#Imports

import tkinter as tk

from tkinter.constants import \*

from tkinter import \*

import re

import sys

import time

import random

import math

from PIL import Image, ImageTk

import itertools

import tkinter.messagebox

#var

scale = 100

pieceSize = scale/4

chessCardinals = [(1,0),(0,1),(-1,0),(0,-1)]

chessDiagonals = [(1,1),(-1,1),(1,-1),(-1,-1)]

startpos = []

endpos = []

WHITE = "white"

BLACK = "black"

pieceSelected = False

i = 0

colortemp = "black"

#window initialization

root = tk.Tk()

root.tk.call('tk', 'scaling', 1000.0)

#window

canvas = tk.Canvas(root, width=scale\*8, height=scale\*8)

canvas.pack()

#tile locations

for y in range(0,9):

globals()['coordY%s' % y] = y\*scale-(scale/2)

for x in range(0,9):

globals()['coordX%s' % x] = x\*scale-(scale/2)

#peice image files

whitepawnopen = Image.open("textures\whitepawn.png")

whitepawnopen = whitepawnopen.resize((scale-10, scale-10), Image.ANTIALIAS)

whitepawnimage = ImageTk.PhotoImage(whitepawnopen)

blackpawnopen = Image.open("textures\lackpawn.png")

blackpawnopen = blackpawnopen.resize((scale-10, scale-10), Image.ANTIALIAS)

blackpawnimage = ImageTk.PhotoImage(blackpawnopen)

whiterookopen = Image.open("textures\whiterook.png")

whiterookopen = whiterookopen.resize((scale-10, scale-10), Image.ANTIALIAS)

whiterookimage = ImageTk.PhotoImage(whiterookopen)

blackrookopen = Image.open("textures\lackrook.png")

blackrookopen = blackrookopen.resize((scale-10, scale-10), Image.ANTIALIAS)

blackrookimage = ImageTk.PhotoImage(blackrookopen)

whiteknightopen = Image.open("textures\whiteknight.png")

whiteknightopen = whiteknightopen.resize((scale-10, scale-10), Image.ANTIALIAS)

whiteknightimage = ImageTk.PhotoImage(whiteknightopen)

blackknightopen = Image.open("textures\lackknight.png")

blackknightopen = blackknightopen.resize((scale-10, scale-10), Image.ANTIALIAS)

blackknightimage = ImageTk.PhotoImage(blackknightopen)

whitebishopopen = Image.open("textures\whitebishop.png")

whitebishopopen = whitebishopopen.resize((scale-10, scale-10), Image.ANTIALIAS)

whitebishopimage = ImageTk.PhotoImage(whitebishopopen)

blackbishopopen = Image.open("textures\lackbishop.png")

blackbishopopen = blackbishopopen.resize((scale-10, scale-10), Image.ANTIALIAS)

blackbishopimage = ImageTk.PhotoImage(blackbishopopen)

whitequeenopen = Image.open("textures\whitequeen.png")

whitequeenopen = whitequeenopen.resize((scale-10, scale-10), Image.ANTIALIAS)

whitequeenimage = ImageTk.PhotoImage(whitequeenopen)

blackqueenopen = Image.open("textures\lackqueen.png")

blackqueenopen = blackqueenopen.resize((scale-10, scale-10), Image.ANTIALIAS)

blackqueenimage = ImageTk.PhotoImage(blackqueenopen)

whitekingopen = Image.open("textures\whiteking.png")

whitekingopen = whitekingopen.resize((scale-10, scale-10), Image.ANTIALIAS)

whitekingimage = ImageTk.PhotoImage(whitekingopen)

blackkingopen = Image.open("textures\lackking.png")

blackkingopen = blackkingopen.resize((scale-10, scale-10), Image.ANTIALIAS)

blackkingimage = ImageTk.PhotoImage(blackkingopen)

#moves pieces

def Move(p1,x,y):

coords = canvas.coords(p1)

x = (x+1)\*scale-scale/2

y = (y+1)\*scale-scale/2

newCoords = [x-(scale/2),y-(scale/2)]

return canvas.coords(p1, newCoords)

#Game

class Game:

def \_\_init\_\_(self):

self.playersturn = BLACK

self.gameboard = {}

self.placeTiles()

self.placePieces()

self.main()

self.temp = []

def placePieces(self):

for i in range(0,8):

#creates white pieces

self.gameboard[(i,1)] = Pawn(WHITE,uniDict[WHITE][Pawn],1)

canvas.create\_image((globals()['coordX%s' % (i+1)])-(scale/2), coordY2-(scale/2), image = whitepawnimage, anchor = tk.NW, tags=('whitepiece'))

#creates black pieces

self.gameboard[(i,6)] = Pawn(BLACK,uniDict[BLACK][Pawn],-1)

canvas.create\_image((globals()['coordX%s' % (i+1)])-(scale/2), coordY7-(scale/2), image = blackpawnimage, anchor = tk.NW, tags=('blackpiece'))

placers = [Rook,Knight,Bishop,Queen,King,Bishop,Knight,Rook]

placerstrings = ["rook","knight","bishop","queen","king","bishop","knight","rook"]

for i in range(0,8):

#creates white pieces

self.gameboard[(i,0)] = placers[i](WHITE,uniDict[WHITE][placers[i]])

canvas.create\_image((globals()['coordX%s' % (i+1)])-(scale/2), coordY1-(scale/2), image = ((globals()['white%simage' % placerstrings[i]])), anchor = tk.NW, tags=('whitepiece'))

#creates black pieces

self.gameboard[((7-i),7)] = placers[i](BLACK,uniDict[BLACK][placers[i]])

canvas.create\_image((globals()['coordX%s' % (i+1)])-(scale/2), coordY8-(scale/2), image = ((globals()['black%simage' % placerstrings[i]])), anchor = tk.NW, tags=('blackpiece'))

placers.reverse()

def placeTiles(self):

color = 'black'

for y in range(8):

for x in range(8):

x1 = x\*scale

y1 = y\*scale

x2 = x1 + scale

y2 = y1 + scale

if color == 'white':

color = 'black'

else:

color = 'white'

canvas.create\_rectangle((x1, y1, x2, y2), fill=color, tags=('tile'))

if color == 'white':

color = 'black'

else:

color = 'white'

def main(self):

root.after(40, self.main)

def onMouseClick(event):

global pieces

global pieceSelected

global i

global pieceTemp

global tileTemp

global startpos

global endpos

global pieceID

global num2

global colortemp

item = canvas.find\_closest(event.x, event.y)

if i == 3:

canvas.delete('temp')

pieceSelected = False

i = 0

try:

target = self.gameboard[startpos]

except:

tk.messagebox.showinfo("Game","Invalid Move")

target = None

if target:

if target.isValid(startpos,endpos,target.Color,self.gameboard):

try:

self.gameboard[endpos]

except:

pass

else:

canvas.delete(canvas.find\_closest((endpos[0]\*scale)+scale/2,(endpos[1]\*scale)+scale/2))

self.gameboard[endpos] = self.gameboard[startpos]

Move(pieceID,endpos[0],endpos[1])

del self.gameboard[startpos]

self.isCheck()

if self.playersturn == BLACK:

self.playersturn = WHITE

colortemp = "white"

else:

self.playersturn = BLACK

colortemp = "black"

else:

tk.messagebox.showinfo("Game","Invalid Move")

else:

if pieceSelected == False:

if 'whitepiece' in canvas.gettags(item) and self.playersturn == WHITE:

pieceSelected = True

pieceID = canvas.find\_closest(event.x, event.y)

itemCoords = canvas.coords(item), (canvas.coords(item))[0]+(scale/4), (canvas.coords(item))[1]+(scale/4)

pieceTemp = canvas.create\_rectangle(itemCoords, fill="blue", tag ="temp")

startpos = int((canvas.coords(pieceTemp))[0]/scale), int((canvas.coords(pieceTemp))[1]/scale)

if 'blackpiece' in canvas.gettags(item) and self.playersturn == BLACK:

pieceSelected = True

pieceID = canvas.find\_closest(event.x, event.y)

itemCoords = canvas.coords(item), (canvas.coords(item))[0]+(scale/4), (canvas.coords(item))[1]+(scale/4)

pieceTemp = canvas.create\_rectangle(itemCoords, fill="blue", tag ="temp")

startpos = int((canvas.coords(pieceTemp))[0]/scale), int((canvas.coords(pieceTemp))[1]/scale)

elif pieceSelected == True:

if 'tile' in canvas.gettags(item) and i == 2:

current\_color = canvas.itemcget(item, 'fill')

tileTemp = canvas.create\_rectangle(canvas.coords((item)), fill="blue", tag="temp")

endpos = int((canvas.coords(tileTemp))[0]/scale), int((canvas.coords(tileTemp))[1]/scale)

i += 1

def deselect(event):

global i

global startpos

global endpos

global pieceSelected

canvas.delete('temp')

pieceSelected = False

i = 0

startpos = None

endpos = None

canvas.bind("<Button-1>", onMouseClick)

canvas.bind("<Button-3>", deselect)

def isCheck(self):

king = King

kingDict = {}

pieceDict = {BLACK : [], WHITE : []}

for position,piece in self.gameboard.items():

if type(piece) == King:

kingDict[piece.Color] = position

pieceDict[piece.Color].append((piece,position))

if self.canSeeKing(kingDict[WHITE],pieceDict[BLACK]):

winner = "black"

tk.messagebox.showinfo("Game","black won")

if self.canSeeKing(kingDict[BLACK],pieceDict[WHITE]):

winner = "white"

tk.messagebox.showinfo("Game","White won")

def canSeeKing(self,kingpos,piecelist):

for piece,position in piecelist:

if piece.isValid(position,kingpos,piece.Color,self.gameboard):

return True

#piece move validation

class Piece:

def \_\_init\_\_(self,color,name):

self.name = name

self.position = None

self.Color = color

def isValid(self,startpos,endpos,Color,gameboard):

if endpos in self.availableMoves(startpos[0],startpos[1],gameboard, Color = Color):

return True

return False

def \_\_repr\_\_(self):

return self.name

def \_\_str\_\_(self):

return self.name

def availableMoves(self,x,y,gameboard):

tk.messagebox.showinfo("Game","Invalid Move")

def AdNauseum(self,x,y,gameboard, Color, intervals):

answers = []

for xint,yint in intervals:

xtemp,ytemp = x+xint,y+yint

while self.isInBounds(xtemp,ytemp):

target = gameboard.get((xtemp,ytemp),None)

if target is None: answers.append((xtemp,ytemp))

elif target.Color != Color:

answers.append((xtemp,ytemp))

break

else:

break

xtemp,ytemp = xtemp + xint,ytemp + yint

return answers

def isInBounds(self,x,y):

if x >= 0 and x < 8 and y >= 0 and y < 8:

return True

return False

def noConflict(self,gameboard,initialColor,x,y):

if self.isInBounds(x,y) and (((x,y) not in gameboard) or gameboard[(x,y)].Color != initialColor) : return True

return False

chessCardinals = [(1,0),(0,1),(-1,0),(0,-1)]

chessDiagonals = [(1,1),(-1,1),(1,-1),(-1,-1)]

def knightList(x,y,int1,int2):

"""sepcifically for the rook, permutes the values needed around a position for noConflict tests"""

return [(x+int1,y+int2),(x-int1,y+int2),(x+int1,y-int2),(x-int1,y-int2),(x+int2,y+int1),(x-int2,y+int1),(x+int2,y-int1),(x-int2,y-int1)]

def kingList(x,y):

return [(x+1,y),(x+1,y+1),(x+1,y-1),(x,y+1),(x,y-1),(x-1,y),(x-1,y+1),(x-1,y-1)]

#piece movement rules

class Knight(Piece):

def availableMoves(self,x,y,gameboard, Color = None):

if Color is None : Color = self.Color

return [(xx,yy) for xx,yy in knightList(x,y,2,1) if self.noConflict(gameboard, Color, xx, yy)]

class Rook(Piece):

def availableMoves(self,x,y,gameboard ,Color = None):

if Color is None : Color = self.Color

return self.AdNauseum(x, y, gameboard, Color, chessCardinals)

class Bishop(Piece):

def availableMoves(self,x,y,gameboard, Color = None):

if Color is None : Color = self.Color

return self.AdNauseum(x, y, gameboard, Color, chessDiagonals)

class Queen(Piece):

def availableMoves(self,x,y,gameboard, Color = None):

if Color is None : Color = self.Color

return self.AdNauseum(x, y, gameboard, Color, chessCardinals+chessDiagonals)

class King(Piece):

def availableMoves(self,x,y,gameboard, Color = None):

if Color is None : Color = self.Color

return [(xx,yy) for xx,yy in kingList(x,y) if self.noConflict(gameboard, Color, xx, yy)]

class Pawn(Piece):

def \_\_init\_\_(self,color,name,direction):

self.name = name

self.Color = color

self.direction = direction

def availableMoves(self,x,y,gameboard, Color = None):

if Color is None : Color = self.Color

answers = []

if (x+1,y+self.direction) in gameboard and self.noConflict(gameboard, Color, x+1, y+self.direction) : answers.append((x+1,y+self.direction))

if (x-1,y+self.direction) in gameboard and self.noConflict(gameboard, Color, x-1, y+self.direction) : answers.append((x-1,y+self.direction))

if (x,y+self.direction) not in gameboard and Color == self.Color : answers.append((x,y+self.direction))

return answers

#spare parts

uniDict = {WHITE : {Pawn : "♙", Rook : "♖", Knight : "♘", Bishop : "♗", King : "♔", Queen : "♕" }, BLACK : {Pawn : "♟", Rook : "♜", Knight : "♞", Bishop : "♝", King : "♚", Queen : "♛" }}

Game()

pieces = []

root.mainloop()