ECE 250 -Project0

Potential Fields

Design Document

Abdullah Al Amaan, UW UserID=aaamaan

September 23rd, 2024

Class Design:

In this project, I implemented two primary classes:

Vector2D:

Purpose: This class represents a 2D vector with x and y components, which store the potential values in each grid cell of the map.

Data Structures:

- **double x, y**: These variables represent the potential in the x and y directions.

Why I Used This: Since each cell of the grid stores a vector representing the potential in two directions, using a structure that directly holds x and y values makes it clear and efficient access, without any additional functionality.

Design Decisions: The only purpose of this class is to hold and manipulate numerical data. I used public data members for this class as it allows direct access when updating potential values during calculations.

PotentialField:

Purpose: This class represents the map for the robot's environment, handling dynamic memory allocation and potential field calculations.

Data Structures:

- **Vector2D** map:** A dynamically allocated 2D array that holds the potential vectors for each cell. I chose to use a dynamically allocated array since the size of the grid is determined at runtime, and using a vector library was not allowed by the project constraints.
- **int N, M:** These variables store the dimensions of the grid (N for rows and M for columns). Storing the grid dimensions as private members helps control and protect the grid size throughout the class.

- **double K:** a scalar constant that establishes the potential field's strength. To isolate the system settings, I made this a private variable, and I can only access it through the updateK() function.

Why I Used These Data Structures:

The 2D array of Vector2D objects was selected for the flexibility it offers in managing a grid of any size while adhering to the project's constraints of not using STL containers.

Design Decisions:

I kept most variables and functions private to protect the internal state of the potential field system, ensuring that access to the map and constant K are controlled through member functions.

I avoided getters and setters method but ensured that any important updates to k were done through a function that checks the validity of the new value.

Function Design:

- **createMap(int n, int m):**

This function handles the dynamic allocation of the map and its initialization. I chose to encapsulate the grid allocation in this function to allow the map to be resized or reset easily without duplicating code. The runtime for this function is O(N × M) since every cell in the grid is initialized with default values.

- **addPoint(char type, int x, int y):**

This function recalculates the potential field based on whether a goal or obstacle is added. The potential is calculated using the formula provided in the project description. The runtime is O(N × M) because every cell in the map needs to have its potential updated based on the distance to the point being added.

- **moveRobot(int x, int y):**

This function returns the potential at a specific location. The runtime is O(1) since it only retrieves the calculated values from the map.

- **clearMap():**

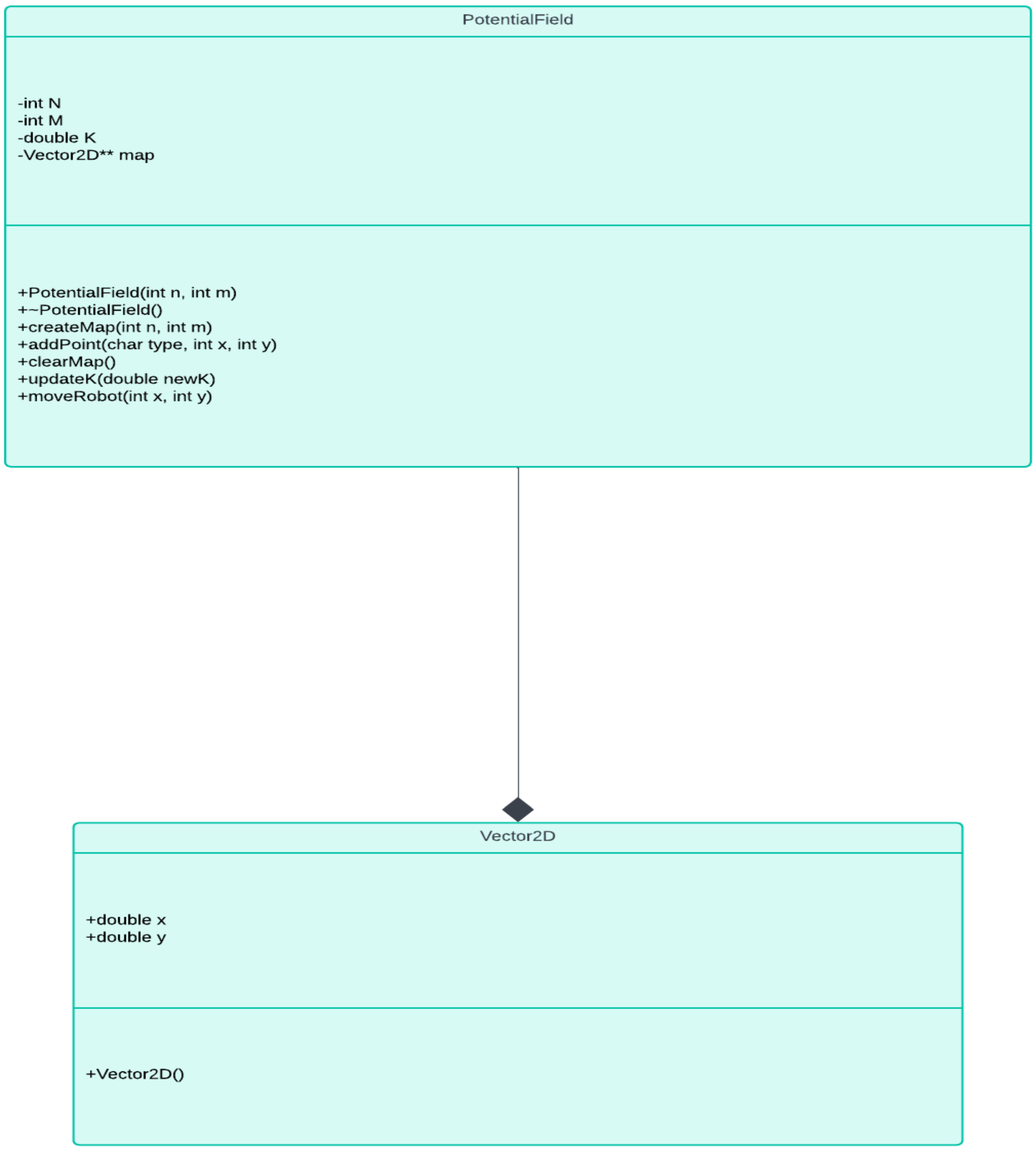
This function resets the potential values in the grid. The runtime is O(N × M) as it iterates over every cell in the grid to reset the values.

Constructors and Destructors:

- **Constructor:** The constructor of PotentialField uses createMap() to handle dynamic memory allocation. This was chosen to simplify memory management by having a single function responsible for both allocation and initialization.

- Destructor: I implemented a destructor to ensure all dynamically allocated memory is properly freed, avoiding memory leaks. This is crucial since the grid is dynamically allocated using new[].

UML Diagram:



Conclusion:

This design was chosen to balance the constraints of dynamic memory management while avoiding the use of STL containers. Flexibility is offered by the 2D array structure, and the security of the system remains intact by enclosing functionality inside well-defined classes.