the board class ---

from box import box from animation import animation import pygame import player from end_game import end_game class board:

```
#construct with rows ,cols and initialize box_list to empty list
def __init__(self,data):

    board.data =data
    board.rows= data["rows"]
    board.cols =data["cols"]
    board.speed = data["speed"]
    board.multiplier = data["multiplier"]
    self.total_box_width = data["cols"]*data["multiplier"]
    self.total_box_height= self.total_height=data["rows"]*data["multiplier"]
    self.total_width=self.total_box_width+350
    board.img_loc = data["img_loc"]
    board.rotation_speed=data["rotation_speed"]
    animation.multiplier = board.multiplier #setup the same on animation side
    board.box_list =[]
    board.animations = set() # it is a set
    board.remove_cycle=set()
    self.running=False
    self.update_disp=False
    self.count=0
    self.check_end=False

    self.state=1
    self.end_setup=False
    self.animation_owners=[]
    self.init_window()
    board.make_image()
    self.make_boxes()
    self.players = []

    i=0# for location fo ingame count
    for p in data["players"]:
        self.players.append(player.player(self,p,i))
        i+=1
    self.alive_players = self.players
    self.current = self.players[0]
    self.end_game=end_game(self)

def init_window(self):
    self.w1 = pygame.display.set_mode((self.total_width,self.total_height))
    self.icon=pygame.image.load(self.data["icon_loc"])
    pygame.display.set_icon(self.icon)
    pygame.display.set_caption(self.data["title"])
```

```
def make_boxes(self):

    for r in range(0,board.rows):
        for c in range(0,board.cols):
            #for each box first contruct box with board object reference
            temp = box(self) # self is the board instance
            # look at setup in box
            temp.setup(r,c)
            board.box_list.append(temp)



  # when making each box the rest of the boxes arent always available
  # like surrounding for 1,1 would include 2,1 which hasnt been made
  # so first store the surrounding data in tuples and then convert them into boxes
  # look at the setup function
    temp_f = lambda t : board.box_list[t[0]*board.cols+t[1]]
 # returns the postion of (row,col)th box in box_list and replaces the tuple with the

    for b in board.box_list:
        temp = map(temp_f,b.surrounding)
        b.surrounding = list(temp)



def make_image():
    board.img=[]
    img_main = pygame.image.load(board.img_loc)

    img=img_main.subsurface(0,0,200,200).convert_alpha()
    img=pygame.transform.scale(img,(board.multiplier,board.multiplier))
    board.img.append(img)

    img=img_main.subsurface(200,0,200,200).convert_alpha()
    img=pygame.transform.scale(img,(board.multiplier,board.multiplier))
    board.img.append(img)

    img=img_main.subsurface(0,200,200,200).convert_alpha()
    img=pygame.transform.scale(img,(board.multiplier,board.multiplier))
    board.img.append(img)

    img = img_main.subsurface(200,200,200,200).convert_alpha()
    img = pygame.transform.scale(img,(board.multiplier,board.multiplier))
    board.img.append(img)
    board.make_grid()





def make_grid():
    w = board.cols*board.multiplier
    h =board.rows*board.multiplier
    board.grid = pygame.Surface((w,h),pygame.SRCALPHA)
    for v_l in range(0,w,board.multiplier):
        pygame.draw.line(board.grid,(255,255,255),(v_l,0),(v_l,h))
    for h_l in range(0,h,board.multiplier):
        pygame.draw.line(board.grid,(255,255,255),(0,h_l),(w,h_l))
    pygame.draw.line(board.grid,(255,255,255),(0,h-1),(w,h-1))
    pygame.draw.line(board.grid,(255,255,255),(w-1,0),(w-1,h-1))

def user_event(self,pos):
    if self.state==1:
        if pos[0]>self.total_box_width or pos[1]>self.total_box_height:
            return
        id = pos[0]//self.multiplier +self.cols*(pos[1]//self.multiplier)
        if not(self.running):
            if self.add_atom(id):
                self.check_end=True


    elif self.state==2:
        self.end_game.update(pos)

def check_change(self):
    if self.check_end:
        if not(self.running):
            self.cycle()
            self.check_end=False


def cycle(self):


    self.count+=1
    self.current=self.players[self.count%len(self.players)]


    while self.current.alive==False:
        self.count+=1
        self.current=self.players[self.count%len(self.players)]




def run(self):

    self.w1.fill((0,0,0))
    if self.state==1:
        self.update()
        self.render()
        self.running = bool(self.animations) or self.running
        self.alive_players =[ a for a in self.players if a.alive or a in self.animati

        if len(self.alive_players)==1:
            self.state=2
            print(self.current.name,"won")

        for a in self.players:
            a.render()
            self.check_change()

        if self.update_disp:
            for a in self.players:
                a.update_holding()
            self.update_disp=False
    if self.state==2:
        if self.end_setup==False:
            self.end_game.setup()
        self.end_game.render()
        #self.main_running=False


def reset_all(self):
    self.animations.clear()
    self.animation_owners.clear()
    for a in self.box_list:
        a.events.clear()
        a.holding=0
        a.owner=None
    for a in self.players:
        a.boxes.clear()
        a.alive=True
        a.holding=0
        a.update_holding()
    self.count =0
    self.current=self.players[self.count]
    self.check_end=False

def render(self):
    if bool(board.animations):
        self.w1.blit(board.grid,(0,0))
    else:
        self.w1.blit(self.current.grid,(0,0))
    for a in self.box_list:
        a.render()
    for a in self.animations:
        a.render()

def update(self):
    self.running = False
    if board.remove_cycle: # if there are animations to be removed
    # then remove them
        self.remove_all()

    for b in self.box_list: # for all the boxes
        if b.update(): # update events in all boxes
            self.running =True  # this variable will be needed for pygame
            #to check if all boxes are in idle state

    for a in self.animations:# for each animation in animations
        a.update() # update animations



def remove_animation(self,anim):
    board.remove_cycle.add(anim)

def remove_all(self):
    # for each animation in remove_cycle remove that animation from
    # the animations list
    # why im doing this is because i cant remove an animation when im iterating throu
    # itself and so i need to store them for the next cycle

    for a in board.remove_cycle:
        self.animations.discard(a) # remove animation from animations


    board.remove_cycle.clear()
    #clear the remove_cycle as all to be removed animations have been removed

def add_atom(self,index):
    ret=True
    temp_box = board.box_list[index]
    if temp_box.owner!=None:
        if temp_box.owner!=self.current:
            pass
            ret=False
    else:
        self.current.add_box(temp_box)

    if ret:#if passed the filter
        temp_box.add_atom()
        if temp_box.holding!=0:
            self.current.add_box(temp_box)

    # as of now im calling directly but then this will
    # be a much more complex function later
    return ret
```