

```

1  /**
2   * @file main.cpp
3   * @author Christian Prather
4   * @brief Testing algorithm for the optimization algorithm
5   * @version 0.1
6   * @date 2020-11-12
7   *
8   * @copyright Copyright (c) 2020
9   *
10  */
11 #include <iostream>
12 using namespace std;
13 /// Enum defines
14 #define FORWARD 0
15 #define RIGHT 1
16 #define LEFT 2
17 #define DISTANCE_SEG 10
18
19 int movesCount = 6;
20 // Global array for tracking move order (move, distance) or (move, degree)
21 int moveList[50] = {FORWARD, FORWARD, RIGHT, RIGHT, FORWARD, FORWARD};
22
23 int optimizedMoves[50];
24
25 void optimize()
26 {
27     /// Key patterns 0 = F, 1 = R, 2 = L, 3 = DELETE
28     int keyPatterns_6[2][6] = {{0, 0, 1, 1, 0, 0}, {2, 0, 1, 1, 0, 2}};
29     int keyPatterns_5[2][5] = {{2, 0, 1, 1, 0}, {0, 1, 1, 0, 2}};
30     int keyPatterns_4[1][4] = {{0, 1, 1, 0}};
31
32     int optimizedPattern_6[1][8] = {{FORWARD, 2 * DISTANCE_SEG, RIGHT, 90,
33     RIGHT, 90, FORWARD, DISTANCE_SEG}};
34     int optimizedPattern_5[2][2] = {{RIGHT, 90}, {RIGHT, 90}};
35     int optimizedPatter_4[1][4] = {{LEFT, 90, LEFT, 90}};
36     /** This is going to be checking in a priority tree fashion given highest
37     priority
38     * given highest priority patterns are 6 long then 5 long then 4 I can
39     batch this
40     */
41     for (int i = 0; i < movesCount; i++)
42     {
43         /// Get next move in explored list
44         /// int move = moveList[i];
45         /// Get next 6 moves if enough in list
46
47         // Check 6 out first
48         int future[6];
49         for (int j = 0; j < 6; j++)
50         {
51             if ((j + i) < movesCount)
52             {
53                 /// j (0-5) i (0-movesCount)
54                 future[j] = moveList[j + i];
55                 cout << "Move: " << future[j] << endl;
56             }
57         }
58         int tracker = 0;
59         for (auto potential : keyPatterns_6)

```

```

58     {
59         bool match = true;
60         for (int m = 0; m < 6; m++)
61         {
62             if (future[m] != potential[m])
63             {
64                 match = false;
65             }
66         }
67         if (match)
68         {
69             cout << "Matched " << tracker << endl;
70             int keyPatternLength = (sizeof(potential) /
sizeof(potential[0]));
71             // Insert optimized move
72             for (int x = 0; x < (sizeof(optimizedPattern_6[tracker]) /
sizeof(optimizedPattern_6[tracker][0])); x++)
73             {
74                 if (optimizedPattern_6[tracker][x] != 3)
75                 {
76                     optimizedMoves[x] = optimizedPattern_6[tracker][x];
77                 }
78             }
79             i = i + 6;
80             break;
81         }
82         tracker = tracker + 1;
83     }
84
85     ////////////////////////////////////////
86     // Check 5 out first
87     int future_5[5];
88     for (int j = 0; j < 5; j++)
89     {
90         if ((j + i) < movesCount)
91         {
92             /// j (0-5) i (0-movesCount)
93             future_5[j] = moveList[j + i];
94             cout << "Move5: " << future_5[j] << endl;
95         }
96     }
97     tracker = 0;
98     for (auto potential : keyPatterns_6)
99     {
100         bool match = true;
101         for (int m = 0; m < 5; m++)
102         {
103             if (future_5[m] != potential[m])
104             {
105                 match = false;
106             }
107         }
108         if (match)
109         {
110             cout << "Matched " << tracker << endl;
111             int keyPatternLength = (sizeof(potential) /
sizeof(potential[0]));
112             // Insert optimized move

```

```

113         for (int x = 0; x < (sizeof(optimizedPattern_6[tracker]) /
sizeof(optimizedPattern_6[tracker][0])); x++)
114         {
115             if (optimizedPattern_6[tracker][x] != 3)
116             {
117                 optimizedMoves[x] = optimizedPattern_6[tracker][x];
118             }
119         }
120         i = i+ 5;
121         break;
122     }
123     tracker = tracker + 1;
124 }
125
126
127 ////////////////////////////////////////////////////
128 //
129 // Check 4 out first
130 int future_4[4];
131 for (int j = 0; j < 4; j++)
132 {
133     if ((j + i) < movesCount)
134     {
135         /// j (0-5) i (0-movesCount)
136         future_4[j] = moveList[j + i];
137         cout << "Move4: " << future_4[j] << endl;
138     }
139 }
140 tracker = 0;
141 for (auto potential : keyPatterns_6)
142 {
143     bool match = true;
144     for (int m = 0; m < 4; m++)
145     {
146         if (future_4[m] != potential[m])
147         {
148             match = false;
149         }
150     }
151     if (match)
152     {
153         cout << "Matched " << tracker << endl;
154         int keyPatternLength = (sizeof(potential) /
sizeof(potential[0]));
155         // Insert optimized move
156         for (int x = 0; x < (sizeof(optimizedPattern_6[tracker]) /
sizeof(optimizedPattern_6[tracker][0])); x++)
157         {
158             if (optimizedPattern_6[tracker][x] != 3)
159             {
160                 optimizedMoves[x] = optimizedPattern_6[tracker][x];
161             }
162         }
163         i = i+4;
164         break;
165     }
166     tracker = tracker + 1;
167 }
168 }
169 }

```

```
168
169 int main()
170 {
171     optimize();
172     cout << "Optimized" << endl;
173     for (auto move : optimizedMoves)
174     {
175         cout << move << ", ";
176     }
177 }
```