tensentric

Software Engineer Intern Take Home Project

(2025)

tensentric

Introduction

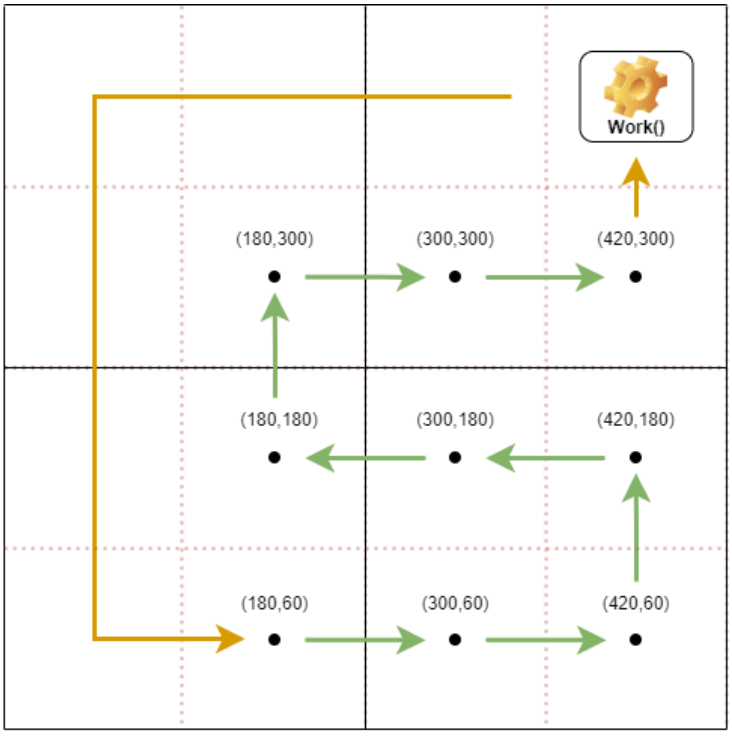
Congratulations! We're excited to have you interview with us. Instead of a traditional technical interview, we'd like to see how you apply your skills in a more relaxed setting. Specifically, we want to understand how you approach complex problems while balancing clean code performance.

General Guidelines

- There is **no** time limit for this. We expect that this portion of the project should take you only a few hours at most.
    - A working solution is **not required.** While we encourage you to try to find a working solution, you will not be disqualified, or not considered for an internship should you not achieve a working solution.
- This project is stack agnostic. While we primarily use C#, C/++ at Tensentric, you are welcome to use whatever programming language you are most comfortable with.
- Be prepared to explain your solution and walk us through how you approached and solved the problem you chose.
- You **are allowed** to use the internet.
    - You are **NOT** allowed to post this question or ask for help directly on StackOverflow, LeetCode, or any other form of programming forum.
- You are allowed to use common libraries to achieve your solution.
    - This includes using things like predicate logic (LINQ, Streams, etc), standard libraries (stdio, stddef, etc).
- While we support teamwork, you should work on this project alone.
- You should use object-oriented practices and design patterns, if applicable.
- Please provide tests, console output, or some other form of validation that you have completed the task.
- Your code must compile with **zero** build errors.

1 The Problem

This problem is one that we encountered on an actual project at Tensentric. Review the following grid and points.



This grid is representative of a surface that *can* support a 4x4 array of what we will call **Pucks**. This surface can magnetically levitate the pucks and move them around. Each section on the grid represents a **parking spot** for each puck, denoted by the red dotted line. The arrows between points represent the path each puck must take from **(180, 60)** to **(420, 300)**. Only one puck can exist in each parking spot at a given time.

The total area of the surface is 480mm x 480mm and the pucks can start at **any** coordinate in this area.

On initialization a maximum of 9 pucks, minimum of 1, can be on the surface, and each need to be assigned to a parking spot. The initial position of the puck can be queried and may cross parking spot boundaries.

1.1 Your Task

Query each random puck location after **Initialize()**, and then assign it to its closest *unoccupied* parking spot. You should keep track of each puck and its current location. There should be no gaps between puck locations before you begin performing **Work().** You should perform the **Work()** function after you pop the puck at the head of the parking spots. Once popped, you should move all the previous pucks to their next position. This can be done *while* the **Work()** function is being done. When **Work()** is completed, you should move the puck to the tail of the parking spots (where there should now be an empty spot after all the other pucks move forward). Your task is completed when all the pucks have processed through the **Work()** task and returned to their original parking position.

1.2 Assumptions you can make & Hints

- There will always be between 1 and 9 pucks on the system, no more and no less.
- The first of the parking spots is **(420, 300)** – The next to perform work.
- The last of the parking spots is **(180,60) –** The last to perform work.
- There should be **no gaps** between pucks.
- This project can be done in at *least* $O(n^2)$ time complexity.
    - Since there is a known queue size, don't focus too much on time complexity. It is negligible for this project.
- No two pucks can exist at the same point at the same time.
- You can assume that **Work()** is asynchronous.
- You do not have to worry about collisions.
    - When you re-queue after **Work()** is finished, you can assume the puck will follow the path laid out in orange when calling **Move()**.
    - Each puck can only exist in one parking spot once assigned, but the puck can be considered as a single point.
- The origin is at **(0,0)** in the bottom left corner of the surface.

1.3 Pseudo-API

To help you succeed, we've provided a pseudo-code API. This is just a template, and by no means are you required to follow it. This is meant to help you spark some ideas.

```
using math;
using random;

public class Puck():
    public int ID {get; set;}
    public boolean HasWorked {get; set;}
    Puck(int id)
    {
        this.id = id;
        this.HasWorked = false;
    }

class PuckLibrary():
    // Basic data structures
    // All the valid parking spots
    public List<(int,int)> parkingSpots =
      [(180,60),(300,60),(420,60),(420,180),(300,180),
      (180,180),(180,300),(300,300),(420,300)]
    public List<Puck> pucks = new();

    // Some functions to implement in your own library
    public PuckLibrary();
    public void Initialize()
    {
        // Generate random number of pucks
        pucks = new List<Puck>(random.randint(1,9));
        foreach Puck p in pucks:
            // Assign a random coordinate position between 0-480 for both X and Y
            // Assert no two pucks are at the same coordinate!
    }

    // Move function to move a puck to a position
    public void Move(pos);

    // Work function, doesn't have to do anything special
    public void Work(Puck puckObj);

    private double dist(pos1, pos2);

class MyImplementation():
    // Ideas for some functions to create that utilize Puck and PuckLibrary
    // park all pucks to their closest unoccupied parking spot

    public void ParkPucks(); // ensure there are no unoccupied spots between pucks
    public void CloseGaps(); // Move and perform the work on all the pucks
    public void MoveAndPerformWork();
```

1.5 Ideas for Output

```
Initial Position of puck 0 set to (478, 238)
Initial Position of puck 1 set to (134, 200)
Moving puck 1 to (180, 180)
Moving puck 0 to (420, 180)

Initial Configuration:
START -> EMPTY -> EMPTY -> EMPTY -> FILLED (Puck: 0) -> EMPTY -> FILLED (Puck: 1) -> EMPTY -> EMPTY -> EMPTY -> END

Close gaps:
Moving puck 1 to (420, 300)
START -> EMPTY -> EMPTY -> EMPTY -> FILLED (Puck: 0) -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> FILLED (Puck: 1) -> END
Moving puck 0 to (300, 300)

No gaps List:
START -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> FILLED (Puck: 0) -> FILLED (Puck: 1) -> END
=== BEGIN WORK ===
Performing work on: 1!
Moving puck 0 to (420, 300)
START -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> FILLED (Puck: 0) -> END

Moving puck 1 to (180, 60)
START  -> FILLED (Puck: 1) -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> FILLED (Puck: 0) -> END
Moving puck 1 to (300, 300)
START -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> FILLED (Puck: 1) -> FILLED (Puck: 0) -> END

Performing work on: 0!
Moving puck 1 to (420, 300)
START -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> FILLED (Puck: 1) -> END
Moving puck 0 to (180, 60)
START -> FILLED (Puck: 0) -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> FILLED (Puck: 1) -> END
Moving puck 0 to (300, 300)

START -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> FILLED (Puck: 0) -> FILLED (Puck: 1) -> END

Final Configuration (Start Configuration == End Configuration):
START -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> FILLED (Puck: 0) -> FILLED (Puck: 1) -> END
Initial Configuration (After gap closing):
START -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> EMPTY -> FILLED (Puck: 0) -> FILLED (Puck: 1) -> END
```

You're welcome to format the output of your program however you like. However, it should be clear what the program is doing, and also have some sort of proof that the pucks have finished performing **Work()**.

We know this might feel overwhelming, so if anything doesn't make sense feel free to reach out; we're here to help!