Student Name: VexProject_Siyuan_Aston_Cean_Leo

Assignment:

Notes:

Project Name: VEXcode Project

Project Type: C++

Date: Wed Mar 26 2025

```cpp
#include <iostream>
#include "vex.h" // Ensure this includes the VEX Robotics API

using namespace std;
using namespace vex;

brain Brain;
motor left_motor(PORT1, false);
motor right_motor(PORT6, true);
inertial imu = inertial();
vex::distance DistanceSensor(PORT7); // Distance sensor
drivetrain Drivetrain(left_motor, right_motor, 200, 320, 40, mm, 1);

// Node structure for doubly linked list
struct Node {
    int x, y;
    bool isNecessary;
    Node* prev;
    Node* next;

    Node(int xPos, int yPos, bool necessary) {
        x = xPos;
        y = yPos;
        isNecessary = necessary;
        prev = nullptr;
        next = nullptr;
    }
};

// Doubly Linked List for Path Storage
class DoublyLinkedList {
private:
    Node* head;
    Node* tail;
    int size;
    int capacity;

public:
    DoublyLinkedList(int cap = 10) {
        head = nullptr;
        tail = nullptr;
        size = 0;
        capacity = cap;
    }

    ~DoublyLinkedList() {
        while (head) {
            Node* temp = head;
            head = head->next;
            delete temp;
        }
    }

```

```
54          void insert(int x, int y, bool isNecessary) {
55              Node* newNode = new Node(x, y, isNecessary);
56              if (!head) {
57                  head = tail = newNode;
58              } else {
59                  tail->next = newNode;
60                  newNode->prev = tail;
61                  tail = newNode;
62              }
63              size++;
64
65              if (size > capacity) {
66                  removeUnnecessaryNodes();
67              }
68          }
69
70          void removeUnnecessaryNodes() {
71              Node* current = head;
72              while (current && size > capacity) {
73                  if (!current->isNecessary) {
74                      Node* toDelete = current;
75                      current = current->next;
76                      if (toDelete == head) head = head->next;
77                      if (toDelete == tail) tail = tail->prev;
78                      if (toDelete->prev) toDelete->prev->next = toDelete->next;
79                      if (toDelete->next) toDelete->next->prev = toDelete->prev;
80                      delete toDelete;
81                      size--;
82                  } else {
83                      current = current->next;
84                  }
85              }
86          }
87
88          void print() {
89              Node* current = head;
90              while (current) {
91                  cout << "(" << current->x << ", " << current->y << ")";
92                  if (current->isNecessary) cout << " [Necessary]";
93                  cout << " -> ";
94                  current = current->next;
95              }
96              cout << "NULL" << endl;
97          }
98      };
99
100     // Function to calibrate IMU
101     void cali_inertial() {
102         left_motor.setVelocity(15, rpm);
103         right_motor.setVelocity(15, rpm);
104
105         imu.calibrate();
106         while (imu.isCalibrating()) {}
```

```
107
108         wait(1, seconds);
109         Drivetrain.turn(turnType::right);
110         wait(400, msec);
111
112         while (imu.angle(degrees) < 166.0) {}
113
114         Drivetrain.stop(brake);
115         wait(1, seconds);
116     }
117
118     // Function to check for obstacles
119     bool detectObstacle() {
120         double dist = DistanceSensor.objectDistance(mm);
121         cout << "Distance Sensor Reading: " << dist << " mm" << endl;
122         return (dist > 0 && dist <= 40); // Check for valid distance
123     }
124
125     // Path Planning Algorithm
126     void pathPlanning(DoublyLinkedList &path) {
127         int x = 0, y = 0;
128         int targetX = 18, targetY = 18;
129
130         cali_inertial(); // Step 1: IMU Calibration
131
132         path.insert(x, y, true); // Insert starting position
133
134         while (x != targetX || y != targetY) {
135             bool moved = false;
136
137             // Movement Priority
138             if (abs(x - targetX) > abs(y - targetY)) {
139                 // Move East first
140                 if (!detectObstacle()) {
141                     x++;
142                     Drivetrain.driveFor(vex::fwd, 20, mm);
143                     moved = true;
144                 }
145             } else {
146                 // Move North
147                 if (!detectObstacle()) {
148                     y++;
149                     Drivetrain.driveFor(vex::fwd, 20, mm);
150                     moved = true;
151                 }
152             }
153
154             // If movement was successful, insert into path
155             if (moved) {
156                 path.insert(x, y, false);
157                 cout << "Moved to: (" << x << ", " << y << ")\n";
158             } else {
159                 // Obstacle detected, turn and move 3-4 units away
```

```
160                 cout << "Obstacle detected! Turning...\n";
161                 Drivetrain.turn(turnType::right);
162                 wait(500, msec);
163                 for (int i = 0; i < 4; i++) {
164                     x++; // Move away from the obstacle
165                     path.insert(x, y, true);
166                     Drivetrain.driveFor(vex::fwd, 20, mm);
167                     wait(200, msec);
168                 }
169             }
170 
171             wait(500, msec);
172         }
173 
174         cout << "Destination Reached!\n";
175         path.print();
176     }
177 
178     // Main Function
179     int main() {
180         DoublyLinkedList path(10);
181         pathPlanning(path);
182         return 0;
183     }
184 
```