```python
"""create_environments module of "AI training python"

Used to generate the training_data.csv file which is used for
simulating different environments (ENVs)

Returns:
    None
"""
# My standard linting settings
# pylint: disable=trailing-whitespace
# pylint: disable=logging-fstring-interpolation
# pylint: disable=line-too-long

import random
import csv
from math import sqrt

# possible_moves are "shockwave", "bl", "tl", "br", "tr"
x, y = (410, 410) # sprite_width and sprite_height of play_frame in GM
middle_x, middle_y = (x/2, y/2)
middle_point = (middle_x, middle_y)
RADIUS = 150 # threshold value of the radius(distance) to choose the "shockwave" option, RADIUS is a constant
FILENAME = "training_data.csv" # the name of the file to save the training data to, FILENAME is a constant
NUM_OF_START_ENVS = 2000

def get_points(max_x:int, max_y:int, number_of_points:int) -> list[tuple]:
    """Generate a list of random points on the plane
        0 < x <= max_x
        0 < y <= max_y

    Args:
        max_x (int): the maximum x value for the randomly generated points
        max_y (int): the maximum y value for the randomly generated points
        number_of_points (int): the number of random points to generate

    Returns:
        list[tuple] or list[x, y]: list of points with x and y coordinates
    """
    points:list[tuple] = []
    for _i in range(number_of_points):
        ran_x = random.randint(0, max_x)
        ran_y = random.randint(0, max_y)
        point = (ran_x, ran_y)
        points.append(point)
    return points

def calculate_best_move(point:tuple) -> str:
    """Calculate the best move for a given point.

    Args:
        point (tuple): point with x and y coordinate in range of max_x and max_y

    Returns:
        str: the calculated best move for the given point
    """
    vector_x = point[0] - middle_point[0]
    vector_y = point[1] - middle_point[1]
    distance = sqrt((vector_x**2 + vector_y**2))
    if distance <= RADIUS: # if the point is in the circle
        best_move = "shockwave"
    elif point[0] < middle_x: # if point on the left side of coordinate system
        if point[1] < middle_y:
            best_move = "bottom_left"
        else:
            best_move = "top_left"
    elif point[0] >= middle_x:
        if point[1] < middle_y:
            best_move = "bottom_right"
        else:
            best_move = "top_right"
```

```
71
72          return best_move
73
74      def compile_data(point:tuple, calculated_move:str) -> tuple:
75          """Compiles the data to a single argument(tuple). Would also be possible to do
            inline.
76              Better to do it in a separate function to have the possibility to add/modify
                the given arguments.
77
78          Args:
79              point (tuple): point with x and y coordinate in range of max_x and max_y
80              calculated_move (str): the calculated move by calculate_best_move()
81
82          Returns:
83              tuple: tuple of ((point), calculated_move)
84          """
85          return (point, calculated_move)
86
87      def create_csv(filename:str) -> None:
88          """Sets up a csv file for the given path or creates one of no file with the given
            path exists.
89
90          Args:
91              filename (str): the path of the csv file. Also works with relative paths
92          """
93          with open(filename, "w", encoding="UTF-8", newline="") as training_file:
94              csvwriter = csv.writer(training_file)
95              csvwriter.writerow(["point", "move"])
96
97      def append_data(filename:str, data:list[tuple, str]) -> None:
98          """Append the given data to the csv file.
99
100         Args:
101             filename (str): the path of the csv file. Also works with relative paths
102             data (list[tuple, str]): the data to append to the given file.
103                 Takes a list of a tuple and string as the point(x, y) and the calculated
                    best move
104         """
105         with open(filename, "a+", encoding="UTF-8", newline="") as training_file:
106             csvwriter = csv.writer(training_file)
107             csvwriter.writerows(data)
108
109     if __name__ == "__main__": # Standard python syntax to test for __name__ ==
        "__main__" to run code in a program only from the program itself
110         point_data:list[tuple, str] = []
111         create_csv(FILENAME)
112         for random_point in get_points(x, y, NUM_OF_START_ENVS):
113             BEST_MOVE = calculate_best_move(random_point) # BEST_MOVE is a constant (for
                each point)
114             point_data.append(compile_data(random_point, BEST_MOVE))
115         append_data(FILENAME, point_data)
116
```