```
1   //================================================================================
2   // Path Finder!
3   //================================================================================
4   boolean pathFindFromCurrent(int targCol, int targRow) {
5
6     pathTargetCol = targCol;
7     pathTargetRow = targRow;
8
9     boxForgive = false;
10
11    //(SectorPath){ nextPoint.col, nextPoint.row, firstPoint.inHeading,
    firstPoint.outAction };
12    subTarget = (SectorPath) {
13      valueToSectorMM( x ), valueToSectorMM( y ), thetaState, ACTION_NONE
14    };
15
16    boolean success = pathFindRecursive( &subTarget );
17    worldPassReset();
18
19    if (!success) {
20      boxForgive = true;
21      subTarget = (SectorPath) {
22        valueToSectorMM( x ), valueToSectorMM( y ), thetaState, ACTION_NONE
23      };
24      success = pathFindRecursive( &subTarget );
25      worldPassReset();
26    }
27
28    Serial.print("Subtarget: "); Serial.print(subTarget.col); Serial.println( subTarget.row);
29
30    if (subTarget.col > 5)
31      subTarget.col = 4;
32    else if (subTarget.col < 0)
33      subTarget.col = 0;
34
35    if (subTarget.row > 5)
36      subTarget.row = 4;
37    else if (subTarget.row < 0)
38      subTarget.row = 0;
39
40    Serial.print("\t Modified Subtarget: "); Serial.print(subTarget.col); Serial.println
    (subTarget.row);
41
42    return success;
43  }
44
45  boolean pathFindRecursive(SectorPath *thisSector) {
46
47    //Check if this is even a valid position
48    if ( !inSectors(thisSector->col, thisSector->row) || getSectorPass(thisSector->col,
    thisSector->row) == PASSED_BLOCKED ) {
49      resetSectorPassPlan(thisSector->col, thisSector->row);
50      return false;
51    }
52
53    //Check if we hit the target
54    if ( thisSector->col == pathTargetCol && thisSector->row == pathTargetRow) {
55      //If we did, set this action as STOP
56      thisSector->outAction = ACTION_STOP;
57      //Set our subtarget to this
58      subTarget.col = thisSector->col;
59      subTarget.row = thisSector->row;
60      resetSectorPassPlan(thisSector->col, thisSector->row);
61      //Spread the word!
62      return true;
63    }
64
65    //Mark our postion on the pass grid
66    setSectorPassPlan( thisSector->col, thisSector->row);
```

```
 67
 68     boolean success = false;
 69
 70     //These track the direction indicies
 71     //Since we came from one direction,
 72     //it naturally follows that there are only three we can take
 73     //Thus, three variables
 74     byte headings[3] = {255, 255, 255};
 75
 76     //And we only need to track the low and mid scores
 77     //Since we only care about their positions relative to each other;
 78     byte scores[3] = {255, 255, 255};
 79     byte genScore = 255;
 80
 81     //Now, probe our four options:
 82     for (int probeHead = 0; probeHead < THETA_INCREMENTS; probeHead++) {
 83
 84       genScore = headingScore(
 85                    thisSector->col + projectCol(probeHead), thisSector->row + projectRow
    (probeHead),
 86                    thisSector->col, thisSector->row, thisSector->inHeading, probeHead
 87                  );
 88
 89       //if the score is valid
 90       if ( genScore != 255 ) {
 91         //if it's lower than the lowest, or equal
 92         if ( genScore == scores[0] || min(genScore, scores[0]) == genScore ) {
 93           //boot the middle to the highest
 94           headings[2] = headings[1]; scores[2] = scores[1];
 95           //boot the lowest to middle
 96           headings[1] = headings[0]; scores[1] = scores[0];
 97           //set the lowest
 98           headings[0] = probeHead; scores[0] = genScore;
 99         }
100         //else, if it's lower than the middle, or equal
101         else if ( genScore == scores[1] || min(genScore, scores[1]) == genScore ) {
102           //boot the middle to the highest
103           headings[2] = headings[1]; scores[2] = scores[1];
104           //set the middle
105           headings[1] = probeHead; scores[1] = genScore;
106         }
107         //else, if it's lower than the highest, or equal
108         else if ( genScore == scores[2] || min(genScore, scores[2]) == genScore )
109           //set the highest
110           headings[2] = probeHead; scores[2] = genScore;
111       }
112     }
113
114     SectorPath nextSector = (SectorPath) {
115       0, 0, 0, ACTION_NONE
116     };
117
118     for (int i = 0; i < 3 && !success; i++) {
119       if (headings[i] != 255 && scores[i] != 255) {
120         thisSector->outAction = headingsToAction( thisSector->inHeading, headings[i] );
121
122         nextSector.col = thisSector->col + projectCol( headings[i] );
123         nextSector.row = thisSector->row + projectRow( headings[i] );
124         nextSector.inHeading = actionToNewHeading( thisSector->inHeading, thisSector-
    >outAction );
125         nextSector.outAction = ACTION_NONE;
126
127         success = pathFindRecursive( &nextSector );
128       }
129     }
130
131     //If we succeeded
132     if (success)
133       //And this sector's action is a waypoint action
```

```
134       if (thisSector->outAction >= ACTION_STOP)
135         //And its not the start sector
136         if ( thisSector->col != valueToSectorMM( x ) || thisSector->row != valueToSectorMM
      ( y ) ) {
137           #ifdef DEBUG
138           Serial.println("subtargetReassign"); Serial.print(thisSector->col); Serial.print
      (thisSector->row);
139           #endif
140           subTarget.col = thisSector->col; subTarget.row = thisSector->row;
141         }
142
143     //Finally, undo the marking so we can use the pass grid later
144     resetSectorPassPlan(thisSector->col, thisSector->row);
145
146     return success;
147   }
148
149   byte headingScore(int col, int row,  int currCol, int currRow, int oldHeading, int
      newHeading) {
150     //If a path isn't possible, reject it;
151     if ( !isPathPossible( col, row ) )
152       return 255;
153
154     //if we are considering high probability squares dangerous, and this is high prob,
      reject.
155     if ( !boxForgive && statGrid[ row ][ col ] >= 0.50)
156       return 255;
157
158     //Base score
159     byte score = 1;
160
161     //If the parities of the heading don't match, that means we have to turn - that's worth
      two points.
162     if ( (oldHeading % 2) != (newHeading % 2) )
163       score += 2;
164
165     //If the sector is an unknown, we multiply the blockage certainty by 10 and add it to
      the score
166     if ( boxForgive && passGrid[ row ][ col ] == PASSED_UNSURE )
167       score += byte( 10 * statGrid[ row ][ col ] );
168
169     //If the sector is an unknown, and we are in GoHome or GoIncline, unknown spaces cost
      more
170     if ( machineState == GOHOME && passGrid[ row ][ col ] != PASSED_OPENED )
171       score += 3;
172
173     //If, relative to our last position, we're moving away from the goal, we add 4.
174     if ( abs(pathTargetCol - currCol) - abs(pathTargetCol - col ) < 0  ||
175          abs(pathTargetRow - currRow) - abs(pathTargetRow - row ) < 0 )
176       score += 4;
177
178     return score;
179   }
```