## properties.py

"""Contains changeable properties relevant to different aspects of the game."""  
  
FPS = 80  
  
SCREEN\_WIDTH = 1200  
BOARD\_HEIGHT = 600  
HALF\_WIDTH = SCREEN\_WIDTH / 2  
HALF\_HEIGHT = BOARD\_HEIGHT / 2  
CONTROLS\_HEIGHT = 100  
SCREEN\_HEIGHT = BOARD\_HEIGHT + CONTROLS\_HEIGHT  
SCREEN\_CENTER = (HALF\_WIDTH, SCREEN\_HEIGHT / 2)  
  
BORDER\_THICKNESS = 20  
HALF\_BORDER = BORDER\_THICKNESS / 2

## basicpages.py

from colors import ORANGE, CORNSILK, WHITE, BLACK  
from fonts import SMALL, XLARGE, LARGE, MEDLARGE  
from properties import SCREEN\_CENTER, HALF\_WIDTH  
from button import Button  
from eventmanager import EventManager  
import pygame  
from properties import \*  
from leaderboard import LeaderBoard  
  
class Page(object):  
 def \_\_init\_\_(self, screen, color, event\_manager):  
 self.color = color  
 self.screen = screen  
 self.buttons = []  
 self.texts = []  
 self.textboxes = []  
 self.caption = "PONG!!!"  
 self.event\_manager = event\_manager  
  
 def display(self):  
 pygame.display.set\_caption(self.caption)  
 self.screen.fill(self.color)  
 for text, text\_rect in self.texts:  
 self.screen.blit(text, text\_rect)  
 for button in self.buttons:  
 button.display()  
 pygame.display.flip()  
  
 def clean(self):  
 for button in self.buttons:  
 button.clean()  
  
class SplashScreen(Page):  
 def \_\_init\_\_(self, screen, event\_manager):  
 super(SplashScreen, self).\_\_init\_\_(screen, ORANGE, event\_manager)  
  
 splash = XLARGE.render("PONG!!!", True, WHITE)  
 splash\_rect = splash.get\_rect()  
 splash\_rect.center = SCREEN\_CENTER  
  
 self.texts.append((splash, splash\_rect))  
  
class TextInput(Page):  
 def \_\_init\_\_(self, screen, event\_manager, prompt, callback):  
 super(TextInput, self).\_\_init\_\_(screen, CORNSILK, event\_manager)  
  
 prompt = MEDLARGE.render(prompt, True, ORANGE)  
 prompt\_rect = prompt.get\_rect()  
 prompt\_rect.topleft = (20, HALF\_HEIGHT - 100)  
  
 self.texts.append((prompt, prompt\_rect))  
  
 self.value = ""  
 self.value\_rect = pygame.Rect(0, HALF\_HEIGHT, SCREEN\_WIDTH, 140)  
 self.value\_surface = pygame.Surface((SCREEN\_WIDTH, 140))  
  
 def cb():  
 callback(self.value)  
  
 submit\_rect = pygame.Rect(0, 0, 100, 60)  
 submit\_rect.bottomright = (SCREEN\_WIDTH - 20, SCREEN\_HEIGHT - 20)  
 submit = Button("Submit", submit\_rect, ORANGE, WHITE, cb, screen, event\_manager)  
  
 self.buttons.append(submit)  
   
 def display(self):  
 super(TextInput, self).display()  
 self.event\_manager.add\_input\_listener(self.add\_text)  
 self.show\_value()  
   
 def show\_value(self):  
 value\_img = MEDLARGE.render(self.value, True, BLACK)  
 self.value\_surface.fill(WHITE)  
 self.value\_surface.blit(value\_img, (20, 20))  
 self.screen.blit(self.value\_surface, (0, HALF\_HEIGHT))  
 pygame.display.update(self.value\_rect)  
   
 def add\_text(self, letter):  
 if letter == -1:  
 self.value = self.value[:-1]  
 else:  
 self.value += letter  
   
 self.show\_value()  
   
 def clean(self):  
 super(TextInput, self).clean()  
 self.event\_manager.remove\_input\_listener(self.add\_text)  
  
class LeaderBoardPage(Page):  
 def \_\_init\_\_(self, screen, event\_manager):  
 super(LeaderBoardPage, self).\_\_init\_\_(screen, CORNSILK, event\_manager)  
  
 header = LARGE.render("Leaderboard", True, ORANGE)  
 header\_rect = header.get\_rect()  
 header\_rect.midtop = (HALF\_WIDTH, 20)  
   
 heading = SMALL.render("Rank | Name | Matches Played | Wins | Losses | Point Difference | Win Percentage", True, ORANGE)  
 heading\_rect = heading.get\_rect()  
 heading\_rect.midtop = (HALF\_WIDTH, 150)  
  
 self.texts.extend([(header, header\_rect), (heading, heading\_rect)])  
  
 self.leaderboard = LeaderBoard()  
   
 def display(self):  
 del self.texts[2:]  
 TOP = self.texts[1][1].top + 35  
   
 format\_string = "{rank:>4d} | {name:20s} | {matches\_played:>14d} | {wins:>4d} | {losses:>6d} | {point\_diff:>16d} | {win\_per:>14.2f}"  
 for index, leader in enumerate(self.leaderboard.get\_top\_number(10)):  
 res = SMALL.render(format\_string.format(\*\*dict(zip(("rank", "name", "matches\_played", "wins", "losses", "point\_diff",   
 "win\_per"), leader))), True, BLACK)  
 res\_rect = res.get\_rect()  
 res\_rect.top = TOP + 35 \* index  
 res\_rect.centerx = HALF\_WIDTH  
 self.texts.append((res, res\_rect))  
   
 super(LeaderBoardPage, self).display()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 from properties import SCREEN\_HEIGHT, SCREEN\_WIDTH  
 from eventmanager import EventManager  
 c = pygame.time.Clock()  
 s = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))  
 e = EventManager()  
 l = LeaderBoardPage(s, e)  
 l.display()  
 while True:  
 e.run()  
 c.tick(30)

## boardpage.py

import pygame  
from properties import SCREEN\_WIDTH, BOARD\_HEIGHT, CONTROLS\_HEIGHT, HALF\_HEIGHT, HALF\_WIDTH, BORDER\_THICKNESS, HALF\_BORDER  
from gameobjects import Paddle, Ball  
from colors import CORNSILK, ORANGE, RED, WHITE, BLACK  
from fonts import NORMAL, MEDLARGE  
from basicpages import Page  
from leaderboard import LeaderBoard  
  
LINE\_THICKNESS = 2  
  
ARENA = pygame.Surface((SCREEN\_WIDTH, BOARD\_HEIGHT))  
ARENA.fill(CORNSILK)  
  
pygame.draw.line(ARENA, ORANGE, (HALF\_WIDTH, 0),  
 (HALF\_WIDTH, BOARD\_HEIGHT), LINE\_THICKNESS)  
pygame.draw.rect(  
 ARENA, ORANGE, ((0, 0), (SCREEN\_WIDTH, BOARD\_HEIGHT)), BORDER\_THICKNESS)  
pygame.draw.circle(ARENA, ORANGE, (HALF\_BORDER, HALF\_HEIGHT), 2 \* BOARD\_HEIGHT / 5, LINE\_THICKNESS)  
pygame.draw.circle(ARENA, ORANGE, (SCREEN\_WIDTH - HALF\_BORDER, HALF\_HEIGHT), 2 \* BOARD\_HEIGHT / 5, LINE\_THICKNESS)  
  
  
class BoardPage(Page):  
 LINE\_THICKNESS = 2  
 ARENA = ARENA  
 WINNING\_SCORE = 3  
  
 def \_\_init\_\_(self, screen, event\_manager, names, finish\_cb):  
 super(BoardPage, self).\_\_init\_\_(screen, CORNSILK, event\_manager)  
  
 def game\_end\_listener(winner):  
 self.suspended = True  
 self.score\_board.update(winner)  
 if not any(score == BoardPage.WINNING\_SCORE for score in self.score\_board.scores):  
 event\_manager.add\_timer(1, self.reset)  
 else:  
 self.declare\_winner()  
 event\_manager.add\_timer(3, finish\_cb)  
  
 event\_manager.add\_game\_end\_listener(game\_end\_listener)  
 event\_manager.add\_tick\_listener(self.run)  
 self.event\_manager = event\_manager  
  
 self.paddle\_1 = Paddle(30, RED, pygame.K\_w, pygame.K\_s, event\_manager)  
 self.paddle\_2 = Paddle(1160, RED, pygame.K\_UP,  
 pygame.K\_DOWN, event\_manager)  
 self.paddles = pygame.sprite.RenderUpdates()  
 self.paddles.add(self.paddle\_1, self.paddle\_2)  
  
 self.ball = Ball(RED, (self.paddle\_1, self.paddle\_2), event\_manager)  
 self.ball\_group = pygame.sprite.RenderUpdates()  
 self.ball\_group.add(self.ball)  
  
 self.suspended = True  
 self.score\_board = ScoreBoard(names, screen, event\_manager)  
   
 def declare\_winner(self):  
 self.score\_board.store\_result()  
 winner = 0 if self.score\_board.scores[0] == BoardPage.WINNING\_SCORE else 1  
 name = MEDLARGE.render(self.score\_board.names[winner], True, BLACK)  
 name\_rect = name.get\_rect()  
 name\_rect.center = (HALF\_WIDTH, HALF\_HEIGHT - 80)  
  
 wins = MEDLARGE.render("Wins!!!", True, BLACK)  
 wins\_rect = wins.get\_rect()  
 wins\_rect.center = (HALF\_WIDTH, HALF\_HEIGHT + 10)  
  
 self.screen.blit(name, name\_rect)  
 self.screen.blit(wins, wins\_rect)  
 pygame.display.update((name\_rect, wins\_rect))  
   
 def display(self):  
 super(BoardPage, self).display()  
 self.score\_board.render()  
 self.reset(2)  
   
 def start(self):  
 self.suspended = False  
  
 def reset(self, delay=0):  
 self.screen.blit(ARENA, (0, 0))  
  
 self.paddle\_1.reset()  
 self.paddle\_2.reset()  
 self.ball.reset()  
  
 self.paddles.clear(self.screen, ARENA)  
 self.ball\_group.clear(self.screen, ARENA)  
  
 self.paddles.draw(self.screen)  
 self.ball\_group.draw(self.screen)  
  
 pygame.display.flip()  
 self.event\_manager.add\_timer(delay, self.start)  
   
 def clean(self):  
 super(BoardPage, self).clean()  
 self.event\_manager.remove\_tick\_listener(self.run)  
 self.paddle\_1.clean()  
 self.paddle\_2.clean()  
  
 def run(self):  
 if not self.suspended:  
 self.paddles.clear(self.screen, ARENA)  
 self.ball\_group.clear(self.screen, ARENA)  
  
 self.ball\_group.update()  
  
 pygame.display.update(self.paddles.draw(self.screen))  
 pygame.display.update(self.ball\_group.draw(self.screen))  
  
  
class ScoreBoard(object):  
 def \_\_init\_\_(self, names, screen, event\_manager):  
 self.names = names  
 self.screen = screen  
 self.background = pygame.Surface((SCREEN\_WIDTH, CONTROLS\_HEIGHT))  
 self.rect = self.background.get\_rect()  
 self.rect.topleft = (0, BOARD\_HEIGHT)  
 self.leaderboard = LeaderBoard()  
  
 self.scores = [0, 0]  
  
 def update(self, winner):  
 self.scores[winner] += 1  
 self.render()  
  
 def render(self):  
 self.background.fill(ORANGE)  
 for number in xrange(2):  
 self.background.blit(  
 NORMAL.render(str(self.scores[number]), True, WHITE),  
 (HALF\_WIDTH \* (number + 1) - HALF\_BORDER - 40, HALF\_BORDER))  
 self.background.blit(  
 NORMAL.render(self.names[number], True, WHITE),  
 (HALF\_WIDTH \* number + HALF\_BORDER, HALF\_BORDER))  
 self.screen.blit(self.background, self.rect)  
 pygame.display.update(self.rect)  
   
 def store\_result(self):  
 for index in range(2):  
 self.leaderboard.store\_match\_result(self.names[index], self.scores[index] - self.scores[index ^ 1])

## button.py

import pygame  
from fonts import SMALL  
  
  
class Button(object):  
  
 def \_\_init\_\_(self, text, rect, color, text\_color, callback, screen, event\_manager):  
 self.rect = rect  
 self.image = pygame.Surface((rect.width, rect.height))  
 self.callback = callback  
 self.screen = screen  
  
 self.image.fill(color)  
  
 text = SMALL.render(text, True, text\_color)  
 text\_rect = text.get\_rect()  
 text\_rect.center = self.image.get\_rect().center  
 self.image.blit(text, text\_rect)  
  
 self.event\_manager = event\_manager  
  
 def display(self):  
 self.screen.blit(self.image, self.rect)  
 pygame.display.update(self.rect)  
 self.event\_manager.add\_click\_listener(self.callback, self.rect)  
  
 def clean(self):  
 self.event\_manager.remove\_click\_listener(self.callback)

## colors.py

"""Named colors for reusability."""  
  
RED = (255, 0, 0)  
GREEN = (0, 255, 0)  
BLUE = (0, 0, 255)  
BLACK = (0, 0, 0)  
ORANGE = (255, 165, 0)  
WHITE = (255, 255, 255)  
CORNSILK = (255, 230, 220)

## eventmanager.py

import sys  
import heapq  
  
import pygame  
from properties import FPS  
  
  
class EventManager(object):  
 """An event manager that dispatches event listeners."""  
  
 def \_\_init\_\_(self):  
 self.key\_listeners = {}  
 self.exit\_listeners = set()  
 self.game\_end\_listeners = set()  
 self.timed = []  
 self.ticker = 0  
 self.input\_listeners = set()  
 self.click\_listeners = {}  
 self.tick\_listeners = set()  
 self.num\_timers = 0  
  
 def add\_exit\_listener(self, listener):  
 self.exit\_listeners.add(listener)  
  
 def add\_key\_listener(self, key, listener):  
 try:  
 self.key\_listeners[key].add(listener)  
 except KeyError:  
 self.key\_listeners[key] = set()  
 self.add\_key\_listener(key, listener)  
  
 def remove\_key\_listener(self, key, listener):  
 self.key\_listeners[key].remove(listener)  
  
 def notify\_game\_end(self, winner):  
 for listener in self.game\_end\_listeners:  
 listener(winner)  
  
 def add\_game\_end\_listener(self, listener):  
 self.game\_end\_listeners.add(listener)  
  
 def add\_timer(self, seconds, listener):  
 heapq.heappush(self.timed, (self.ticker +  
 int(seconds \* FPS), self.num\_timers, listener))  
 self.num\_timers += 1  
  
 def add\_input\_listener(self, listener):  
 self.input\_listeners.add(listener)  
  
 def remove\_input\_listener(self, listener):  
 self.input\_listeners.remove(listener)  
  
 def add\_click\_listener(self, listener, rect):  
 self.click\_listeners[listener] = rect  
  
 def remove\_click\_listener(self, listener):  
 del self.click\_listeners[listener]  
  
 def add\_tick\_listener(self, listener):  
 self.tick\_listeners.add(listener)  
  
 def remove\_tick\_listener(self, listener):  
 self.tick\_listeners.remove(listener)  
  
 def run(self):  
 pressed = pygame.key.get\_pressed()  
 for listener in self.tick\_listeners:  
 listener()  
  
 for key in self.key\_listeners:  
 if pressed[key]:  
 for listener in self.key\_listeners[key]:  
 listener()  
  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 self.key\_listeners = None  
 self.game\_end\_listeners = None  
  
 for listener in self.exit\_listeners:  
 listener()  
 self.exit\_listeners = None  
 pygame.quit()  
 sys.exit()  
 elif event.type == pygame.KEYDOWN:  
 try:  
 if event.key == pygame.K\_BACKSPACE:  
 for listener in self.input\_listeners:  
 listener(-1)  
 elif 'a' <= chr(event.key) <= 'z' or '0' <= chr(event.key) <= '9' or event.key == pygame.K\_SPACE:  
 if pressed[pygame.K\_RSHIFT] or pressed[pygame.K\_LSHIFT]:  
 char = chr(event.key).upper()  
 else:  
 char = chr(event.key)  
 for listener in self.input\_listeners:  
 listener(char)  
 except ValueError:  
 pass  
 elif event.type == pygame.MOUSEBUTTONUP and event.button == 1:  
 for listener, rect in self.click\_listeners.items():  
 if rect.left <= event.pos[0] <= rect.right and rect.top <= event.pos[1] <= rect.bottom:  
 listener()  
  
 while self.timed and self.ticker >= self.timed[0][0]:  
 self.timed[0][2]()  
 heapq.heappop(self.timed)  
  
 self.ticker += 1

## fonts.py

from sys import platform  
  
import pygame  
pygame.init()  
  
if platform.startswith("linux") or platform == "darwin":  
 FONT\_NAME = "FreeMono"  
else:  
 FONT\_NAME = "Lucida Console"  
  
SMALL = pygame.font.SysFont(FONT\_NAME, 20)  
NORMAL = pygame.font.SysFont(FONT\_NAME, 40)  
MEDLARGE = pygame.font.SysFont(FONT\_NAME, 80)  
LARGE = pygame.font.SysFont(FONT\_NAME, 100)  
XLARGE = pygame.font.SysFont(FONT\_NAME, 280)

## gameobjects.py

import random  
  
import pygame  
  
from properties import BOARD\_HEIGHT, SCREEN\_WIDTH, HALF\_HEIGHT, HALF\_WIDTH, HALF\_BORDER  
  
  
class Paddle(pygame.sprite.DirtySprite):  
 PADDLE\_HEIGHT = 100  
 PADDLE\_WIDTH = 15  
 MOVE\_DISTANCE = 5  
 TOP = HALF\_BORDER  
 BOTTOM = BOARD\_HEIGHT - HALF\_BORDER  
  
 def \_\_init\_\_(self, x\_coordinate, color, up\_key, down\_key, event\_manager):  
 super(Paddle, self).\_\_init\_\_()  
  
 self.image = pygame.Surface(  
 (Paddle.PADDLE\_WIDTH, Paddle.PADDLE\_HEIGHT))  
 self.image.fill(color)  
  
 self.rect = self.image.get\_rect()  
 self.x\_coordinate = x\_coordinate  
  
 self.reset()  
  
 def up\_mover():  
 if self.rect.top >= Paddle.TOP + Paddle.MOVE\_DISTANCE:  
 self.move(-Paddle.MOVE\_DISTANCE)  
  
 def down\_mover():  
 if self.rect.bottom <= Paddle.BOTTOM - Paddle.MOVE\_DISTANCE:  
 self.move(Paddle.MOVE\_DISTANCE)  
  
 event\_manager.add\_key\_listener(up\_key, up\_mover)  
 event\_manager.add\_key\_listener(down\_key, down\_mover)  
  
 self.up\_mover = up\_mover  
 self.down\_mover = down\_mover  
 self.up\_key = up\_key  
 self.down\_key = down\_key  
 self.event\_manager = event\_manager  
  
 def reset(self):  
 self.rect.center = (  
 self.x\_coordinate + Paddle.PADDLE\_WIDTH / 2, HALF\_HEIGHT)  
   
 def clean(self):  
 self.event\_manager.remove\_key\_listener(self.up\_key, self.up\_mover)  
 self.event\_manager.remove\_key\_listener(self.down\_key, self.down\_mover)  
  
 def move(self, distance):  
 """Moves the Paddle rect  
  
 -ve: Move up  
 +ve: Move down  
 """  
 self.dirty = 1  
 self.rect.move\_ip(0, distance)  
  
  
class Ball(pygame.sprite.DirtySprite):  
 BALL\_DIMENSIONS = 12  
 MOVE\_DISTANCE = 8  
 TOP = HALF\_BORDER  
 BOTTOM = BOARD\_HEIGHT - HALF\_BORDER  
 LEFT = HALF\_BORDER  
 RIGHT = SCREEN\_WIDTH - HALF\_BORDER  
 DIRECTIONS = (-1, 1)  
  
 def \_\_init\_\_(self, color, paddles, event\_manager):  
 super(Ball, self).\_\_init\_\_()  
  
 self.dirty = 2  
 self.image = pygame.Surface(  
 (Ball.BALL\_DIMENSIONS, Ball.BALL\_DIMENSIONS))  
 self.image.fill(color)  
  
 self.rect = self.image.get\_rect()  
 self.rect.center = (HALF\_WIDTH, HALF\_HEIGHT)  
  
 self.x\_velocity = random.choice(Ball.DIRECTIONS) \* Ball.MOVE\_DISTANCE  
 self.y\_velocity = random.choice(Ball.DIRECTIONS) \* Ball.MOVE\_DISTANCE  
  
 self.left\_paddle, self.right\_paddle = paddles  
 self.left\_limit = self.left\_paddle.rect.right  
 self.right\_limit = self.right\_paddle.rect.left  
  
 self.not\_crossed = True  
 self.event\_manager = event\_manager  
  
 def reflect\_coordinate(self, reference, coordinate, velocity):  
 return 2 \* reference - coordinate - velocity  
  
 def move\_y(self):  
 if Ball.TOP > self.rect.top + self.y\_velocity:  
 self.rect.top = self.reflect\_coordinate(  
 Ball.TOP, self.rect.top, self.y\_velocity)  
 elif Ball.BOTTOM < self.rect.bottom + self.y\_velocity:  
 self.rect.bottom = self.reflect\_coordinate(  
 Ball.BOTTOM, self.rect.bottom, self.y\_velocity)  
 else:  
 self.rect.move\_ip(0, self.y\_velocity)  
 return  
 self.y\_velocity \*= -1  
  
 def in\_limits(self, paddle):  
 return paddle.rect.bottom >= self.rect.centery >= paddle.rect.top  
  
 def move\_x(self):  
 if self.in\_limits(self.left\_paddle) and 0 >= self.left\_limit - self.rect.left > self.x\_velocity:  
 self.rect.left = self.reflect\_coordinate(  
 self.left\_limit, self.rect.left, self.x\_velocity)  
 elif self.in\_limits(self.right\_paddle) and 0 <= self.right\_limit - self.rect.right < self.x\_velocity:  
 self.rect.right = self.reflect\_coordinate(  
 self.right\_limit, self.rect.right, self.x\_velocity)  
 else:  
 self.rect.move\_ip(self.x\_velocity, 0)  
 return  
 self.x\_velocity \*= -1  
  
 def update(self):  
 self.move\_x()  
 self.move\_y()  
 if self.rect.left <= Ball.LEFT or self.rect.right >= Ball.RIGHT:  
 winner = 1 if self.rect.left <= Ball.LEFT else 0  
 self.not\_crossed = False  
 if self.rect.left <= Ball.LEFT:  
 xloc = Ball.LEFT  
 else:  
 xloc = Ball.RIGHT - Ball.BALL\_DIMENSIONS  
 self.rect.move\_ip(0, abs(xloc - self.rect.left) \* (self.y\_velocity / abs(self.y\_velocity)))  
 self.rect.left = xloc  
 self.event\_manager.notify\_game\_end(winner)  
  
 def reset(self):  
 self.rect.center = (HALF\_WIDTH, HALF\_HEIGHT)

## leaderboard.py

import pickle as p  
  
NAME = 0  
POINT\_DIFFS = 1  
WINS = 2  
MATCHES = 3  
  
class LeaderBoard(object):  
 def \_\_init\_\_(self):  
 self.file = "leaderlist.dat"  
 with open(self.file, "ab") as \_:  
 pass  
  
  
  
 @staticmethod  
 def win\_percentage(person):  
 return float(person[WINS] \* 100) / person[MATCHES]  
  
  
 @staticmethod  
 def compare\_leaders(person\_1, person\_2):  
 return cmp((LeaderBoard.win\_percentage(person\_1), person\_1[POINT\_DIFFS], person\_1[WINS]),  
 (LeaderBoard.win\_percentage(person\_2), person\_2[POINT\_DIFFS], person\_2[WINS]))  
  
 @staticmethod  
 def sort(array):  
 return sorted(array, cmp=LeaderBoard.compare\_leaders, reverse=True)  
  
 def get\_top\_number(self, number):  
 with open(self.file, "rb") as f:  
 raw\_leaders = LeaderBoard.sort(p.load(f))[:number]  
  
 new\_leaders = []  
 prev\_leader = ["Batman", 0, 2, 1]  
 rank = 0  
 count = 1  
 for leader in raw\_leaders:  
 if LeaderBoard.compare\_leaders(leader, prev\_leader) == -1:  
 rank += count  
 count = 1  
 else:  
 count += 1  
  
 new\_leaders.append(  
 [rank, leader[NAME], leader[MATCHES], leader[WINS], leader[MATCHES] - leader[WINS], leader[POINT\_DIFFS], LeaderBoard.win\_percentage(leader)]  
 )  
  
 prev\_leader = leader  
  
 return new\_leaders  
  
 def store\_match\_result(self, name, point\_diff):  
 with open(self.file, "rb") as f:  
 try:  
 people = p.load(f)  
 except:  
 people = []  
  
 for person in people:  
 if person[NAME].lower() == name.lower():  
 person[MATCHES] += 1  
 person[POINT\_DIFFS] += point\_diff  
 if point\_diff > 0:  
 person[WINS] += 1  
 break  
 else:  
 people.append([name, point\_diff, 1 if point\_diff > 0 else 0, 1])  
  
 with open(self.file, "wb") as f:  
 p.dump(people, f)

## main.py

import pygame  
from eventmanager import EventManager  
from basicpages import Page, SplashScreen, TextInput, LeaderBoardPage  
from boardpage import BoardPage  
from colors import BLACK  
from properties import SCREEN\_WIDTH, SCREEN\_HEIGHT, FPS  
  
class Browser(object):  
 def \_\_init\_\_(self):  
 self.clock = pygame.time.Clock()  
 self.screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))  
 self.event\_manager = EventManager()  
 self.current\_page = Page(pygame.Surface((0, 0)), BLACK, self.event\_manager)  
  
 self.name\_1 = "Batman"  
 self.name\_2 = "Superman"  
   
 def get\_shower(self, klass, \*args):  
 def shower():  
 self.current\_page.clean()  
 self.current\_page = klass(self.screen, self.event\_manager, \*args)  
 self.current\_page.display()  
 return shower  
   
 def run(self):  
 self.get\_shower(SplashScreen)()  
  
 def cb1(value):  
 self.name\_1 = value  
 self.get\_shower(TextInput, "Enter Player 2's name", cb2)()  
   
 def finish\_cb():  
 self.get\_shower(LeaderBoardPage)()  
   
 def cb2(value):  
 self.name\_2 = value  
 self.get\_shower(BoardPage, (self.name\_1, self.name\_2), finish\_cb)()  
   
 self.event\_manager.add\_timer(2, self.get\_shower(TextInput, "Enter Player 1's name", cb1))  
  
 while True:  
 self.event\_manager.run()  
 self.clock.tick(FPS)  
if \_\_name\_\_ == '\_\_main\_\_':  
 BROWSER = Browser()  
 BROWSER.run()