Checkers in Python

Adrian Chen – ENGR 13300 – 001-23

# Description

This program creates checker pieces and a checkerboard in a pygame window with basic mouse capture by setting drawing checker pieces and controlling checker pieces as functions and classes.

# Overview

Inputs for this program are only keyboard keys when prompted and mouse movement and clicks. Output is a pygame window and the checkers game.

# User-Defined Functions

checkerboard()

This function draws the checkerboard pattern in an 8 by 8-pixel image, then upscales to the size of the screen

whitecp(), blackcp(), kingwhitecp(), kingblackcp()

These functions draw the white checker piece, black checker piece, king white checker piece and king black checker piece, respectively.

# User-Defined Classes

whitecp(), blackcp(), kingblackcp(), kingwhitecp()

These classes define the characteristics for each type of piece. These classes are referenced in the main program, where each pygame sprite is assigned an attribute from this list of classes.

# Interaction Diagram

Asks for a creation/reset

Gives information on movement

Checkerpiece\_controller

Contains classes that define the behavior of checkerpiece sprites

Creates/resets the playing field

Asks for checker piece characteristics

Checkerboard\_items

Creates the checkerboard and checkerpieces and stores the files

Main Program

Creates and runs the pygame window, includes exit conditions and mouse capture

# User Manual

====INSTRUCTIONS ON HOW TO PLAY====

The checkerboard is on a 8 by 8 checkerboard, and a display on the **right will show where** your selection is from (0,0) to (8,8)

To move a checker piece off the board, press the button "T"

To reset the game, press the button "R"

To change between a king and a regular piece, press the button "E"

To close the game, press the exit button for the window

The logic section would have added much more to the code, so currently, the movement is on an honor system

This shouldn't be a problem either way because the current game was two-player anyways

Occasionally, the game will glitch and crash on startup. If this happens, close the program and wait for the console to restart

Once the instance is reset to In [1], run again

Currently movement is not added into the game, but it includes all of the required components.

The project kinda got too out of hand on complexity.

# Appendix

## Checkerboard items

*#Import statements*

*#Uses Image for creating the .jpg file*

*#Uses ImageDraw for creating the shapes that represent each checkerpiece*

**from** PIL **import** Image, ImageDraw

*#Generates the checkerboard pattern, for initialization and reset*

*#First creates a 8 by 8 pixel image, then upscales to correct size*

**def** **checkerboard**():

size = 8

pix = Image.new("RGB", (size, size), "red")

pixelmap = pix.load()

**for** x **in** range(8):

**for** y **in** range(8):

**if** ((x%2==0 **and** y%2==1) **or** (x%2==1 **and** y%2==0)):

**for** i **in** range(x,x+1):

**for** j **in** range(y,y+1):

pixelmap[i,j] = (50, 50, 50)

pix = pix.resize((size\*100,size\*100),Image.NEAREST)

pix.save("checkerboard.jpg")

*#Code for drawing checkerpieces*

*#Size of canvas: 100 pixels: fits on current checkerboard scaling*

*#Draws the white checkerpiece*

**def** **whitecp**():

size = 100

pix = Image.new("RGB", (size, size), (50,50,50))

draw = ImageDraw.Draw(pix)

draw.ellipse((5,5,95,95), fill=(256,256,256))

draw.ellipse((7,7,93,93), fill=(0,0,0))

draw.ellipse((9,9,91,91), fill=(256,256,256))

**del** draw

pix.save("whitecp.jpg")

*#Draws the black checkerpiece*

**def** **blackcp**():

size = 100

pix = Image.new("RGB", (size, size), (50,50,50))

draw = ImageDraw.Draw(pix)

draw.ellipse((5,5,95,95), fill=(0,0,0))

draw.ellipse((7,7,93,93), fill=(256,256,256))

draw.ellipse((9,9,91,91), fill=(0,0,0))

**del** draw

pix.save("blackcp.jpg")

*#Draws the white king checkerpiece*

**def** **kingwhitecp**():

size = 100

pix = Image.new("RGB", (size, size), (50,50,50))

draw = ImageDraw.Draw(pix)

draw.ellipse((5,5,95,95), fill=(256,256,256))

draw.ellipse((7,7,93,93), fill=(0,0,0))

draw.ellipse((9,9,91,91), fill=(256,256,256))

draw.ellipse((20,20,80,80), fill=(0,0,0))

**del** draw

pix.save("kingwhitecp.jpg")

*#Draws the black king checkerpiece*

**def** **kingblackcp**():

size = 100

pix = Image.new("RGB", (size, size), (50,50,50))

draw = ImageDraw.Draw(pix)

draw.ellipse((5,5,95,95), fill=(0,0,0))

draw.ellipse((7,7,93,93), fill=(256,256,256))

draw.ellipse((9,9,91,91), fill=(0,0,0))

draw.ellipse((20,20,80,80), fill=(256,256,256))

**del** draw

pix.save("kingblackcp.jpg")

100%

## Checkerboard main

'''

====INSTRUCTIONS ON HOW TO PLAY====

Cursor movement is really hard to code, so movement is done using the keyboard

The checkerboard is on a 8 by 8 checkerboard, and a display on the right will **show** **where** your selection **is** **from** (0,0) **to** (8,8)

**To** **move** a checkerpiece **off** the board, press the button "T"

**To** **reset** the game, press the button "R"

**To** **change** **between** a king **and** a regular piece, press the button "E"

**To** **close** the game, press the **exit** button **for** the window

The logic **section** would have added much more **to** the code, so currently, the **movement** **is** **on** an honor **system**

This shouldn't be a problem either way because the current game was two-player anyways

Occasionally, the game will glitch and crash on startup. If this happens, close the program and wait for the console to restart

Once the instance is reset to In [1], run again

Currently movement is not added into the game, but it includes all of the required components

The project kinda got too out of hand on complexity

'''

# **Import** Statments

**import** checkerboard\_items **as** ch

**import** pygame

**import** pygame.locals

**from** checkerpiece\_controller **import** blackcp, whitecp, kingwhitecp, kingblackcp

# Various **variables** used later

selection = [0,0]

selected = True

# Reloads images **for** checkerpieces **and** checkerboard

ch.whitecp()

ch.checkerboard()

ch.blackcp()

ch.kingwhitecp()

ch.kingblackcp()

# Initialization **of** pygame **for** visuals **and** control

pygame.init()

# **Set** up the drawing window

screen = pygame.display.set\_mode([1200, 800])

pygame.display.set\_caption('Checkers Game - Adrian Chen ENGR 13300')

pygame.display.toggle\_fullscreen()

# Counters **for** checkerpieces **left** **on** either side

blackcpcount = 0

whitecpcount = 0

kingblackcpcount = 0

kingwhitecpcount = 0

# Text **for** the menu **and** game

checkerboard = pygame.image.load('checkerboard.jpg')

myfont = pygame.font.SysFont('Times New Roman', 30)

title = myfont.render('===Checkers Game===', True, (0,0,0))

restart = myfont.render('Restart - Press R', True, (0,0,0))

blackcheckercounter = myfont.render("{}={}".format("Black Checker Count", blackcpcount), True, (0,0,0))

whitecheckercounter = myfont.render("{}={}".format("White Checker Count", whitecpcount), True, (0,0,0))

blackkingcheckercounter = myfont.render("{}={}".format("Black King Checker Count", kingblackcpcount), True, (0,0,0))

whitekingcheckercounter = myfont.render("{}={}".format("White King Checker Count", kingwhitecpcount), True, (0,0,0))

# Loads the checkerpieces

blackcp = blackcp()

whitecp = whitecp()

kingwhitecp = kingwhitecp()

kingblackcp = kingblackcp()

# Fill the background **with** white **and** creates the **static** parts **of** the screen

screen.fill((255, 255, 255))

screen.blit(checkerboard, (0,0))

screen.blit(title, (820,10))

screen.blit(restart, (820,750))

# Generates the **first** **set** **of** checkerpieces

screen.blit(kingblackcp.image, (100,0))

screen.blit(blackcp.image, (300,0))

screen.blit(blackcp.image, (500,0))

screen.blit(blackcp.image, (700,0))

screen.blit(blackcp.image, (000,100))

screen.blit(blackcp.image, (200,100))

screen.blit(blackcp.image, (400,100))

screen.blit(blackcp.image, (600,100))

screen.blit(blackcp.image, (100,200))

screen.blit(blackcp.image, (300,200))

screen.blit(blackcp.image, (500,200))

screen.blit(blackcp.image, (700,200))

screen.blit(whitecp.image, (0,500))

screen.blit(whitecp.image, (200,500))

screen.blit(whitecp.image, (400,500))

screen.blit(whitecp.image, (600,500))

screen.blit(whitecp.image, (100,600))

screen.blit(whitecp.image, (300,600))

screen.blit(whitecp.image, (500,600))

screen.blit(whitecp.image, (700,600))

screen.blit(whitecp.image, (0,700))

screen.blit(whitecp.image, (200,700))

screen.blit(whitecp.image, (400,700))

screen.blit(whitecp.image, (600,700))

# Run **until** the **user** asks **to** quit

running = True

**while** running:

# **Exit** the **loop** **if** the **exit** button **is** pressed **on** the window

**for** **event** **in** pygame.event.get():

**if** event.type == pygame.QUIT:

running = False

# **If** the R button **is** pressed, **reset** all the checkerpieces back **to** **where** they were

**if** event.type == pygame.KEYDOWN **and** event.key == pygame.K\_r:

screen.blit(blackcp.image, (100,0))

screen.blit(blackcp.image, (300,0))

screen.blit(blackcp.image, (500,0))

screen.blit(blackcp.image, (700,0))

screen.blit(blackcp.image, (000,100))

screen.blit(blackcp.image, (200,100))

screen.blit(blackcp.image, (400,100))

screen.blit(blackcp.image, (600,100))

screen.blit(blackcp.image, (100,200))

screen.blit(blackcp.image, (300,200))

screen.blit(blackcp.image, (500,200))

screen.blit(blackcp.image, (700,200))

screen.blit(whitecp.image, (0,500))

screen.blit(whitecp.image, (200,500))

screen.blit(whitecp.image, (400,500))

screen.blit(whitecp.image, (600,500))

screen.blit(whitecp.image, (100,600))

screen.blit(whitecp.image, (300,600))

screen.blit(whitecp.image, (500,600))

screen.blit(whitecp.image, (700,600))

screen.blit(whitecp.image, (0,700))

screen.blit(whitecp.image, (200,700))

screen.blit(whitecp.image, (400,700))

screen.blit(whitecp.image, (600,700))

# Moves the selection

pygame.key.set\_repeat(2)

**if** event.type == pygame.KEYDOWN **and** event.key == pygame.K\_UP:

**if** selection[0] != 0:

selection[0] -= 1

print(selection)

**if** event.type == pygame.KEYDOWN **and** event.key == pygame.K\_RIGHT:

**if** selection[1] != 8:

selection[1] += 1

print(selection)

**if** event.type == pygame.KEYDOWN **and** event.key == pygame.K\_DOWN:

**if** selection[0] != 8:

selection[0] += 1

print(selection)

**if** event.type == pygame.KEYDOWN **and** event.key == pygame.K\_LEFT:

**if** selection[1] != 0:

selection[1] -= 1

print(selection)

**if** event.type == pygame.KEYDOWN **and** event.key == pygame.K\_SPACE:

**if** selected == True:

selected == False

print(selected)

**else**:

selected == True

print(selected)

# Adds checker **type** counters

blackcheckercounter = myfont.render("{}={}".format("Black Checker Count", blackcpcount), True, (0,0,0))

whitecheckercounter = myfont.render("{}={}".format("White Checker Count", whitecpcount), True, (0,0,0))

blackkingcheckercounter = myfont.render("{}={}".format("Black King Checker Count", kingblackcpcount), True, (0,0,0))

whitekingcheckercounter = myfont.render("{}={}".format("White King Checker Count", kingwhitecpcount), True, (0,0,0))

selectionblit = myfont.render("{}={}".format("Selection", selection), True, (0,0,0))

pygame.draw.rect(screen, (255, 255,255), (820,40,500,300))

screen.blit(blackcheckercounter, (820,40))

screen.blit(whitecheckercounter, (820,70))

screen.blit(blackkingcheckercounter, (820,100))

screen.blit(whitekingcheckercounter, (820,130))

screen.blit(selectionblit, (820,160))

pygame.display.flip()

#Exits the program

pygame.quit()

## Checkerboard controller

*# Import Statments*

**import** pygame

*# Classes defines position and sprite appearance for black, white, king black and king white checkerpieces*

*# Includes mouse capture and movement*

**class** **blackcp**(pygame.sprite.Sprite):

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

super().\_\_init\_\_()

pygame.sprite.Sprite.\_\_init\_\_(self)

self.surf = pygame.Surface((100,100))

self.image = pygame.image.load('blackcp.jpg').convert\_alpha()

self.rect = self.surf.get\_rect()

'''

while True:

for event in pygame.event.get():

if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:

pos = pygame.mouse.get\_pos()

if self.rect.collidepoint(event.pos):

return

'''

**class** **whitecp**(pygame.sprite.Sprite):

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

super().\_\_init\_\_()

pygame.sprite.Sprite.\_\_init\_\_(self)

self.surf = pygame.Surface((100,100))

self.image = pygame.image.load('whitecp.jpg').convert\_alpha()

self.rect = self.surf.get\_rect()

'''

while True:

for event in pygame.event.get():

if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:

pos = pygame.mouse.get\_pos()

if self.rect.collidepoint(event.pos):

return

'''

**class** **kingblackcp**(pygame.sprite.Sprite):

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

super().\_\_init\_\_()

pygame.sprite.Sprite.\_\_init\_\_(self)

self.surf = pygame.Surface((100,100))

self.image = pygame.image.load('kingblackcp.jpg').convert\_alpha()

self.rect = self.surf.get\_rect()

'''

while True:

for event in pygame.event.get():

if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:

pos = pygame.mouse.get\_pos()

if self.rect.collidepoint(event.pos):

return

'''

**class** **kingwhitecp**(pygame.sprite.Sprite):

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

super().\_\_init\_\_()

pygame.sprite.Sprite.\_\_init\_\_(self)

self.surf = pygame.Surface((100,100))

self.image = pygame.image.load('kingwhitecp.jpg').convert\_alpha()

self.rect = self.surf.get\_rect()

'''

while True:

for event in pygame.event.get():

if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:

pos = pygame.mouse.get\_pos()

if self.rect.collidepoint(event.pos):

return

'''