from gameobjects import \*

class OpponentDashboard(GameEngine,Frame):

"""A dashboard monitoring opponents. It is not another thread, instead,any new received actions will be processed by the receiver's GameEngine"""

def \_\_init\_\_(self, master,blocksize,name,level=1):

Frame.\_\_init\_\_(self)

#threading.Thread.\_\_init\_\_(self)

#Default background color

##Points to self

self.boss=self

##Points to actual master widget

self.master=master

self.bg="black"

self.blocksize=blocksize

self.mixer=False

##The computers ID

self.\_name=name

self.base\_level=level

##Set to True when this opponent pressed PLAY

self.ready=False

self.gameOver=False

#The main canvas and the map of the game

self.can = Canvas(self, width=10\*blocksize, height=20\*blocksize+5, bg=self.bg)

self.can.create\_line(0,0,10\*blocksize, 0, fill="white")

self.can.yview\_scroll(22, 'units')

self.change\_lock=threading.Lock()

#The hold canvas

self.hold\_can = Canvas(self, width=6\*blocksize, height=4\*blocksize, bg=self.bg)

#Widget placements

self.hold\_can.grid(row=0, column=0,columnspan=2, padx=5, pady=5, sticky=N)

self.can.grid(row=0, column=2, pady=5, rowspan=5)

#Font

x=12

if self.master.max==2:

x=19

self.font=font.Font(family='Comic Sans MS', size=x, weight='bold', slant='roman')

#Labels:

self.label\_name=Label(self,text=name,font=self.font)

self.label\_name.grid(row=2, column=0, sticky="SW")

self.label\_frame=Frame(self)

self.label\_frame.grid(row=4, column=0)

Label(self.label\_frame,text="Points: ",font=self.font).grid(row=1, column=0, sticky="SW")

Label(self.label\_frame,text="Level: ",font=self.font).grid(row=2, column=0, sticky="SW")

Label(self.label\_frame,text="Lines cleared: ",font=self.font).grid(row=3, column=0, sticky="SW")

self.l\_points=Label(self.label\_frame,text="0",font=self.font)

self.l\_levels=Label(self.label\_frame,text="%d"%self.base\_level,font=self.font)

self.l\_lines=Label(self.label\_frame,text="0",font=self.font)

self.l\_points.grid(row=1, column=1, sticky="SW")

self.l\_levels.grid(row=2, column=1, sticky="SW")

self.l\_lines.grid(row=3, column=1, sticky="SW")

self.GM = [[0]\*40 for x in range(10)]

self.OGM = [[0]\*40 for x in range(10)]

self.active = None

self.actions=[]

self.online=False #Because of inheritance

##This is important stuff here

self.commands = {"COORDS":self.set\_coords,

"ELIM":self.set\_eliminate,

"NEW": self.new\_mino,

"STAT":self.set\_statistics,

"LOCK":self.lock\_down,

"HOLD":self.hold,

"OVER":self.over,

"ATTACK": self.attacked,

"LIFT": self.lift,

"ABANDON": self.abandoned,

"READY": self.set\_ready,

"WON": self.won}

def defaults(self):

"""To reset to the default dashboard appearance and variables"""

self.GM = [[0]\*40 for x in range(10)]

self.OGM = [[0]\*40 for x in range(10)]

self.active = None

self.actions=[]

self.can.delete(ALL)

self.hold\_can.delete(ALL)

self.can.create\_line(0,0,10\*self.blocksize, 0, fill="white")

self.l\_points.config(text="0")

self.l\_levels.config(text="%d"%self.base\_level)

self.l\_lines.config(text="0")

def won(self):

"""When someone won the game, we reset everything"""

self.master.reset()

def abandoned(self):

"""An opponent abandoned the game"""

self.destroy()

def set\_ready(self):

"""Set this player ready, and send a check request to the master widget"""

self.ready=True

self.master.check\_ready()

def attacked(self, num):

"""This method forwards the line attack from the opponent's class to the Engine. The player, received a line attack from an opponent"""

num=int(num)

self.master.panel.gameThread.receive\_attacks(num)

def lift(self, gap="0"): #XD

"""This opponent received a line attack"""

gap=int(gap)

bs=self.blocksize

for x in range(10):

for y in range(40):

if self.OGM[x][y]!=0:

self.can.move(self.OGM[x][y], 0, -self.blocksize)

for x in range(10):

del self.GM[x][39]

del self.OGM[x][39]

if x == gap:

self.GM[x][0:0]=[0]

self.OGM[x][0:0]=[0]

continue

self.GM[x][0:0]=['B']

self.OGM[x][0:0]=[self.can.create\_rectangle(2+(bs\*x),(19)\*bs,2+bs+(bs\*x), (20)\*bs, fill="darkgray", outline="gray")]

def hold(self):

"""The player held his Tetromino"""

if self.ghost:

for i in self.ghost:

self.can.delete(i)

for x,y in self.active['coords']:

self.GM[x][y]=0

for i in self.active['objects']:

self.can.delete(i)

self.place\_hold(self.active['type'])

def run(self):

"""This function is a one time status-update check. Any actions gathered in this instance's property <actions> will be executed."""

self.change\_lock.acquire()

for item in self.actions:

action=item[0]

forward=item[1]

if forward!='0':

self.commands[action](forward)

else:

self.commands[action]()

self.actions=[]

self.change\_lock.release()

def log(self,msg):

"""Action log receiving function. Runs the action on the panel."""

self.change\_lock.acquire()

self.actions.append(msg)

self.change\_lock.release()

def over(self):

"""Once an opponent is out, call this function."""

self.gameOver=True

self.can.create\_line(0,0, 10\*self.blocksize, 20\*self.blocksize+5, fill="red", width=6)

self.can.create\_line(10\*self.blocksize, 0,0,20\*self.blocksize+5, fill="red", width=6)

self.master.check\_over()

def new\_mino(self, mino):

"""Function to receive the type and coords of the new mino"""

bs=self.blocksize

self.active=eval(mino).generate()

self.active['objects']=[]

for x,y in self.active['coords']:

self.GM[x][y]='A'

self.active['objects'].append(self.can.create\_rectangle(2+(bs\*x),-(y-19)\*bs,2+bs+(bs\*x), -(y-20)\*bs, fill=self.active['color']))

distance=self.distance\_from\_surface()

self.ghost=[]

for x,y in self.active['coords']:

self.ghost.append(self.can.create\_rectangle(2+(bs\*x),-(y-19-distance)\*bs,2+bs+(bs\*x), -(y-20-distance)\*bs, outline=self.active['color']))

for i in self.active['objects']:

self.can.tag\_raise(i)

def set\_coords(self, coords):

"""Function to receive the opponent's matrix"""

coords=eval(coords)

bs=self.blocksize

for x,y in self.active['coords']:

self.GM[x][y]=0

self.active['coords']=coords

for x,y in self.active['coords']:

self.GM[x][y]='A'

for i in range(4):

x,y=self.active['coords'][i]

self.can.coords(self.active['objects'][i],2+(bs\*x),-(y-19)\*bs,2+bs+(bs\*x), -(y-20)\*bs)

self.ghost\_adjust()

#self.lock\_down gets called by server, then eliminate

def set\_eliminate(self, lines):

"""Function to recieve the line ID-s to eliminate"""

lines=eval(lines)

self.eliminate=lines

self.clear\_marked\_lines()

def set\_statistics(self, stats):

"""Function to receive opponent stats"""

stats=eval(stats)

self.l\_points.config(text=stats[0])

self.l\_levels.config(text=stats[1])

self.l\_lines.config(text=stats[2])

def \_destroy(self):

pass