Python Projects

Solved by Michael Zolotarenko

The Great Robot Race 🔪 Specialized Containers

from the Codecademy Intermediate Python 3 course (view Certificate of completion)

Task (view at codecademy.com)

The brushless motors are roaring, and the race is about to begin! In this project, we will be using some of the advanced containers in Python to command, track, and score simple robots which are trying to traverse different mazes. Your job will be to fill in the missing code from the **robot_race.py** file. A lot of background code has been provided in the **robot_race_functions.py** file which we will be using, but the primary focus of this project is to use advanced collections in a meaningful way.

Reviewing the Code

1. Let's start by taking a look at what the mazes files look like!

This one is **maze_data_1.csv**:

```
#,#,#,#,#,#,#,#,#,#
#,_,_,_,#,_,_,_,$,#
#,_,A,_,_,_,#,_,_,#
#,B,_,_,#,_,#,_,_,#
#,_,C,_,#,_,_,_,_,#
#,#,#,#,#,#,#,#,#,#
```

The mazes are csv files which contain different characters that represent different objects in the robot race. Any letter represents a robot, the **#** character represents a wall which the robots can collide with, and the **$** is the goal which the robots are racing towards. Robots can traverse into any empty space (shown by an _) and they can even occupy the same space as another robot (shown as a **+**). At any point during this project, feel free to create your own csv maze and use it in the code instead of the example ones.

2. Now take a look at the **robot_race.py** file. After importing the required modules and classes, there are some values to take note of at the top of the file. The **maze_file_name** can be changed to any csv file which follows the maze structure (as defined earlier). The **seconds_between_turns** value will determine how much time passes between updating the visualized maze in the console (which we will be coding during the project). Finally, the **max_turns** value determines how many turns the robots have before the race ends. The race will end if the max turns are reached or if all of the robots reach the goal.

The robots will be scored based on the amount of moves they make plus the number of collisions they have with the walls. The robot with the lowest score wins.

3. Throughout this project, run the code in the terminal by using the command: **python3 robot_race.py**

Commanding the Bots

4. To begin, we are going to use the provided **compute_robot_logic** function from **robot_race_functions.py** to calculate every move for every robot which has not finished the maze yet. This function accepts the **walls**, **goal**, and **bot** variables in that order and returns a tuple containing the robot name, selected action, and if the robot has collided.

5. The first **while** loop iterates until the race is over. Inside of this, loop through every bot which has not finished the race yet (**bot.has_finished == False**). Pass the **walls**, **goal**, and **bot** to the **compute_robot_logic** function in that order. This will return the robot's decision given its position in the map in the form of a **tuple** containing **(robot_name, action, has_collided)**. Append the robot's decision to the **robot_moves deque**.

Scoring the Bots

6. Now that all robots' moves have been calculated, we can use that data and the **Counter** container to find the exact number of moves that each robot makes using one line of code! This should count every name (the first element in the move **tuple**) for every move in the **deque**.

7. We can use the **Counter** container to count how many collisions each robot made as well. To do this, make sure to only count the robot name when **has_collided** is **True** in each move **tuple** within the **robot_moves deque**.

8. Since we have the move and collision count for each robot, we can now calculate the final scores for each bot. In order to make it easier to read, let's create a **namedtuple** to keep track of our bot score data. Create a **namedtuple** subclass called **BotScoreData** which contains the field names: **'name'**, **'num_moves'**, **'num_collisions'**, and **'score'**.

9. Loop through each of the robots in **bots**, and for every robot, create a new **BotScoreData** object containing all the correct data. Append this new object to the **bot_scores** list. Remember that the score for each robot is found by adding the number of moves and the number of collisions. Try calling the **print_results** function using the **bot_scores** and see what's printed to the console. You should see the final results of the race in the terminal, but now let's watch them actively move through the maze!

Displaying the Live Race

10. To watch the robots travel through the maze, let's start by creating a **dict** which allow us to easily access each robot object by its name when looking at the different moves in the **deque**. Loop through every robot in **bots** and add it to **bot_data** using the robot name as the key and the robot object as the value.

11. Create a loop which continues while moves still exist in the **robot_moves deque**. For every iteration, pop the move from the front of the **deque** and call the **process_move** method on the bot accessed from the **bot_data**. The **process_move** accepts a string which represents an action. This is found in the second element of each move tuple from the **robot_moves deque** (**move[1]**).

The next three lines of code update the character maze with the new robot positions, prints the maze, and pauses the program briefly based on the settings from the top of the script.

12. Finally, remove the call to **rr.print_results(bot_scores)** from earlier in the code and place it at the end of the code to see the final results after the race!

13. In the terminal, type **python3 robot_race.py** in order to watch the robots race and see the final results.

Solution

robot_race.py (download robot-race.tar.gz from GitHub Portfolio repository)

```
import robot_race_functions as rr
from collections import deque, Counter, namedtuple
from time import time, sleep

maze_file_name = 'maze_data_1.csv'
seconds_between_turns = 0.3
max_turns = 35

# Initialize the robot race
maze_data = rr.read_maze(maze_file_name)
rr.print_maze(maze_data)
walls, goal, bots = rr.process_maze_init(maze_data)

# Populate a deque of all robot commands for the provided maze
```

```
robot_moves = deque()
num_of_turns = 0
while not rr.is_race_over(bots) and num_of_turns < max_turns:
    # For every bot in the list of bots, if the bot has not reached the end, add a new move to
the robot_moves deque
    for bot in bots:
        if not bot.has_finished:
            # Call compute_robot_logic function and append the result to the robot_moves deque
            move = rr.compute_robot_logic(walls, goal, bot)
            robot_moves.append(move)

    num_of_turns += 1

# Count the number of moves based on the robot names
move_counts = Counter(move[0] for move in robot_moves)

# Count the number of collisions by robot name
collision_counts = Counter(move[0] for move in robot_moves if move[2])

# Create a namedtuple to keep track of our robots' points
BotScoreData = namedtuple('BotScoreData', ['name', 'num_moves', 'num_collisions', 'score'])

# Calculate the scores (moves + collisions) for each robot and append it to bot_scores
bot_scores = []
for bot in bots:
    score_data = BotScoreData(
        name=bot.name,
        num_moves=move_counts[bot.name],
        num_collisions=collision_counts[bot.name],
        score=move_counts[bot.name] + collision_counts[bot.name]
    )
    bot_scores.append(score_data)

# Populate a dict to keep track of the robot movements
bot_data = {bot.name: bot for bot in bots}

# Move the robots and update the map based on the moves deque
while len(robot_moves) > 0:
    move = robot_moves.popleft()
    bot = bot_data[move[0]]
    bot.process_move(move[1])

    # Update the maze characters based on the robot positions and print it to the console
    rr.update_maze_characters(maze_data, bots)
    rr.print_maze(maze_data)
    sleep(seconds_between_turns - time() % seconds_between_turns)

# Print out the results!
rr.print_results(bot_scores)
```