提出日：2017 年 06 月 20 日

M スイーパの計画書

|  |
| --- |
| 木 更 津 工 業 高 等 専 門 学 校 |
| 情 報 工 学科 |
| 学生番号： 1 3 － 444 |
| 氏名： L E　 Q U A N G　 H U N G |

1. **ゲームの概要**

　M スイーパは、M ボード上に隠れている M を捜し出すゲームである。

M ボードは、下のように 8×8 のます目になっている。それぞれの点には、M があるかま たは M がない。ゲーム開始時には、この情報は隠されている。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | . | . | . | . | . | . |
| 1 | . | . | . | . | . | . |
| 2 | . | . | . | . | . | . |
| 3 | . | . | . | . | . | . |
| 4 | . | . | . | . | . | . |
| 5 | . | . | . | . | . | . |

　プレイヤーは、ボード上の任意の点を“開く”ことができる。ある点 P を“開く”と次のい ずれかが起こる。

* P に M が隠されていたとき 🡪 ゲームオーバ
* P に M がないとき 🡪 P の周囲の点に隠されている M の数が 公開される

ただし、ここで P の周囲の点とは、下の図の Q0～Q7 の位置関係にある 8 点である。P が辺（たとえば (0、4）または隅（たとえば (0、0）にあって、Q がボードの外にはみだす ときには、その Q は周囲の点に含まれない．。

Q0 Q1 Q2

Q3 P Q4

Q5 Q6 Q7

　 例として、(1、2)、(3、3)、(5、4) に M があるときに、(4、4) を開いたとする。(4、4) には M がないので、(4、4) の周囲の M の数 = 2 が公開される。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | . | . | . | . | . | . |
| 1 | . | . | . | . | . | . |
| 2 | . | . | . | . | . | . |
| 3 | . | . | . | . | . | . |
| 4 | . | . | . | . | 2 | . |
| 5 | . | . | . | . | . | . |

　M のある点を開かずに、すべての安全な点を開くことができたら、ゲームクリアである。 M のある点を開いてしまったら、ゲームオーバとなる。

1. **実行方法と制約** 
   1. **実行方法**

　ソースコードの中に使われるcurses.hライブラリが実行環境にインス トール済みを前提としてプログラムを実行する。

コンパイルと実行コマンド： gcc msweeper.c -lcurses (コンパイルコマンド)

　　　./a.out （実行コマンド）

* 1. **制約**
* 原則として、グローバル変数を用いてはならない
* 少なくとも 10 個以上の関数を作らなくてはならない。
* 原則として 40 行以上の関数を作ってはならない。

1. **プログラム構造とデータ構造**
   1. **プログラム外部仕様**

　プログラムを実行するときに、初期値としてゲームボードのサイズ（縦、行）と爆弾の数 を入力してゲーム開始する。入力データに応じて画面上に、ゲームボードが表示される。プ レイヤーは矢印キーや１～９キーを使って位置を移動しながら‘o’ボタンでブロックを開 く（既に‘？’が付けられるところは開かないこと）。その時、以下の 3 パタンのいずれかが起こる。

* 開く所が爆弾 🡪 ゲームオーバになる
* 開く所に爆弾がない、周りに爆弾がある 🡪 周りの爆弾の数を表示する
* 開く所にも周りにも爆弾がない 🡪 自動的に近くの安全ブロックも開く

　又は、爆弾があるかどうか分からない時、‘m’ボタンを使って‘？’記号を付けられる。 ‘？’の数が爆弾数に一致になる時、自動的にチェックする。もし、爆弾の位置が間違っ たらゲームオーバになる。ゲームクリア条件は安全なブロックを全部開くか‘？’を付けら れる所が全部爆弾である。ゲームをクリアするときハイスコアを記録する機能も設定され ている。

* 1. **プログラム内部仕様** 
     1. **データ構造**

　プログランムは以下の 2 つのデータ構造が使われている。

1. 一つブロックのデータを格納するデータ構造

struct {

int data;

int flag;

int open;

} BLOCK;

変数の役割：

* bomb　:　 そのブロックの周囲の爆弾の数　（そのブロックが爆弾ならば－1 が入る）
* flag　:　‘？’が付けられるかどうかの情報　（０：付けられない、１：付けられる）
* open　：　ブロックが開くかどうかの情報　（０：開かない、１：開く）

1. プレイヤーの情報を格納するデータ構造

struct {

char name[20];

int data[3];

double time;

　　} PLAYER;

変数の役割：

* name : プレイヤーの名前を格納する
* map： 順番にゲームボードのサイズ（縦、行）と爆弾の数の情報
* time：ゲームをクリアする時間
  + 1. **各関数**

1. void init\_data(int \*col,int \*row, int \*M)；

* 引数：マップサイズ（col、row）と爆弾の数のポインター。
* 戻り値：なし
* 処理内容：3つのポインターを受け取ってキーボードー入力による各データの初期値を格納する関数。まず、端末が表示できる範囲を調べ、入力データは範囲外にならないように各変数に条件付ける。具体的に以下のように決められる：

1　≤　マップの列数　 ≤　端末の最大列数

1　≤　マップの行数　 ≤　端末の最大行数

1　≤　マップの爆弾数 ≤ マップの列数ｘマップの行数

1. void init\_screen()；

* 引数：なし。
* 戻り値：なし
* 処理内容：使用する前に画面を初期化する。

1. void init\_map(int col,int row,int M, BLOCK map[col][row]);

* 引数：マップの初期化データとマップ。
* 戻り値：なし。
* 処理内容：マップ上の爆弾位置をランダム関数で配置する。又は周りの爆弾の数を各ブロックに初期化する。

1. void game\_play(int col,int row,int M, BLOCK map[col][row]);

* 引数：マップの初期化データとマップ。
* 戻り値：なし。
* 処理内容：　ゲームが終了するまでにループしながら入力待ち。入力によって次の処理に分岐する。

1. void display\_map(int col,int row,int M,BLOCK map[col][row]);

* 引数：マップの初期化データとマップ。
* 戻り値：なし。
* 処理内容：各ブロックのデータ構造の中身によって画面に表示する。

1. void display\_help(int row,int mark\_cnt,int M);

* 引数：マップの初期化データ。
* 戻り値：なし。
* 処理内容：画面に表示されるマップの下にゲーム操作ガイドを表示する。

1. int auto\_open(int x,int y,int col,int row,BLOCK map[col][row]);

* 引数：マップの初期化データとマップ。
* 戻り値：ゲーム終了を判断するgameover変数。
* 処理内容：ブロックの爆弾数を見て 0 ならば再帰でそのブロックの周囲の８ブロックも同じ 操作を行う。そうではなければ、BLOCK データ構造の open 変数に 1 をセットする。

1. int mark(int x,int y,int col,int row,BLOCK map[col][row]);

* 引数：マップの初期化データとマップ。
* 戻り値：半自動チェックする関数の引数mark\_cnt
* 処理内容：そのブロックがオ ーペンされるかどうか見に行き、オーペンされたら 0 を戻し、関数をぬけ出す。BLOCK デー タ構造の flag 変数の値をかえる。最後に、flag 変数の値をみて、1 ならば１を戻すし（？ がセットされ）0 ならば-1 を戻す（？を取り消す）。

1. int check\_mark(int col,int row,BLOCK map[col][row]);

* 引数：マップの初期化データとマップ。
* 戻り値：。ゲーム終了を判断するgameover変数。
* 処理内容：：？を付けるところを全部みて、どこかで爆弾でなければ 1 を戻す、全部爆弾ならば -1 を戻す。

1. void game\_over(time\_t start\_t,int gameover,int col,int row,int M,BLOCK map[col][row]);

* 引数：開始時刻、マップの初期化データとマップ
* 戻り値：なし。
* 処理内容：プレヤーが勝ったか負けたかをみて対応する表示を出す。また、新記録になるかどうかもチェックする。

1. int check\_gm(int col,int row,BLOCK map[col][row]);

* 引数：マップの初期化データとマップ。
* 戻り値：ゲーム終了を判断するgameover変数。
* 処理内容：auto\_open関数の中で使われる。ゲームが続くかどうかをチェックする。

1. void show\_mines(int col,int row,BLOCK map[col][row]);

* 引数：マップの初期化データとマップ。
* 戻り値：なし。
* 処理内容：　マップ上の爆弾の位置を全部開ける。

1. void high\_score(double time,int col,int row,int M);

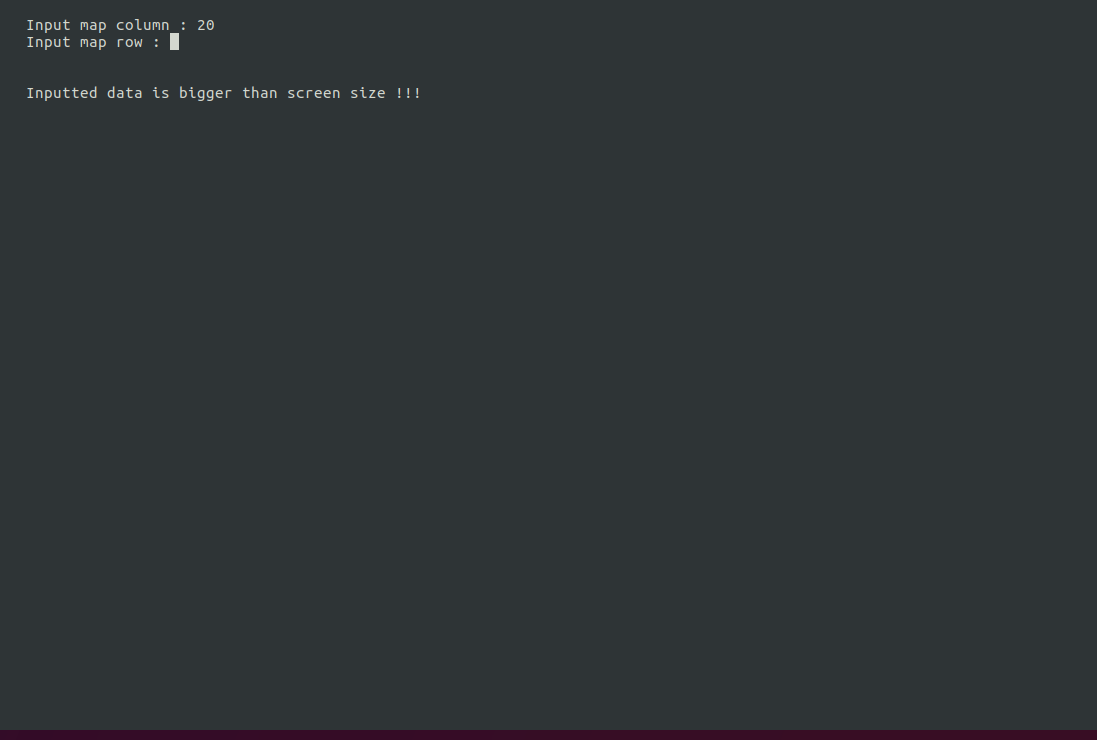
* 引数：プレイする時間、マップの初期化データとマップ。。
* 戻り値：なし。
* 処理内容：　新記録が出たときファイルに書き込む。

1. void view\_score();

* 引数：なし。
* 戻り値：なし。
* 処理内容：　ハイスコアを保存するファイルからデータを読み取る、画面に表示する。

4.3 プログラムの正しさの検証

* 入力データは範囲外にする場合　（row = 200）：



* すべて爆弾位置を‘？’を付ける場合：

|  |
| --- |
|  |
| ‘ｍ’を押すと |
|  |

* すべて安全な位置を開ける場合：

|  |
| --- |
|  |
| ‘o’を押すと |
|  |

* ハイスコアが出た場合：

|  |
| --- |
|  |
| “test”を入力すると |
|  |