ゲームプログラミング

山本昌志*

2007 年 12 月 5 日

概 要

簡単なインベーダーゲームの作成を通して，マウスとキーボード イベントの基礎的な内容を学習する．

1 本日の学習内容

マウスおよびキーボード イベント処理，および乱数の発生方法を学ぶ．本日のゴールは，次の通りである．

- マウスイベントの処理方法が分かる．
- キーボード イベントの処理方法が分かる．
- 乱数の発生方法が分かる．

2 インベーダーゲーム

2.1 プログラムリスト

昔，流行ったインベーダーゲーム (もどき) のプログラムをリスト 1 に示す．遊び方は，言うまでもないであろう．このプログラムでは，マウスとキーボード イベントを使っている．

このプログラムは最適化を全く行っていないので，動作は遅い．もし，諸君が中間試験以降，作成するプログラムの参考にするならば，最適化を図った方が良いだろう．

リスト 1: マウスとキーボード イベントを使ったプログラム例．インベーダーゲームができる．

```
1  #include <stdio.h>
2  #include <GL/glut.h>
3  #include <math.h>
4  #include <stdlib.h>
5  #include <time.h>
6
7  #define H_WIN 400                    // ウィンドウの幅
8  #define W_WIN 300                    // ウィンドウの高さ
9
10 #define W2_HODAI 10                  // 砲台の横幅の半分
11 #define H_HODAI 15                   // 砲台の上面の y座標
12 #define L_HODAI 5                    // 砲台の下面の y座標
```

*独立行政法人　秋田工業高等専門学校　電気情報工学科

```
13 #define L_E_BEAM 20                    // 防衛軍のビームの長さ
14 #define V_E_BEAM 1.5                   // 防衛軍のビームの速度
15 #define N_E_BEAM 1                     // 防衛軍のビームの画面上の最大数
16
17 #define L_I_BEAM 10                    // インベーダー軍のビームの長さ
18 #define V_I_BEAM 0.8                   // インベーダー軍のビームの速度
19 #define P_I_BEAM 500                   // インベーダー軍のビームの初期発射確率
20
21 #define N_I_BEAM 20                    // インベーダー軍のビームの画面上の最大数
22 #define NXIV 9                         // インベーダー軍の列の数
23 #define NYIV 4                         // インベーダー軍の行の数
24 #define V_INVADER 0.1                  // インベーダー軍の速度
25
26 #define NOT_DECIDE 0
27 #define INVADER 1
28 #define HUMAN 2
29
30 //---- プロトタイプ宣言 ----
31 void initialize(void);                 // 初期化
32
33 void draw(void);                       // 図を描く
34 void draw_result(void);                // 結果表示
35 void draw_hodai(void);                 // 防衛軍の砲台の描画
36 void draw_e_beam(void);                // 防衛軍のビームの描画
37 void draw_i_beam(void);                // インベーダー軍のビームの描画
38 void draw_invader(void);               // インベーダー軍の描画
39
40 void change_state(void);               // 状態変化に関する処理
41 void state_e_beam(void);               // 防衛軍のビームの状態変化
42 void state_invader(void);              // インベーダー軍の状態変化
43 void state_i_beam(void);               // インベーダー軍のビーム状態変化
44
45 void mouse_xy(int x, int y);
46 void shoot(unsigned char key, int x, int y); // 防衛軍ビーム発射
47
48 void resize(int w, int h);             // サイズの調整
49 void set_color(void);                  // 塗りつぶし色の設定
50
51 //---- グローバル変数 ----
52 double xc = 100.0;                     // マウスのx座標
53
54 typedef struct{
55   unsigned char status;                // 0:dead 1:alive
56   double x, y;                         // 中心座標
57 }invader;
58
59 invader invd[NXIV][NYIV];              // インベーダー
60 int alive_inv=NXIV*NYIV;               // 生きているインベーダーの数
61 double inv_vx=V_INVADER;               // インベーダーの横方向の速度
62
63
64 typedef struct{
65   char status;                         // 0:格納庫 1:砲台の上 2:移動中
66   double x;                            // ビームのx座標
67   double y0, y1;                       // ビームのy座標 y0:先頭 y1:最後尾
68   double vy;                           // ビームの速度
69 }beam;
70
71 beam e_beam[N_E_BEAM];                 // 地球防衛軍のビーム
72 beam *p_e_beam1;                       // 地球防衛軍の次に発射可能なビーム
73 beam i_beam[N_I_BEAM];                 // インベーダー軍のビーム
74
```

```
75  int winner = NOT_DECIDE;
76  char *win="You won a game.";
77  char *lost="You lost a game.";
78  //=========================================================================
79  // main関数
80  //=========================================================================
81  int main(int argc, char *argv[])
82  {
83
84    initialize();
85    glutInitWindowPosition(100,200);           // 初期位置(x,y)指定
86    glutInitWindowSize(W_WIN,H_WIN);           // 初期サイズ(幅, 高さ)指定
87    glutInit(&argc, argv);                     // GLUT 初期化
88    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);    // 表示モードの指定
89    glutCreateWindow("space invader modoki");        // windowをタイトルを付けてを開く
90    glutDisplayFunc(draw);                     // イベントにより呼び出し
91    glutReshapeFunc(resize);                   // サイズ変更のときに呼び出す関数指定
92    glutIdleFunc(change_state);                // 暇なときに実行(状態の変化)
93    glutPassiveMotionFunc(mouse_xy);           // マウスイベント(砲台の移動)
94    glutKeyboardFunc(shoot);                   // キーボードイベント(ビームを発射)
95    set_color();                               // 塗りつぶす色指定
96    glutMainLoop();                            // GLUTの無限ループ
97
98    return 0;
99  }
100
101 //=========================================================================
102 // 初期化
103 //=========================================================================
104 void initialize(void)
105 {
106   int i, j;
107
108   srand((unsigned int)time(NULL));          // 乱数を発生させるため
109
110   for(i=0; i<N_E_BEAM; i++){
111     e_beam[i].status=0;
112     e_beam[i].y0=H_HODAI+L_E_BEAM;
113     e_beam[i].y1=H_HODAI;
114     e_beam[i].vy=0.0;
115   }
116
117   e_beam[0].status=1;                       // 砲台にのせる
118   p_e_beam1=&e_beam[0];
119
120   for(i=0; i<N_I_BEAM; i++){
121     i_beam[i].status = 0;
122     i_beam[i].y0 = 0;
123     i_beam[i].y1 = 0;
124     i_beam[i].vy = V_I_BEAM;
125   }
126
127   for(i=0; i<NXIV; i++){
128     for(j=0; j<NYIV; j++){
129       invd[i][j].status=1;
130       invd[i][j].x = 20*(i+1);              // x,yとも20ピクセル間隔
131       invd[i][j].y = H_WIN - NYIV*20+10+20*j;
132     }
133   }
134 }
135
136
```

```
137 //=============================================================
138 // 図を描く
139 //=============================================================
140 void draw(void)
141 {
142   glClear(GL_COLOR_BUFFER_BIT);
143
144   if(winner != NOT_DECIDE) draw_result();
145
146   draw_hodai();          // 砲台を描く関数呼び出し
147   draw_e_beam();         // 地球防衛軍のビームを描く関数の呼び出し
148   draw_i_beam();         // インベーダー軍のビームを描く関数の呼び出し
149   draw_invader();        // インベーダーを描く関数の呼び出し
150
151   glutSwapBuffers();     // 描画
152 }
153
154
155 //=============================================================
156 // 勝者の表示
157 //=============================================================
158 void draw_result(void)
159 {
160     int i=0;
161
162     glColor3d(0.0, 1.0, 0.0);
163
164     if(winner==HUMAN){
165       while(win[i]!='\0'){
166         glRasterPos2i(50+15*i,100);
167         glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,win[i]);
168         i++;
169       }
170     }else if(winner==INVADER){
171       while(lost[i]!='\0'){
172         glRasterPos2i(50+15*i,100);
173         glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,lost[i]);
174         i++;
175       }
176     }
177 }
178
179
180 //=============================================================
181 // 地球防衛軍の砲台の描画
182 //=============================================================
183 void draw_hodai(void)
184 {
185   glColor3d(0.5, 1.0, 0.5);              // 線の色指定(RGB)
186   glBegin(GL_POLYGON);
187   glVertex2d(xc-W2_HODAI, L_HODAI);
188   glVertex2d(xc+W2_HODAI, L_HODAI);
189   glVertex2d(xc+W2_HODAI, H_HODAI);
190   glVertex2d(xc-W2_HODAI, H_HODAI);
191   glEnd();
192 }
193
194
195 //=============================================================
196 // 地球防衛軍のビーム砲の描画
197 //=============================================================
198 void draw_e_beam(void)
```

```
199 {
200   int i;
201
202   for(i=0;i<N_E_BEAM;i++){
203     if(e_beam[i].status != 0){
204       glColor3d(1.0, 0.0, 0.0);              // 線の色指定(RGB)
205       glBegin(GL_LINES);
206       glVertex2d(e_beam[i].x, e_beam[i].y0);
207       glVertex2d(e_beam[i].x, e_beam[i].y1);
208       glEnd();
209     }
210   }
211 }
212
213
214 //=============================================================
215 // インベーダー軍のビームの描画
216 //=============================================================
217 void draw_i_beam(void)
218 {
219   int i;
220
221   for(i=0; i<N_I_BEAM; i++){
222     if(i_beam[i].status == 2){
223       glColor3d(0.0, 0.0, 1.0);              // 線の色指定(RGB)
224       glBegin(GL_LINES);
225       glVertex2d(i_beam[i].x, i_beam[i].y0);
226       glVertex2d(i_beam[i].x, i_beam[i].y1);
227       glEnd();
228     }
229   }
230 }
231
232
233 //=============================================================
234 // インベーダー軍の描画
235 //=============================================================
236 void draw_invader(void)
237 {
238   int i, j;      // インベーダーの i列j行
239
240   for(i=0; i<NXIV; i++){
241     for(j=0; j<NYIV; j++){
242       if(invd[i][j].status==1){     // 生きているインベーダーのみを描く
243
244         //------ 胴体 ----------------------
245         glColor3d(1.0, 1.0, 1.0);
246         glBegin(GL_POLYGON);
247         glVertex2d(invd[i][j].x-8, invd[i][j].y);
248         glVertex2d(invd[i][j].x-3, invd[i][j].y-4);
249         glVertex2d(invd[i][j].x+3, invd[i][j].y-4);
250         glVertex2d(invd[i][j].x+8, invd[i][j].y);
251         glVertex2d(invd[i][j].x+3, invd[i][j].y+4);
252         glVertex2d(invd[i][j].x-3, invd[i][j].y+4);
253         glEnd();
254
255         //------- 手 足 触覚 ----------------------
256         glBegin(GL_LINES);
257         glVertex2d(invd[i][j].x-7, invd[i][j].y);     // 左手
258         glVertex2d(invd[i][j].x-7, invd[i][j].y+6);
259         glVertex2d(invd[i][j].x+7, invd[i][j].y);     // 右手
260         glVertex2d(invd[i][j].x+7, invd[i][j].y+6);
```

```
261           glVertex2d(invd[i][j].x-4, invd[i][j].y-4);  // 左足
262           glVertex2d(invd[i][j].x-6, invd[i][j].y-8);
263           glVertex2d(invd[i][j].x+4, invd[i][j].y-4);  // 右足
264           glVertex2d(invd[i][j].x+6, invd[i][j].y-8);
265           glVertex2d(invd[i][j].x-2, invd[i][j].y+4);  // 左触覚
266           glVertex2d(invd[i][j].x-5, invd[i][j].y+6);
267           glVertex2d(invd[i][j].x+2, invd[i][j].y+4);  // 右触覚
268           glVertex2d(invd[i][j].x+5, invd[i][j].y+6);
269           glEnd();
270
271           //------- 目玉 -----------------
272           glColor3d(0.0, 0.0, 0.0);
273           glBegin(GL_POLYGON);                  // 左目
274           glVertex2d(invd[i][j].x-3, invd[i][j].y);
275           glVertex2d(invd[i][j].x-1, invd[i][j].y);
276           glVertex2d(invd[i][j].x-1, invd[i][j].y+2);
277           glVertex2d(invd[i][j].x-3, invd[i][j].y+2);
278           glEnd();
279           glBegin(GL_POLYGON);                  // 右目
280           glVertex2d(invd[i][j].x+3, invd[i][j].y);
281           glVertex2d(invd[i][j].x+1, invd[i][j].y);
282           glVertex2d(invd[i][j].x+1, invd[i][j].y+2);
283           glVertex2d(invd[i][j].x+3, invd[i][j].y+2);
284           glEnd();
285         }
286       }
287     }
288   }
289
290
291   //=============================================================
292   // リサイズ
293   //      この関数は window のサイズが変化したら呼び出される
294   //      引数
295   //              w:ウィンドウの幅
296   //              h:ウィンドウの高さ
297   //=============================================================
298   void resize(int w, int h)
299   {
300     glLoadIdentity();                     // 変換行列を単位行列に
301     gluOrtho2D(0, W_WIN, 0, H_WIN);       // world座標系の範囲
302     glViewport(0, 0, w, h);               // ウィンドウ座標系を指定
303   }
304
305
306   //=============================================================
307   // PCが暇なときに実行する. これが実行されると状態が変化する
308   //=============================================================
309   void change_state(void)
310   {
311
312     if(winner == NOT_DECIDE){
313       state_e_beam();       // 地球防衛軍のビームの処理
314       state_invader();      // インベーダー軍の処理
315       state_i_beam();       // インベーダー軍のビームの処理
316     }
317
318     glutPostRedisplay();
319   }
320
321
322   //=============================================================
```

```
323 // 地球防衛軍のビームの状態の処理
324 //=============================================================
325 void state_e_beam(void)
326 {
327   int i,l,m;
328   int st0=0;
329   int rdy=0;
330   int nshoot=0;                    // 発射済みの地球防衛軍の玉の数
331   double min_y=H_WIN+L_E_BEAM;     // 最もしたのミサイルの先のy座標
332   double ydis;                     // 最も下のミサイルと発射台の距離
333
334   for(i=0; i<N_E_BEAM; i++){
335     switch(e_beam[i].status){
336
337     //--------  格納庫にあるビームの処理 ----------------------------
338     case 0:
339       st0=i;                       // 次に発射可能なビームを設定
340       break;
341
342     //--------  砲台にあるビームの処理 ------------------------------
343     case 1:
344       e_beam[i].x = xc;            // x方向に移動
345       rdy=1;                       // 砲台にビームがあることを示すフラグをON
346       break;
347
348     //--------  すでに発射されたビームの処理 ------------------------
349     case 2:
350       nshoot++;                              // 発射されているビームをカウント
351       e_beam[i].y0 += e_beam[i].vy;          // ビームの移動
352       e_beam[i].y1 += e_beam[i].vy;
353
354       // ------ インベーダーにビームが衝突したことを確認して処理 ------
355       for(l=0; l<NXIV; l++){
356         for(m=0; m<NYIV; m++){
357           if(invd[l][m].status==1){
358             if((invd[l][m].x-8 < e_beam[i].x) &&
359                (e_beam[i].x < invd[l][m].x+8) &&
360                (invd[l][m].y-4 < e_beam[i].y0) &&
361                (e_beam[i].y1 < invd[l][m].y+4)){
362               invd[l][m].status=0;             // インベーダーの死亡
363               alive_inv--;                     // 生きているインベーダーの数を-1
364               if(alive_inv==0)winner=HUMAN;
365               e_beam[i].status=0;              // ビームは格納庫へ
366               e_beam[i].y0=H_HODAI+L_E_BEAM;   // ビームの初期化
367               e_beam[i].y1=H_HODAI;
368             }
369           }
370         }
371       }
372
373
374       // ---- 画面から地球防衛軍のビームがはみ出た場合の処理 --------
375       if(H_WIN+L_E_BEAM < e_beam[i].y0){
376         e_beam[i].status = 0;
377         e_beam[i].y0 = H_HODAI+L_E_BEAM;
378         e_beam[i].y1 = H_HODAI;
379         e_beam[i].vy = 0.0;
380       }
381       if(e_beam[i].y0 < min_y) min_y=e_beam[i].y0;
382       break;
383     default:
384       printf("e_beam status error!!\n");
```

```
385         exit(1);
386       }
387     }
388
389
390     // ---- 地球防衛軍の新たな発射可能なビームの処理 ------
391     ydis = min_y-H_HODAI;
392     if( (2.5*L_E_BEAM < ydis) && (rdy==0) && (nshoot<N_E_BEAM) ){
393       e_beam[st0].status=1;
394       p_e_beam1=(beam *)&e_beam[st0];       // 発射可能なビームをポインターで表現
395     }
396   }
397
398
399   //=============================================================
400   // インベーダー軍の状態の処理
401   //=============================================================
402   void state_invader(void)
403   {
404     int i, j, k;
405     double ivmin_x=W_WIN, ivmax_x=0;
406     double ivmin_y=H_WIN, ivmax_y=0;
407     int can_attack;
408
409     for(i=0; i<NXIV; i++){
410       can_attack=1;
411       for(j=0; j<NYIV; j++){
412         if(invd[i][j].status==1){    // インベーダーの生死のチェック
413           invd[i][j].x += inv_vx;    // インベーダーの横方向移動
414           // ---- インベーダー軍のビーム発射の処理 -------
415           if(can_attack == 1 && rand()%P_I_BEAM == 0){  // 発射条件
416             for(k=0; k<N_I_BEAM; k++){
417               if(i_beam[k].status !=2){       // 発射可能なビームを探す
418                 i_beam[k].status =2;          // ビームの発射
419                 i_beam[k].x = invd[i][j].x;
420                 i_beam[k].y0 = invd[i][j].y;
421                 i_beam[k].y1 = invd[i][j].y-L_I_BEAM;
422                 break;
423               }
424             }
425           }
426           // --- インベーダー軍の左右上下の端の座標 -------
427           if(invd[i][j].x < ivmin_x) ivmin_x=invd[i][j].x;    // 左端
428           if(invd[i][j].x > ivmax_x) ivmax_x=invd[i][j].x;    // 右端
429           if(invd[i][j].y < ivmin_y) ivmin_y=invd[i][j].y;    // 下の端
430           if(invd[i][j].y > ivmax_y) ivmax_y=invd[i][j].y;    // 上の端
431           can_attack=0;
432         }
433       }
434     }
435
436
437     if(ivmin_x < 10) inv_vx = V_INVADER;              // 左端に達したとき
438     if(ivmax_x > W_WIN-10) inv_vx = -V_INVADER;       // 右端に達したとき
439
440     if((ivmin_x < 10) || (ivmax_x > W_WIN-10)){       // 左右の端に達しとき
441       for(i=0; i<NXIV; i++){
442         for(j=0; j<NYIV; j++){
443           invd[i][j].y -= 10;                          // 下に降りる
444         }
445       }
446     }
```

```
447 }
448
449
450 //=============================================================
451 // インベーダー軍のビームの状態の処理
452 //=============================================================
453 void state_i_beam(void)
454 {
455   int i;
456
457   for(i=0; i<N_I_BEAM; i++){
458     if(i_beam[i].status ==2){
459       i_beam[i].y0 -= i_beam[i].vy;
460       i_beam[i].y1 -= i_beam[i].vy;
461
462       if(i_beam[i].y1 < 0) i_beam[i].status=0;
463
464       if((xc-W2_HODAI < i_beam[i].x) &&
465          (i_beam[i].x < xc+W2_HODAI) &&
466          (L_HODAI < i_beam[i].y0) &&
467          (i_beam[i].y1 < H_HODAI)){
468         winner=INVADER;              // 地球防衛軍の負け
469       }
470     }
471   }
472 }
473
474
475 //=============================================================
476 // マウスイベントの処理
477 //=============================================================
478 void mouse_xy(int x, int y)
479 {
480   xc=x;                  // マウスの x座標をグローバル変数の  xc へ代入
481 }
482
483
484 //=============================================================
485 // キーボードイベントの処理
486 // スペースキーが押されたら地球防衛軍のビームを発射
487 //=============================================================
488 void shoot(unsigned char key, int x, int y)
489 {
490   //--- スペースキーが押されて, 発射可能なビームがあるとき ----
491   if(key==' ' && p_e_beam1 != NULL){
492     p_e_beam1->status = 2;             // ビームを発射の状態にする
493     p_e_beam1->vy = V_E_BEAM;          // ビームの速度を設定
494     p_e_beam1 = NULL;                  // 発射可能なビームが無い
495   }
496 }
497
498
499 //=============================================================
500 // 色の指定
501 //=============================================================
502 void set_color(void)
503 {
504   glClearColor(0.0, 0.0, 0.0, 1.0);          //赤緑青と透明度
505 }
```

実行結果
図 1 に示すインベーダーゲームで遊ぶことができる．

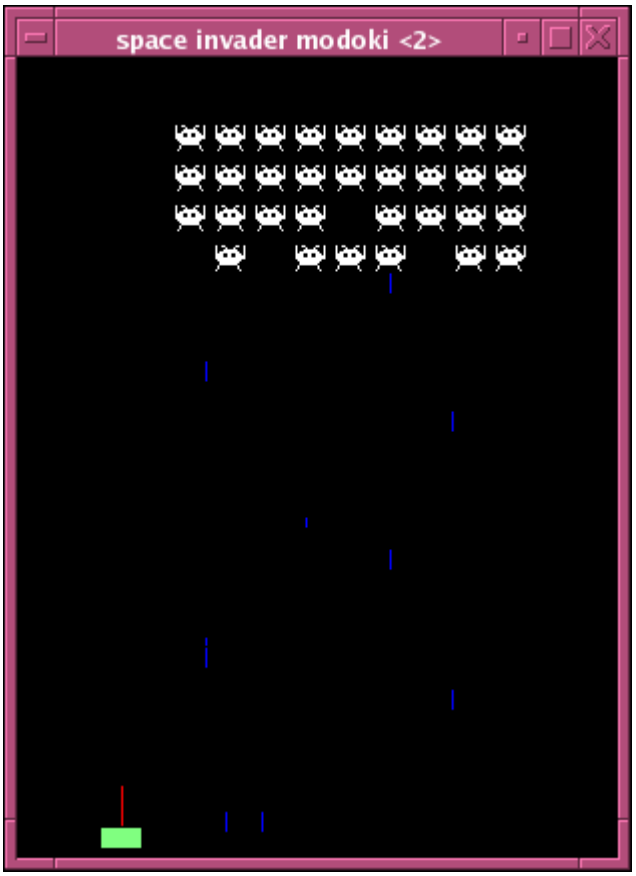

図 1: リスト 1 のプログラムの実行結果．インベーダーゲームができる．

2.2 新規の関数

これまでと比べて，新たに使用した関数を示す．

2.2.1 OpenGL 関係の関数

ここのインベーダーのプログラムでは，これまで学習してきたものに加えて，以下の関数を新たに使っている．最初の 2 つがキーボードとマウスイベントのコールバック関数を登録する関数である．GLUT(OpenGL) には，これらの他にも，これらのイベントに関する便利な関数がある．必要になったときにしれベルト良いだろう．

void glutKeyboardFunc(vod (func*)(**unsigned int** *key*, **int** *x*, **int** *y*))**

特殊キーを除く[1]キーを押したときに実行されるコールバック関数を登録する．*key* は押されたキー，*x* と *y* はキーを押したときのマウスの座標である．

void glutPassiveMotionFunc(vod (func*)(**int** *x*, **int** *y*))**

マウスボタンが押されていない状態で，マウスが移動したときに実行されるコールバック関数を登録する．*x* と *y* はキーを押したときのマウスの座標である．

void glRasterPos2i(int *x*, **int** *y*)

現在のラスター位置 (次のビットマップを描画する原点) を設定する．*x* と *y* はラスター位置の座標である．

void glutBitmapCharacter(void ****font*, **int *char*)

ひとつの文字を描く関数．*font* はフォントの指定．char は描く文字．

2.2.2 乱数

乱数とは，バラバラな数列のことを言う．とくに，自然数がめちゃくちゃに現れるようなものを自然乱数という．0 以上無限大までの全ての自然数を用いた自然乱数が考えられるが，実際上は最大の自然数を決め，その範囲で考えることが多い．我々が使用しているシステムの C 言語の場合，0〜2147483647 の範囲[2]の乱数を発生させることができる．

この乱数は，ゲームのプログラムでは必須の項目であろう．ここでは，インベーダーがビームの発射は乱数で決めている．

C 言語のプログラムでは，rand() 関数を使って，乱数を発生させる．例えば，次のようにすると，rand() 関数が呼び出される度に，それが乱数を返し，配列 a[i] に格納される．

```
for(i=0; i<ndata; i++){
  a[i]=rand();
}
```

コンピューターは正確に言われたとおり (プログラムのとおり) に計算を行うことは，諸君もよく知っているはずである．そのため，コンピューターはめちゃくちゃな順序で数が並んでいる乱数を発生させることは苦手である．先ほどの rand() 関数は，ある初期値[3]を使って，計算により乱数を決めている．同じ初期値を使って，rand() 関数を呼び出すと，同じ数列が発生するこのになる．これでは，乱数とは言い難いので，初期値を毎回変更するのがよい．

初期値も値が毎回異なる整数を決める必要があるが，現在の暦時刻を返す time() 関数を用いるのが一般的である．初期値の設定は，srand() 関数に引数 (符号無し整数) を渡すことにより可能である．次のようにすれば，毎回異なる初期値を決めることができる．

```
srand((unsigned int)time(NULL));
```

[1] ASCII 文字を生成するキーのこと
[2] 0〜$2^3$1-1 の範囲である．
[3] 正確には seed(種) と言うらしい．

ただし，1 秒以内であれば，time は同じ値となり，同じ初期値となり，同じ乱数となることに注意が必要である．(unsigned int) は，キャストと呼ばれる強制型変換で，引き続く値の型を変換している．time() 関数の引数は暦時刻で，暦時刻がポインターで格納される．暦時刻を格納する必要がないときには，NULL と空ポインターを指定する．

以上から，乱数を発生させるためには，rand() と srand()，time() 関数が必要であることが分かった．これらの関数を使うためには，関数の宣言が書かれているヘッダーファイルが必要である．rand() と srand() には stdlib.h が，time() には time.h が必要となる．以上をまとめると，配列 a[i] に 1024 個の乱数を発生させるためには，次のように書く．

```
#include <stdlib.h>
#include <time.h>

int main(void){
  int a[1024], i;

  srand((unsigned int)time(NULL));

  for(i=0; i<1024; i++){
    a[i]=rand();
  }

  return 0;
}
```

リスト 1 のプログラムでは，108 行目で乱数の初期値を決めている．そして，415 行目で乱数を発生させて，インベーダーがビームを発射するか，否かを決めている．415 行目は，

```
rand()%P_I_BEAM == 0
```

となっており，rand()%P_I_BEAM は，P_I_BEAM 分の 1 の確率で 0 になる．%は割ったあまりを計算する演算子である．

3 プログラム作成の練習

[**練習 1**] リスト 1 のプログラムを実行させて，マウスとキーボード イベントを体験せよ．

[**練習 2**] リスト 1 のプログラムのゲームに関するパラメーター (7–24 行の#define) を変えて，プログラムの実行状態を変えてみよ．

[**練習 3**] リスト 1 のプログラムの中で，マウスイベントとキーボード イベントの部分を探し出せ．そして，その内容を理解せよ．

[**練習 4**] リスト 1 のプログラムを適当に改造して，実行してみよ．