```
from math import sin,cos,radians
import random


""" This is the model of the game"""
class Game:
    """ Create a game with a given size of cannon (length of sides) and projectiles
(radius) """
    def __init__(self, cannonSize, ballSize):
        self.cannonSize = cannonSize
        self.ballSize = ballSize
        self.players = [Player(self, False, -90, "blue"), Player(self, True, 90,
"red")]
        self.windspeed = 0
        self.turn = 0

    """ A list containing both players """
    def getPlayers(self):
        return self.players

    """ The height/width of the cannon """
    def getCannonSize(self):
        return self.cannonSize

    """ The radius of cannon balls """
    def getBallSize(self):
        return self.ballSize

    """ The current player, i.e. the player whose turn it is """
    def getCurrentPlayer(self):
        return self.players[self.turn]

    """ The opponent of the current player """
    def getOtherPlayer(self):
        if self.turn == 0:
            return self.players[1]
        else:
            return self.players[0]

    """ The number (0 or 1) of the current player. This should be the position of
the current player in getPlayers(). """
    def getCurrentPlayerNumber(self):
        return self.turn

    """ Switch active player """
    def nextPlayer(self):
        if self.turn == 0:
            self.turn = 1
        else:
            self.turn = 0

    """ Set the current wind speed, only used for testing """
    def setCurrentWind(self, wind):
        self.windspeed = wind

    def getCurrentWind(self):
        return self.windspeed

    """ Start a new round with a random wind value (-10 to +10) """
```

```
    def newRound(self):
        self.windspeed = random.random()*20-10

""" Models a player """
class Player:

    def __init__(self, game, isReversed, xPos, col):
        self.game = game
        self.isReversed = isReversed
        self.xPos = xPos
        self.col = col
        self.angle = 0
        self.velocity = 0
        self.score = 0

    """ Create and return a projectile starting at the centre of this players
cannon. Replaces any previous projectile for this player. """
    def fire(self, angle, velocity):
        self.angle = angle
        self.velocity = velocity
        if not self.isReversed:
            self.projectile = Projectile(angle, velocity,
self.game.getCurrentWind(), self.xPos, self.game.getCannonSize()/2, -110, 110)
        else:
            self.projectile = Projectile(180-angle, velocity,
self.game.getCurrentWind(), self.xPos, self.game.getCannonSize()/2, -110, 110)
        return self.projectile

    """ Gives the x-distance from this players cannon to a projectile. If the
cannon and the projectile touch (assuming the projectile is on the ground and
factoring in both cannon and projectile size) this method should return 0"""
    def projectileDistance(self, proj):
        playerPos = Player.getX(self)
        projectilePos = Projectile.getX(proj)
        compBallSize = self.game.getBallSize()
        compCannonSize = self.game.getCannonSize()

        if projectilePos >= (playerPos - compCannonSize/2) and projectilePos <=
(playerPos + compCannonSize/2): # If projectile lands on cannon
            distance = 0
        elif projectilePos > playerPos:
# If projectile lands one the right side of the cannon
            distance = projectilePos - playerPos - (compCannonSize/2) -
compBallSize
        elif projectilePos < playerPos:
# If the projectile lands on the left side of the cannon
            distance = - playerPos + projectilePos + (compCannonSize/2) +
compBallSize
        return distance

    """ The current score of this player """
    def getScore(self):
        return self.score

    """ Increase the score of this player by 1."""
    def increaseScore(self):
        self.score += 1

    """ Returns the color of this player (a string)"""
```

```
    def getColor(self):
        return self.col

    """ The x-position of the centre of this players cannon """
    def getX(self):
        return self.xPos

    """ The angle and velocity of the last projectile this player fired, initially
(45, 40) """
    def getAim(self):
        return self.angle, self.velocity



""" Models a projectile (a cannonball, but could be used more generally) """
class Projectile:
    """
        Constructor parameters:
        angle and velocity: the initial angle and velocity of the projectile
            angle 0 means straight east (positive x-direction) and 90 straight up
        wind: The wind speed value affecting this projectile
        xPos and yPos: The initial position of this projectile
        xLower and xUpper: The lowest and highest x-positions allowed
    """
    def __init__(self, angle, velocity, wind, xPos, yPos, xLower, xUpper):
        self.yPos = yPos
        self.xPos = xPos
        self.xLower = xLower
        self.xUpper = xUpper
        theta = radians(angle)
        self.xvel = velocity*cos(theta)
        self.yvel = velocity*sin(theta)
        self.wind = wind


    """
        Advance time by a given number of seconds
        (typically, time is less than a second,
         for large values the projectile may move erratically)
    """
    def update(self, time):
        # Compute new velocity based on acceleration from gravity/wind
        yvel1 = self.yvel - 9.8*time
        xvel1 = self.xvel + self.wind*time

        # Move based on the average velocity in the time period
        self.xPos = self.xPos + time * (self.xvel + xvel1) / 2.0
        self.yPos = self.yPos + time * (self.yvel + yvel1) / 2.0

        # make sure yPos >= 0
        self.yPos = max(self.yPos, 0)

        # Make sure xLower <= xPos <= mUpper
        self.xPos = max(self.xPos, self.xLower)
        self.xPos = min(self.xPos, self.xUpper)

        # Update velocities
        self.yvel = yvel1
        self.xvel = xvel1
```

```
    """ A projectile is moving as long as it has not hit the ground or moved
outside the xLower and xUpper limits """
    def isMoving(self):
        return 0 < self.getY() and self.xLower < self.getX() < self.xUpper

    def getX(self):
        return self.xPos

    """ The current y-position (height) of the projectile". Should never be below
0. """
    def getY(self):
        return self.yPos
```