プログラミング演習 IIIA
robots

21-403 伊藤幸太郎

2024 年 8 月 19 日

第 1 章

プログラムの仕様

1.1　遊び方・ルール

robots とは、プレイヤーを追いかけて殺すようにプログラムされたロボットから逃げ、ロボット同士や障害物と衝突させて破壊していくゲームである。

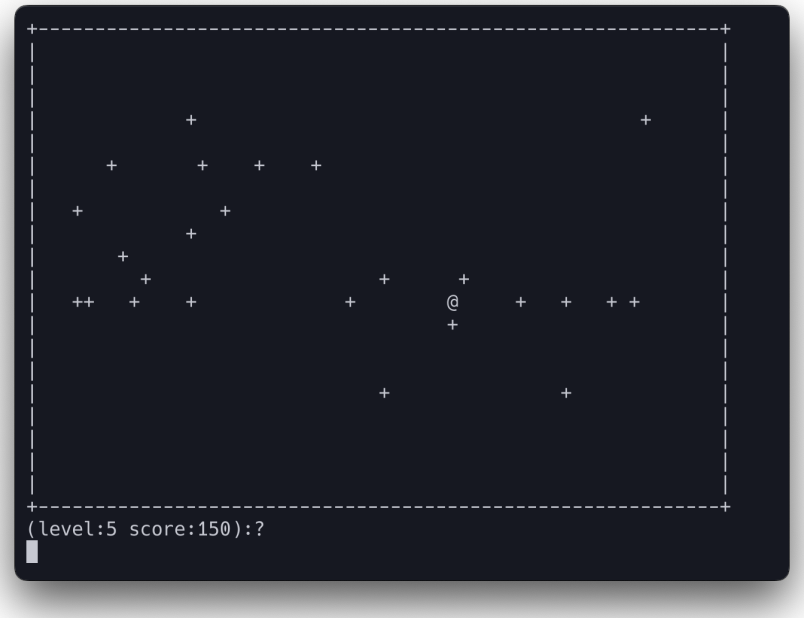

図 1.1　プレイ中の様子

プレイヤーは、1 回の動作で 8 方向（上下左右と斜め）のいずれかに 1 ます移動することができ、「@」で表される。まったく動かないこと、どこかにテレポートする（行き先はランダムに決まる）ことも選択可能である。

7 左上	8 上	9 右上
4 左	5 待機	6 右
1 左下	2 下	3 右下
0 テレポート		

図 1.2　操作方法

プレイヤーが動くと、次にロボットが移動する。ロボットも 8 方向のいずれかに 1 ます移動でき、「+」で表される。ロボットは、常にプレイヤーに最も接近する方向に移動する。

ロボット同士が衝突すると、壊れてスクラップになる。また、ロボットがスクラップに衝突した場合も、そのロボットは壊れる。スクラップは「*」で表される。

プレイヤーがロボットに捕まるとゲームオーバである。フィールド上のすべてのロボットが壊れると、次のフィールドが現れ、レベルが上がる。ロボットの数はレベルに応じて変化する。具体的には、「レベル ×5」と「40」を比較し、小さい方の数をとってその台数分配置される。

ゲーム内のスコアについては、ロボットを 1 台破壊する事に 1 点が与えられる。また、ロボットを全滅させる事にレベル ×10 のボーナスが与えられる。

例えば、最初のフィールドをクリアすると、

$$5(\text{ロボットの数}) + 1(\text{レベル}) \times 10 = 15(\text{点})$$

が得られる

1.2　データ構造

このプログラムでは、構造体及び構造体の配列を使用してプレイヤーとロボットの座標データを管理している。具体的には、`int` 型の要素 `x`, `y` を持つ構造体 `position` を用いて管理している。ロボットは構造体の配列で表しており、要素数はロボットの最大数である 40 に固定している。

1.3　各関数の仕様

1.3.1　getChar()

■書式　`extern char getChar(void);` (getchar.c のオブジェクトファイルとリンクして使用)

■戻り値　入力された文字

■処理内容　標準入力から 1 文字入力し、入力があると直ちに戻る。入力文字はエコーしない。
(この関数は資料内に予め用意されたものである。)

1.3.2 min()

■書式 `int min(int a, int b);`

■引数 比較したい int 型の 2 つの数値

■戻り値 a と b を比較して小さい方の数値

■処理内容 a と b を比較し、小さい方を返す。

1.3.3 shuffle()

■書式 `void shuffle(int *array, int size);`

■引数 シャッフルしたい配列とそのサイズ

■処理内容 array の中身をシャッフルする。

1.3.4 initialize()

■書式 `void initialize(position *player, position robots[], int robots_num, int *array);`

■引数 プレイヤー・ロボットの座標、ロボットの数、座標を決定するための配列

■処理内容 初期のプレイヤー・ロボットの座標をランダムに決定する

1.3.5 print_board()

■書式 `void print_board(position player, position robots[], int robots_num, int level, int score);`

■引数 プレイヤー・ロボットの座標、ロボットの数、レベルとスコア

■処理内容 現在の盤面、レベル、スコアを表示する。

1.3.6 action()

■書式 `int action(position *player, position robots[], int robots_num);`

■引数 プレイヤー・ロボットの座標、ロボットの数

■戻り値 正常に移動が完了したら 0、そうでなければ 1

■処理内容 入力内容に応じてプレイヤーを移動させる。

1.3.7 move_robots()

■書式 `void move_robots(position player, position robots[], int robots_num, int *score);`

■引数 プレイヤー・ロボットの座標、ロボットの数、スコア

■処理内容 プレイヤーの動きに応じてプレイヤーに最も接近する方向にロボットを移動させる。移動後、`check_robots_collision()` を呼び出し、衝突しているか確認する。

1.3.8 check_collision()

■書式 `int check_collision(position player, position robots[], int robots_num);`

■引数 プレイヤー・ロボットの座標、ロボットの数

■戻り値 プレイヤーがロボットに追いつかれたら 1、そうでなければ 0

■処理内容 プレイヤーがロボットに衝突しているか判定する。

1.3.9 compare_positions()

■書式 `int compare_positions(const void *a, const void *b);`

■引数 比較したい 2 つの座標

■処理内容 a, b の x 座標が異なる場合、a と b の x 座標の差を返す。a, b の x 座標が同じ場合、a と b の y 座標の差を返す。
（`qsort()` を使用するために作成）

1.3.10 check_robots_collision()

■書式 `void check_robots_collision(position robots[], int robots_num, int *score);`

■引数 ロボットの座標・数、スコア

■処理内容 ロボットの衝突を判定する。衝突していたらスコアを加算する。

1.3.11 check_level_clear()

■書式 `int check_level_clear(position robots[], int robots_num);`

■引数 ロボットの座標・数

■戻り値 すべてのロボットが死んでいたら 0、そうでなければ 1

■処理内容 フィールド上に生きているロボットが存在するか判定

第 2 章

プログラムの正しさの検証

2.1　衝突処理

例えば以下のような場合、

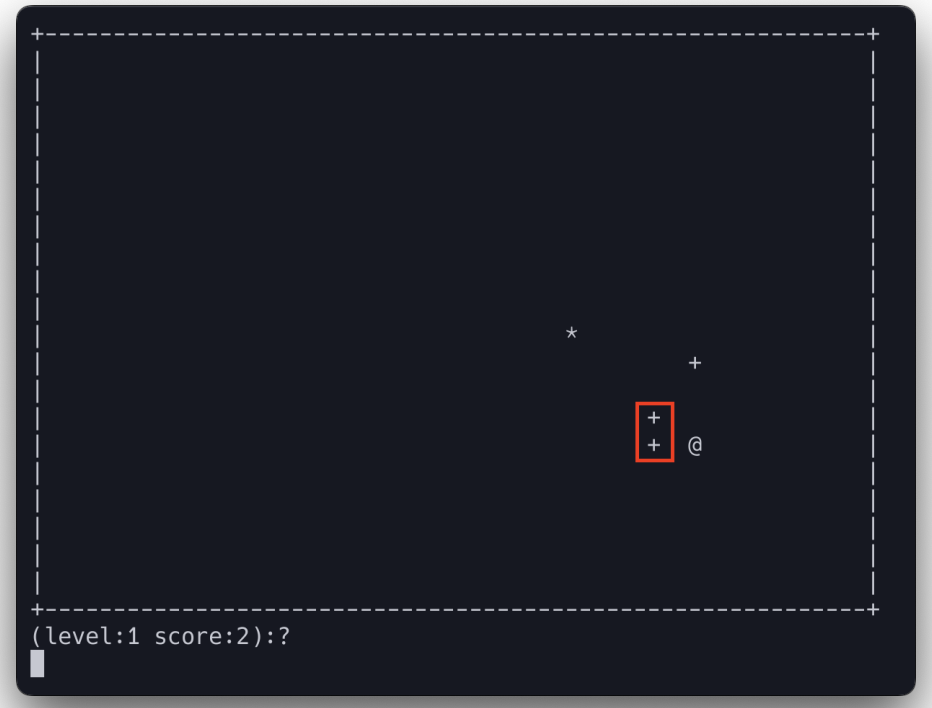

図 2.1　衝突前

プレイヤーが右に移動すると、赤枠で囲ったロボット同士がプレイヤーに接近しようとして衝突する。実際に衝突してスクラップになった様子は以下の通りである。

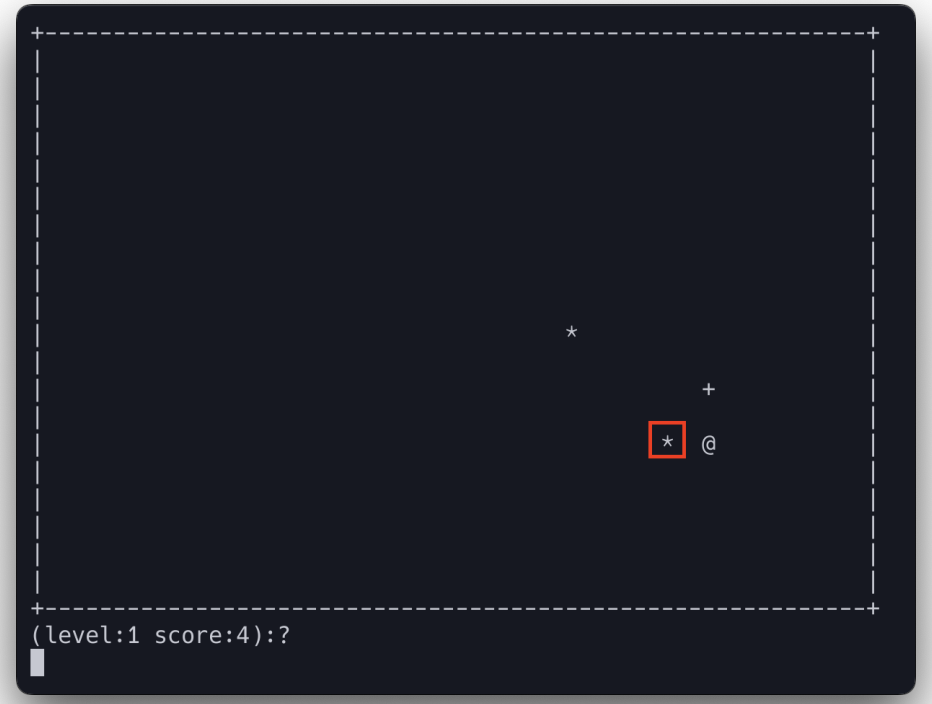

図 2.2　衝突後

2.2　レベルアップ・ゲームオーバー処理

2.2.1　レベルアップ処理

以下のような場合、

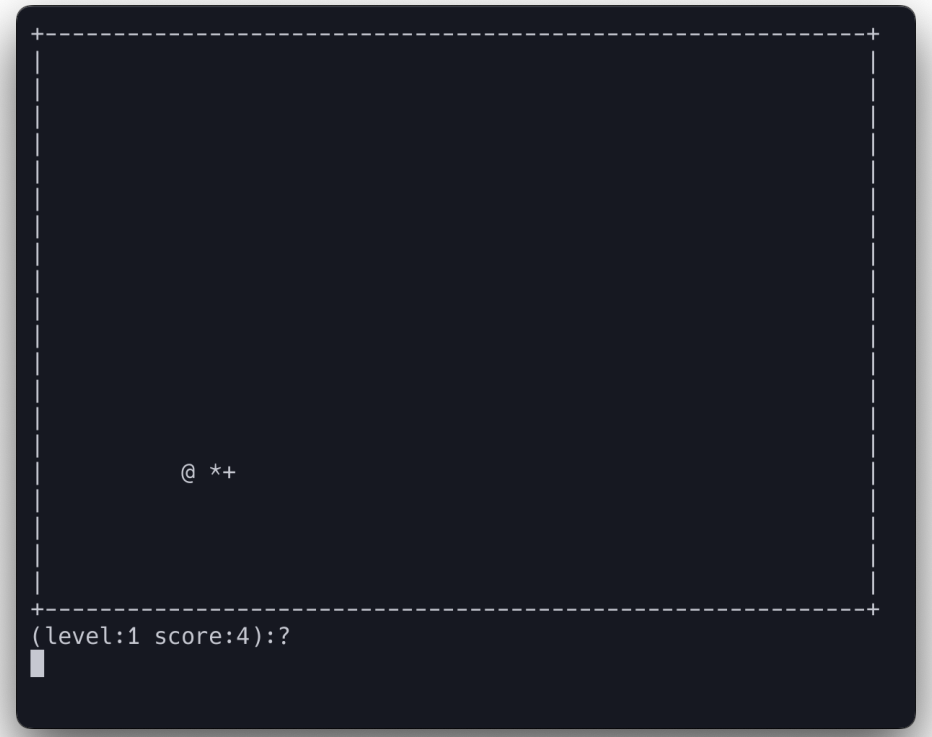

図 2.3　レベルアップ一歩手前

最後のロボットがスクラップに当たれば、そのレベルをクリアできる。ここで、「待機」を選択し、ロボットがスクラップに当たってレベルをクリアし、次の盤面が表示された様子を以下に示す。

図 2.4　レベルアップ後

2.2.2　ゲームオーバー処理

以下のような場合、

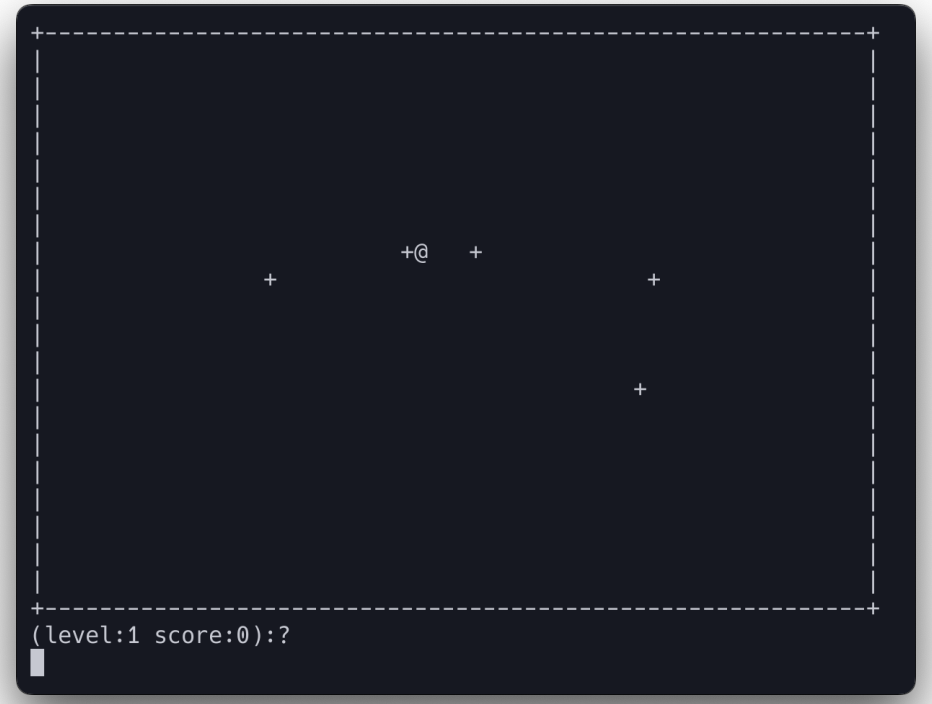

図 2.5　ゲームオーバー一歩手前

例えば、プレイヤーが下に移動すると、移動先はロボットの移動できる範囲内と被っているため、ゲームオーバーとなる。実際の様子を以下に示す。

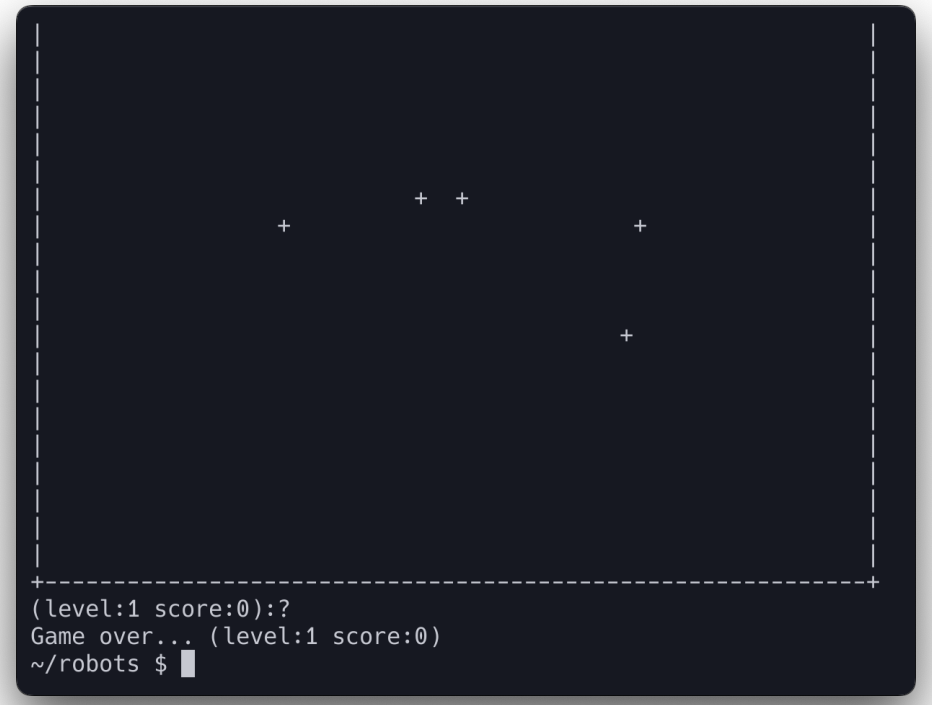

図 2.6　ゲームオーバー後

2.3　入力の制限

以下のように入力できるものを制限することによって想定外の入力に備えている。

Listing 2.1　`action()` の一部

```
1 while(1) {
2     input = getChar();
3     if((input - '0') >= 0 && (input - '0') <= 9) break;
4     }
```

第 3 章

各処理の計算量

3.1 プレイヤー移動

プレイヤーの移動を行う際、スクラップに衝突する方向には移動できない。この判定を行うために移動先の座標にスクラップがあるかどうかを確認する必要がある。全てのロボット分確認するため、計算量は $O(n)$ となる。

3.2 ロボット移動

`move_robots()` にて、処理を行っている。まず、ロボットが死んでいないか確認後、プレイヤーに最も接近する方向に移動させる。この処理を各ロボットに対して行うため、計算量は $O(n)$ となる。

3.3 衝突

`check_robots_collision()` にて、処理を行っている。全てのロボットに対して衝突しているかどうかを確認する必要があるが、二重ループを用いた総当たりでは計算量が $O(n^2)$ と、非効率である。そこで、一度座標順（昇順）にソートしてからループを回すことによって計算量の削減を図った。ソートはクイックソートを用いた。クイックソートには $O(n \log n)$、その後の比較に $O(n)$ かかるため、全体の計算量は $O(n \log n)$ である。

3.4 プログラムが用いるデータ領域の大きさ

プレイヤーの座標で $O(1)$、ロボットの座標で $O(n)$、盤面表示用の配列で $O(m)$、座標決定用の配列で $O(m)$ のデータ領域を使用する。したがって

$$O(1) + O(n) + O(m) + O(m) = O(m + n)$$

となる。

付録 A

プログラムリスト

Listing A.1　j21403.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 #define ROW 60
6 #define COL 20
7 #define MAX_ROBOTS 40
8
9 typedef struct {
10     int x;
11     int y;
12     int died;
13 } position;
14
15 extern char getChar(void);
16 int min(int a, int b);
17 void shuffle(int *array, int size);
18 void initialize(position *player, position robots[], int robots_num, int *array);
19 void print_board(position player, position robots[], int robots_num, int level, int score);
20 int action(position *player, position robots[], int robots_num);
21 void move_robots(position player, position robots[], int robots_num, int *score);
22 int check_collision(position player, position robots[], int robots_num);
23 void check_robots_collision(position robots[], int robots_num, int *score);
24 int check_level_clear(position robots[], int robots_num);
25
26
27 int main(void) {
28     int array[ROW*COL];
29     position player;
30     position robots[MAX_ROBOTS];
31     int level = 1;
32     int score = 0;
33     int robots_num;
34
35     for(int i = 0; i < ROW*COL; i++) {
36         array[i] = i;
37     }
38
39     int flag = 1;
40     while(flag) {
```

```
41          shuffle(array, ROW*COL);
42          robots_num = min(level * 5, MAX_ROBOTS);
43          initialize(&player, robots, robots_num, array);
44          print_board(player, robots, robots_num, level, score);
45
46          while(1) {
47              if(!action(&player, robots, robots_num)) {
48                  move_robots(player, robots, robots_num, &score);
49              }
50              print_board(player, robots, robots_num, level, score);
51              if(check_collision(player, robots, robots_num) == 1) {
52                  printf("Game over... (level:%d score:%d)\n", level, score);
53                  flag = 0;
54                  break;
55              }
56              if(check_level_clear(robots, robots_num) == 0) {
57                  score += level * 10;
58                  level++;
59                  robots_num = min(level * 5, MAX_ROBOTS);
60                  break;
61              }
62          }
63      }
64
65      return 0;
66  }
67
68
69  int min(int a, int b) {
70      if(a < b) return a;
71      return b;
72  }
73
74
75  void shuffle(int *array, int size) {
76      srand((unsigned)time(NULL));
77      int j, temp;
78      for(int i = size - 1; i > 0; i--) {
79          j = rand() % (i + 1);
80          temp = array[i];
81          array[i] = array[j];
82          array[j] = temp;
83      }
84  }
85
86
87  void initialize(position *player, position robots[], int robots_num, int *array) {
88      for(int i = 0; i < MAX_ROBOTS; i++) {
89          robots[i].x = 0;
90          robots[i].y = 0;
91          robots[i].died = 0;
92      }
93
94      player -> x = array[0] / COL;
95      player -> y = array[0] % COL;
96
97      for(int i = 0; i < robots_num; i++) {
```

```
98          robots[i].x = array[i + 1] / COL;
99          robots[i].y = array[i + 1] % COL;
100         robots[i].died = 0;
101     }
102 }
103
104
105 void print_board(position player, position robots[], int robots_num, int level, int score) {
106     char board[COL][ROW];
107
108     for(int i = 0; i < COL; i++) {
109         for(int j = 0; j < ROW; j++) {
110             board[i][j] = ' ';
111         }
112     }
113     //配置
114     board[player.y][player.x] = '@';
115     for(int i = 0; i < robots_num; i++) {
116         if(robots[i].died) {
117             board[robots[i].y][robots[i].x] = '*';
118         } else {
119             board[robots[i].y][robots[i].x] = '+';
120         }
121     }
122     //表示
123     printf("\033[2J\033[1;1H");
124     printf("+");
125     for(int i = 0; i < ROW; i++) printf("-");
126     printf("+\n");
127
128     for(int i = 0; i < COL; i++) {
129         printf("|");
130         for(int j = 0; j < ROW; j++) {
131             printf("%c", board[i][j]);
132         }
133         printf("|\n");
134     }
135
136     printf("+");
137     for(int i = 0; i < ROW; i++) printf("-");
138     printf("+\n");
139     printf("(level:%d score:%d):?\n", level, score);
140 }
141
142
143 int action(position *player, position robots[], int robots_num) {
144     char input;
145     int x = player -> x;
146     int y = player -> y;
147
148     while(1) {
149         input = getChar();
150         if((input - '0') >= 0 && (input - '0') <= 9) break;
151     }
152
153     switch(input) {
154         case '0':
```

```
155             srand((unsigned)time(NULL));
156             x = rand() % ROW;
157             y = rand() % COL;
158             break;
159         case '1':
160             x--;
161             y++;
162             break;
163         case '2':
164             y++;
165             break;
166         case '3':
167             x++;
168             y++;
169             break;
170         case '4':
171             x--;
172             break;
173         case '5':
174             break;
175         case '6':
176             x++;
177             break;
178         case '7':
179             x--;
180             y--;
181             break;
182         case '8':
183             y--;
184             break;
185         case '9':
186             x++;
187             y--;
188             break;
189     }
190
191     //移動先がスクラップかどうか確認
192     for(int i = 0; i < robots_num; i++) {
193         if(robots[i].died && x == robots[i].x && y == robots[i].y) {
194             return 1;
195         }
196     }
197     //移動先が場外でなければ移動を実行
198     if(x >= 0 && x < ROW && y >= 0 && y < COL) {
199         player -> x = x;
200         player -> y = y;
201         return 0;
202     }
203     return 1;
204 }
205
206
207 void move_robots(position player, position robots[], int robots_num, int *score) {
208     for(int i = 0; i < robots_num; i++) {
209         if(robots[i].died) {
210             continue;
211         }
```

```
212
213         if(robots[i].x < player.x) robots[i].x++;
214         if(robots[i].x > player.x) robots[i].x--;
215         if(robots[i].y < player.y) robots[i].y++;
216         if(robots[i].y > player.y) robots[i].y--;
217     }
218
219     check_robots_collision(robots, robots_num, score);
220 }
221
222
223 int check_collision(position player, position robots[], int robots_num) {
224     for(int i = 0; i < robots_num; i++) {
225         if(robots[i].x == player.x && robots[i].y == player.y) {
226             return 1;
227         }
228     }
229     return 0;
230 }
231
232
233 int compare_positions(const void *a, const void *b) {
234     position *posA = (position *)a;
235     position *posB = (position *)b;
236     if(posA->x != posB->x) {
237         return posA->x - posB->x;
238     } else {
239         return posA->y - posB->y;
240     }
241 }
242
243 void check_robots_collision(position robots[], int robots_num, int *score) {
244     qsort(robots, robots_num, sizeof(position), compare_positions);
245     for(int i = 0; i < robots_num - 1; i++) {
246         if(robots[i].x == robots[i + 1].x && robots[i].y == robots[i + 1].y) {
247             if(!robots[i].died) *score += 1;
248             if(!robots[i + 1].died) *score += 1;
249             robots[i].died = 1;
250             robots[i + 1].died = 1;
251         }
252     }
253 }
254
255
256 int check_level_clear(position robots[], int robots_num) {
257     int died_robot = 0;
258     for(int i = 0; i < robots_num; i++) {
259         if(robots[i].died) died_robot++;
260     }
261     if(died_robot == robots_num) return 0;
262     return 1;
263 }
```

Listing A.2　getchar.c

```
#include <termio.h>

char getChar(void)
{
    struct termio old_term, new_term;

    char c;

    /* 現在の設定を得る */
    ioctl(0, TCGETA, &old_term);

    /* 設定のコピーをつくる */
    new_term = old_term;

    /* 入力文字のエコーを抑止する場合 */
    new_term.c_lflag &= ~(ICANON | ECHO);

    /* エコーは止めない場合 */
    //new_term.c_lflag &= ~(ICANON);

    /* 新しい設定を反映する */
    ioctl(0, TCSETAW, &new_term);

    /* 1 文字入力 */
    c = getchar();

    /* 古い設定に戻す */
    ioctl(0, TCSETAW, &old_term);

    return (c);
}
```

Listing A.3　Makefile

```
robots: j21403.o getchar.o
        gcc -o $@ $^

clean:
        rm -f *.o robots
```