayer should no host and port data be provided.
USAGE:
    Open the file and follow the prompts.
"""

import os, time, sys

def valYN(prompt):

    while True:
        value = input(prompt).upper()
        if value not in ["Y", "N"]:
            print("Please enter Y or N according to your answer.")
            continue
        else:
            return value

def valhost(prompt):
    while True:
        value = input(prompt)
        if " " in value:
            print("Host cannot contain spaces.")
            continue
        else:
            return value

def valport(prompt):
    while True:
        try:
            value = int(input(prompt))
            break
        except ValueError:
            print("Insert numbers only.")
            continue
    return str(value) # Convert to string in order to pass as sys argument.

def startGame(host, port):
        if sys.version_info[0] < 3:
            try:
```

```python
                os.system("python3 main.py " + host + " " + port) # If the computer has both Python 2 and Python 3
                #installed, this will start the game under Python 3.
            except:
                raise OSError("Please install Python 3 to play the game.")
        elif sys.version_info[0] == 3:
            os.system("main.py " + host + " " + port)

if __name__ == "__main__":
    print("=========================== Welcome to TicTacPro! ===========================")
    print("We are preparing the TicTacPro GUI for you.")
    ans1 = valYN("Before we start, we need to know whether you will want to play online. (Y/N)")
    if ans1 == "Y":
        print()
        print("In order to play in online mode please provide the host name and the port number you will be connecting to."
        host = valhost("Host: ")
        port = valport("Port: ")

        print()
        print("TicTacPro will now initiate with enabled online multiplayer on host " + host + " and port " + port)
        time.sleep(3)
        startGame(host, port)
    elif ans1 == "N":
        print("You have chosen to not play online during this session.")
        print("TicTacPro will now initiate with disabled online multiplayer.")
        startGame("no", "0")


###-------------NEWFILE

"""
-------------Intial File--------------

This is a file built by Jiminy Haynes before finalising the plan for the
TicTacToe game, it is a line based GUI and was made to test concepts
and understand the game mechanics better
"""

import time
import msvcrt
import os
os.system("mode con cols=50 lines=28")
def draw_board(box):
    print("-"*49)
    for count2 in range(0,3):
        for count in range(0,7):
            print("|{0}|{1}|{2}|".format(box[count2][0][count],box[count2][1][count],box[count2][2][count]))
        print("-"*49)

def generate_X():
    x0=(" x           x ")
    x1=("   x       x   ")
    x2=("     x   x     ")
    x3=("       x       ")
    x4=("     x   x     ")
    x5=("   x       x   ")
    x6=(" x           x ")
    x=[x0,x1,x2,x3,x4,x5,x6]
    return x

def generate_O():
    o0=("       x       ")
    o1=("     x   x     ")
    o2=("   x       x   ")
    o3=(" x           x ")
    o4=("   x       x   ")
    o5=("     x   x     ")
    o6=("       x       ")
    o=[o0,o1,o2,o3,o4,o5,o6]
    return o

def generate_B():
    b0=("               ")
    b1=("               ")
    b2=("               ")
    b3=("               ")
    b4=("               ")
    b5=("               ")
    b6=("               ")
    b=[b0,b1,b2,b3,b4,b5,b6]
    return b
box_x=generate_X()
box_o=generate_O()
box_b=generate_B()

def generate_board(board,command):
    #command 0 = player
    #command 1 = y axis
    #command 2 = x axis

    if command[0]=="x":
        board[command[1]][command[2]]=box_x
    elif command[0]=="o":
        board[command[1]][command[2]]=box_o
    draw_board(board)

def intial_generate_board():
    line1=[box_b,box_b,box_b]
    line2=[box_b,box_b,box_b]
    line3=[box_b,box_b,box_b]

    board=[line1,line2,line3]
    draw_board(board)
    return board


def get_user_move():
    valid=False
    while not valid:
        try:
            #user_move=int(input())
            user_move=int(msvcrt.getch())
            if user_move==7:
                move="a1"
            elif user_move==8:
                move="a2"
            elif user_move==9:
                move="a3"
            elif user_move==4:
                move="b1"
            elif user_move==5:
                move="b2"
            elif user_move==6:
                move="b3"
            elif user_move==1:
                move="c1"
            elif user_move==2:
                move="c2"
            elif user_move==3:
                move="c3"
            valid=True
        except ValueError:
            print("That's not a valid input!")
    return move

def generate_command(board,countplayer):
    while countplayer!=0:
        if countplayer//2==countplayer/2:
            print("Player Two (X)!")
            player="x"
        else:
            print("Player One (O)!")
            player="o"
        countplayer-=1
        valid=False
        while not valid:
            coord=get_user_move()

            if coord[0].lower()=="a":
                yaxis=0
            elif coord[0].lower()=="b":
                yaxis=1
            elif coord[0]=="c":
                yaxis=2
            else:
                print("FAIL")
            command=(player,yaxis,(int(coord[1])-1))
            valid=True
            if board[command[1]][command[2]][0]!=("               "):
                print("That is not a valid move!")
                valid=False
        return command,countplayer

def generate_combos():
    combo1=["00","01","02"]
    combo2=["00","10","20"]
    combo3=["02","12","22"]
    combo4=["20","21","22"]
    combo5=["01","11","21"]
    combo6=["10","11","12"]
    combo7=["00","11","22"]
    combo8=["02","11","20"]
    combos=[combo1,combo2,combo3,combo4,combo5,combo6,combo7,combo8]
    return combos

def game_won(gamewon,combos,board):
    for count in range(0,8):

        if (
            board [int(combos[count][0][0])] [int(combos[count][0][1])] [0] == board [int(combos[count][1][0])] [int(combos
            board [int(combos[count][1][0])] [int(combos[count][1][1])] [0] == board [int(combos[count][2][0])] [int(combos
            board [int(combos[count][2][0])] [int(combos[count][2][1])] [0] == (" x           x ")):

                gamewon=("Player 2",True)

        elif(
            board [int(combos[count][0][0])] [int(combos[count][0][1])] [0] == board [int(combos[count][1][0])] [int(combos
            board [int(combos[count][1][0])] [int(combos[count][1][1])] [0] == board [int(combos[count][2][0])] [int(combos
            board [int(combos[count][2][0])] [int(combos[count][2][1])] [0] == ("       x       ")):

                gamewon=("Player 1",True)

    return gamewon
def winstate(gamewon,board):
    draw_board(board)
    q=True
    print()
    print("{0} Has won!".format(gamewon[0]))
    time.sleep(3)
    return q

def losestate(board):
    draw_board(board)
    print()
    print("No one won!")
    time.sleep(3)
    q=True
    return q

def main():
    q=False
    countplayer=9
    combos=generate_combos()
    gamewon=("No One",False)
    board=intial_generate_board()
    print("Welcome to Tic Tac Pro!")
    while not q:
        if countplayer!=0:
            command,countplayer=generate_command(board,countplayer)
            generate_board(board,command)
            gamewon=game_won(gamewon,combos,board)
            print()
            if gamewon[1]==True:
                q=winstate(gamewon,board)

        else:
            gamewon=game_won(gamewon,combos,board)
            if gamewon[1]==True:
                q=winstate(gamewon,board)
            else:
                q=losestate(board)
    print("Thanks for playing!")
    time.sleep(3)
main()
```