組み込みオペレーティングシステム

課題２レポート

情報科学類３年 201711362 小林瑞季

1. 作成したプログラムの動作(使い方)の説明
   ブロック崩しを行うプログラムで、START ボタンを押すとゲームがスタートする。
   十字キーの右左でラケットを操作し、ボールを落とさないようにラケットで返し、ブロックを崩していく。
   ボールが落ちてしまった場合、START ボタンを押すことで復活できる。しかし、3 回ボールが落ちてしまうと失敗となり、GAME OVER!画面が表示される。
   ブロックを全て崩すとクリアとなり、CLEAR!画面が表示される。
   CLEAR!画面、GAME OVER!画面で START ボタンを押すと初期状態からゲームがスタートする。

2. 作成したプログラムの内部の説明
   main.c について
   このプログラムについては、ほとんど資料 No.5 と同じであり、資料 No.8 を参考に、ちらつきの抑止を行う wait()関数を実装した。

   box.c について
   draw_box(),move_box()は資料 No.3 の関数を使った。cross()については、box2 つ(ボールとラケット)を引数に取り、ボールとラケットの領域が重なっているかどうかを判定するための関数である。ボールとラケットにおいて、それぞれの頂点の位置関係から重なっているかどうかを判定し、重なっていなければ 3 を、重なっていれば 0 か 1 か 2 を返すようにした。ラケットからボールが左側にはみ出ていれば 0 を、左にも右にもはみ出ていなければ 1 を、右側にはみ出ていれば 2 を返すようにしている。なぜこうしたかについては、racket.のところで説明する。

このプログラムでは、cross()以外については、ほとんど実験資料を参考にした。

ball.c について
dx は横方向の速度、dy は縦方向の速度、ball はボールの大きさと現在の位置、firstball はボールの初期位置をそれぞれ表す変数である。
first_ball()は、ボールを初期位置に描画するための関数である。
ball_get_dy(),ball_set_dy(),ball_get_box は資料 No.5 の関数を使った。また、ball_get_dx()と ball_set_dx()は、ball_get_dy(),ball_set_dy()内の dy を dx に変えただけである。ball_get_dx()と ball_set_dx()を実装した理由は、ラケットやブロックとのあたり方によって x 方向の速度も変えたかったためである。
ball_step()は資料 No.6 を参考に作成した。START 状態、RESTART 状態では、draw_box()を使って最後にあったボールを消し、first_ball()を使って初期位置にボールを表示するようにした。RUNNING 状態では、move_box()を使って速度分ボールを動かし、画面の左端に当たると dx を反転し、画面の上端に当たると dy を反転するように if 文を用いて実装した。DEAD 状態では、3 回目の死亡ならば draw_box()を使って最後にあったボールを消すようにした。CLEAR 状態では、draw_box()を使って最後にあったボールを消すようにした。

racket.c について
COLOR_WHITE は白色、COLOR_BLACK は黒色を定義したものである。
racket はラケットの大きさと現在の位置、firstracket はラケットの初期位置をそれぞれ表す変数である。
first_racket()は、ラケットを初期位置に描画するための関数である。
racket_step()は資料 No.6 を参考に作成した。key は押されているキーを表す変数である。START 状態、RESTART 状態では、draw_box()を使って最後にあったラケットを消し、first_ball()を使って初期位置にラケットを表示するようにした。RUNNING 状態では、左キーが押されている場合は左に、右キーが押されている場合は右にラケットを動かすようにした。また、rex は現在のボールの横方向の速度、rey は現在のボールの縦方向の速度を表す

変数である。cross()を用いてボールとラケットが重なってるかを判定し、0が返ってきてかつボールが右方向に移動していたら dx,dy を反転し、左方向に移動していたら dy のみ反転する。1 が返ってきたら dy のみ反転する。2が返ってきてかつボールが左方向に移動していたら dx,dy を反転し、右方向に移動していたら dy のみ反転する。3 が帰ってきたらなにもしない。これはボールの動きをプレイヤーが意図的に変更できた方がゲーム性が出ると考えて、ラケットとボールの位置関係によって cross()の返り値を変更するような実装を行った。DEAD 状態では、3 回目の死亡ならば draw_box()を使って最後にあったラケットを消すようにした。CLEAR 状態では、draw_box()を使って最後にあったラケットを消すようにした。

block.c について
COLOR_WHITE は白色、COLOR_BLACK は黒色、BLOCK_TOP はブロックの最上行表示する y 座標、BLOCK_WIDTH は 1 個のブロックの幅、BLOCK_HEIGHT は 1 個のブロックの高さを定義したものである。
yoko はブロックの列数、tate はブロックの行数、boxes,flags,num_blocks は資料 No.7 に同じもの、null はブロックを消すための box、updown はボールが今後上下どちらに移動するかを決定するための、rightleft はボールが今後左右どちらに移動するかを決定するための、delete1, delete2,delete3,delete4 はそれぞれボールの左上、右上、左下、右下がブロックに衝突したかどうかを判定するための、nowball は現在のボールの状態を表す変数である。
block_get_num()は他のプログラムが現在のブロック数を取得するための関数である。
first_blocks()はブロックを初期状態にする関数である。
delete_blocks()は、残っているブロックを全消しするための関数である。
hit()は、ボールの頂点からブロックとボールが重なっているかを判定し、重なっているかつブロックが残っていたら 1 を返し、それ以外の場合は 0 を返す関数である。
block_step()は、START 状態、RESTART 状態では、first_blocks()を使ってブロックを初期状態にするようにした。RUNNING 状態では、最初は updown,rightleft,delete1, delete2, delete3, delete4 をそれぞれ 0 に設定し、

nowball にはボールの現在の位置を格納する。hit()を用いてボールの 4 頂点とブロックの衝突判定を行い、衝突した場合はそれぞれ対応する delete をインクリメントする。updown については上の 2 点が衝突した場合はデクリメントし、下の 2 点が衝突した場合はインクリメントする。最終的に、updown が 0 でなかったらボールの向きを反転し、0 だったらそのままにする。rightleft については右の 2 点が衝突した場合はデクリメントし、左の 2 点が衝突した場合はインクリメントする。最終的に、rightleft が 0 でなかったらボールの向きを反転し、0 だったらそのままにする。それぞれの delete について、1 だったら衝突したブロックの消去(ブロックを消すのと、flags を 0 にする)とブロックの数を減らす操作を行い、0 だったらなにもしない。
rex,rey については racket.c の説明と同様である。DEAD 状態では、3 回目の死亡ならば delete_blocks()を使ってブロックを全て消すようにした。
CLEAR 状態では、ブロックが全て消えている状態なので、何もしない。
このプログラムについては、ほとんど実験資料を参考にした。

game.c について
COLOR_WHITE は白色、COLOR_BLACK は黒色、FONT_SIZE は文字の大きさを 8*8 に定義したものである。
current_state はゲームの状態、count は死んだ回数、fb は VRAM のアドレスを表す変数である。
draw_char()は資料 No.2 の関数を使った。
game_over()と game_clrar()は、draw_char を使ってそれぞれ画面のなるべく中央に GAME OVER!と CLEAR!を指定された色で表示させるための関数である。
game_get_count()は、他のプログラムが現在何回死んでいるかを取得するための関数である。
game_step()は資料 No.6 を参考に作成した。key は押されているキーを表す変数である。START 状態では、GAME OVER!と CLEAR!を消し、START ボタンが押されると RUNNING 状態に移行するようにしている。
RUNNING 状態では、ボールの底面の位置を取得し、ボールが画面の下についていたら DEAD 状態に移行する。また、ブロックが全て消えたら CLEAR 状態に移行する。DEAD 状態では、死んだ回数が 3 回になったと

き、GAME_OVER!画面が表示される。死んだ回数が 3 回のときに START ボタンが押されると START 状態に移行し、3 回未満のときに START ボタンが押されると RESTART 状態に移行する。RESTART 状態では、START ボタンが押されると RUNNING 状態に移行する。CLEAR 状態では、CLEAR! 画面を表示し、START ボタンが押されたら死んだ回数をリセットし、START 状態に移行する。

今までのプログラムで最後にボールやラケット、ブロックを消していたのは、最後に文字画面のみを表示させたかったからである。

3. プログラムのリスト

main.c

```
#include "gba.h"
#include "box.h"
#include "game.h"
#include "ball.h"
#include "racket.h"
#include "block.h"

#define COLOR_WHITE BGR(31, 31, 31)
#define COLOR_BLACK 0

void wait_until_vblank(void) {
  while ((gba_register(LCD_STATUS) & 1) == 0)
  ;
}
void wait_while_vblank(void) {
  while ((gba_register(LCD_STATUS) & 1))
  ;
}

int main(void)
{
  gba_register(LCD_CTRL) = LCD_BG2EN | LCD_MODE3;

  int key;

  // 画面を初期化
  // タイマーを初期化
  while (1) {
    hword begin = gba_register(TMR_COUNT0);
    key = gba_register(KEY_STATUS);
```

```
            wait_until_vblank();  // 垂直ブランク期間になるまで待つ

            ball_step();
            racket_step();
            block_step();
            game_step();

            wait_while_vblank();  // 垂直ブランク期間が終るまで待つ
        }
}
```

box.c

```
#include "gba.h"
#include "box.h"

#define COLOR_WHITE BGR(31, 31, 31)
#define COLOR_BLACK 0

void draw_box(struct box *b, int x, int y, hword color){
  hword *base, *d;
  int w,h;

  /* Base point is at (x, y). */
  base = (hword*)VRAM + LCD_WIDTH * y + x;

  /* Draw box from (x, y). */
  for (h = b->height; h > 0; h--) {
    d = base;
    for (w = b->width; w > 0; w--) *(d++) = color;
    base += LCD_WIDTH;
  }

  /* Set the current position. */
  b->x = x;
  b->y = y;
}

void move_box(struct box *b, int x, int y, hword color){
  draw_box(b, b->x, b->y, COLOR_BLACK);
  draw_box(b, x, y, color);
}

int cross(struct box *b1, struct box *b2){
```

```
        // b1 と b2 の領域が重なっていれば 1 以上，重なっていなければ 0 を返す
        if(b1->y <= b2->y + b2->height &&
            b1->y + b1->height >= b2->y &&
            b1->x <= b2->x + b2->width &&
            b1->x + b1->width >= b2->x){
            if(b2->x <= b1->x && b1->x <= b2->x + b2->width){
                return 0;
            }else if(b1->x <= b2->x && b2->x + b2->width <= b1->x + b1->width){
                return 1;
            }else if(b2->x <= b1->x + b1->width && b1->x + b1->width <= b2->x + b2->width){
                return 2;
            }
        }else{
            return 3;
        }
}
```

box.h

```
#ifndef BOX_H //二重で include されることを防ぐ
#define BOX_H

struct box {
   int x, y;
   int width, height;
};

void draw_box(struct box *b, int x, int y, hword color);
void move_box(struct box *b, int x, int y, hword color);
int cross(struct box *b1, struct box *b2);

#endif
```

ball.c

```
#include "gba.h"
#include "box.h"
#include "game.h"
#include "ball.h"
#include "racket.h"

#define COLOR_WHITE BGR(31, 31, 31)
#define COLOR_BLACK 0

static int dx = 2;
static int dy = 3;              /* ボールの現在の速度 */
static struct box ball = {25,129,9,9};          /* ボールの箱の現在の位置 */
static struct box firstball = {25,129,9,9};

void first_ball(){
  dx = 2;
  dy = 3;              /* ボールの現在の速度 */
  ball = firstball;
  draw_box(&ball,ball.x,ball.y,COLOR_WHITE);
}

int ball_get_dy() { return dy; }
int ball_get_dx() { return dx; }
void ball_set_dy(int new_dy) { dy = new_dy; }
void ball_set_dx(int new_dx) { dx = new_dx; }
struct box *ball_get_box() { return &ball; }

void ball_step(void)
{
  switch (game_get_state()) {
```

```
            case START:
               draw_box(&ball, ball.x, ball.y, COLOR_BLACK);
               //ボールの位置，速度を初期状態にし，ボールを表示する.
               first_ball();
               break;
            case RUNNING:
               //ボールのアニメーションを 1 ステップ行なう.
               move_box(&ball,ball.x + dx,ball.y + dy,COLOR_WHITE);
               if(ball.x < 0 || ball.x + ball.width > 240){
                  dx = -dx;
               }

               if(ball.y < 0){
                  dy = -dy;
               }
               break;
            case DEAD:
               if(game_get_count() == 2){
                  draw_box(&ball, ball.x, ball.y, COLOR_BLACK);
               }
               break;
            case RESTART:
               //現在のボールを画面から消し,
               draw_box(&ball, ball.x, ball.y, COLOR_BLACK);
               //ボールの位置，速度を初期状態にし，ボールを表示する.
               first_ball();
               break;
            case CLEAR:
               draw_box(&ball, ball.x, ball.y, COLOR_BLACK);
               break;
            }
}
```

ball.h

```
#ifndef BALL_H //二重で include されることを防ぐ
#define BALL_H

int ball_get_dy(void);                  // ボールの y 方向の速度を
返す.
int ball_get_dx(void);                  // ボールの y 方向の速度を
返す.
void ball_set_dy(int new_dy);           // ボールの y 方向の速度を
セットする.
void ball_set_dx(int new_dx);           // ボールの x 方向の速度を
セットする.
struct box *ball_get_box(void);         // ボールの箱の位置を返す.
void ball_step(void);                   // アニメーションの 1 ステ
ップを行なう. p(void);

#endif
```

racket.c

```
#include "gba.h"
#include "box.h"
#include "game.h"
#include "ball.h"
#include "racket.h"

#define COLOR_WHITE BGR(31, 31, 31)
#define COLOR_BLACK 0

static struct box racket = {20,140,20,2};          /* ボールの箱の
現在の位置 */
static struct box firstracket = {20,140,20,2};

void first_racket(){
  racket = firstracket;
  draw_box(&racket,racket.x,racket.y,COLOR_WHITE);
}

void racket_step(void)
{
  int key = gba_register(KEY_STATUS);

  switch (game_get_state()) {
  case START:
    draw_box(&racket, racket.x, racket.y, COLOR_BLACK);
    //ボールの位置，速度を初期状態にし，ボールを表示する.
    first_racket();
    break;
  case RUNNING:
    //ラケットのアニメーションを 1 ステップ行なう.
    draw_box(&racket, racket.x, racket.y, COLOR_WHITE);
```

```
        if (! (key & KEY_LEFT)){
          move_box(&racket,racket.x - 7,racket.y,
COLOR_WHITE);
        }else if(! (key & KEY_RIGHT)){
          move_box(&racket,racket.x + 7,racket.y,
COLOR_WHITE);
        }

        int rey = ball_get_dy();
        int rex = ball_get_dx();

        switch(cross(&racket,ball_get_box())){
          case 0:
            if(rex > 0){
              ball_set_dx(-rex);
            }
            ball_set_dy(-rey);
            break;
          case 1:
            ball_set_dy(-rey);
            break;
          case 2:
            if(rex < 0){
              ball_set_dx(-rex);
            }
            ball_set_dy(-rey);
            break;
          case 3:
            break;
        }
        break;
      case DEAD:
```

```
                if(game_get_count() == 2){
                    draw_box(&racket, racket.x, racket.y, COLOR_BLACK);
                }
                break;
            case RESTART:
                //現在のラケットを画面から消し,
                draw_box(&racket, racket.x, racket.y, COLOR_BLACK);
                //ラケットの位置，速度を初期状態にし，ラケットを表示する.
                first_racket();
                break;
            case CLEAR:
                draw_box(&racket, racket.x, racket.y, COLOR_BLACK);
                break;
            }
}
```

racket.h

```
#ifndef RACKET_H //二重で include されることを防ぐ
#define RACKET_H

extern void racket_step(void);                    // アニメーション
の 1 ステップを行なう．p(void);
void first_racket();

#endif
```

block.c

```
#include "gba.h"
#include "ball.h"
#include "game.h"
#include "box.h"
#include "block.h"

#define COLOR_WHITE BGR(31, 31, 31)
#define COLOR_BLACK 0
#define BLOCK_TOP 0
#define BLOCK_WIDTH 48
#define BLOCK_HEIGHT 10

static int yoko = 5;
static int tate = 3;

static struct box boxes[5][3];
static char flags[5][3];
static int num_blocks = 15;
static struct box null = {0,0,0,0};
int updown,rightleft,delete1,delete2,delete3,delete4;

struct box *nowball;

int block_get_num() { return num_blocks; }

void first_blocks(){
   int l,r;
   for(l = 0; l < yoko; l++){
      for(r = 0; r < tate; r++){
         struct box box = {l * BLOCK_WIDTH, BLOCK_TOP + r
* BLOCK_HEIGHT, BLOCK_WIDTH - 1, BLOCK_HEIGHT -
```

```
1};
            boxes[l][r] = box;
            flags[l][r] = 1;

draw_box(&boxes[l][r],boxes[l][r].x,boxes[l][r].y,COLOR_WHI
TE);
        }
    }
    num_blocks = 15;
}

void delete_blocks(){
    int l,r;
    for(l = 0; l < yoko; l++){
        for(r = 0; r < tate; r++){
            struct box box = {l * BLOCK_WIDTH, BLOCK_TOP + r
* BLOCK_HEIGHT, BLOCK_WIDTH - 1, BLOCK_HEIGHT -
1};
            boxes[l][r] = box;

draw_box(&boxes[l][r],boxes[l][r].x,boxes[l][r].y,COLOR_BLA
CK);
        }
    }
}

static int hit(int x, int y){
    int i = x / BLOCK_WIDTH;
    int j = (y - BLOCK_TOP) / BLOCK_HEIGHT;

    if(0 <= i && i < yoko){
        if(0 <= j && j < tate){
```

```
            if(flags[i][j] == 1){
               return 1;
            }else{
               return 0;
            }
         }else{
            return 0;
         }
      }else{
         return 0;
      }
   }

   void block_step(void)
   {
      switch (game_get_state()) {
      case START:
         first_blocks();
         break;
      case RUNNING:
         updown = 0;
         rightleft = 0;
         delete1 = 0;
         delete2 = 0;
         delete3 = 0;
         delete4 = 0;

         nowball = ball_get_box();
         if(hit(nowball->x,nowball->y) == 1){
            struct box hitbox = boxes[nowball->x /
   BLOCK_WIDTH][(nowball->y - BLOCK_TOP) /
   BLOCK_HEIGHT];
```

```
            draw_box(&hitbox,hitbox.x,hitbox.y,COLOR_BLACK);
            updown--;
            rightleft++;
            delete1++;
        }
        if(hit(nowball->x + nowball->width,nowball->y) == 1){
            struct box hitbox = boxes[(nowball->x + nowball->width)
/ BLOCK_WIDTH][(nowball->y - BLOCK_TOP) /
BLOCK_HEIGHT];
            draw_box(&hitbox,hitbox.x,hitbox.y,COLOR_BLACK);
            updown--;
            rightleft--;
            delete2++;
        }
        if(hit(nowball->x + nowball->width,nowball->y + nowball-
>height) == 1){
            struct box hitbox = boxes[(nowball->x + nowball->width)
/ BLOCK_WIDTH][(nowball->y + nowball->height -
BLOCK_TOP) / BLOCK_HEIGHT];
            draw_box(&hitbox,hitbox.x,hitbox.y,COLOR_BLACK);
            updown++;
            rightleft--;
            delete3++;
        }
        if(hit(nowball->x,nowball->y + nowball->height) == 1){
            struct box hitbox = boxes[nowball->x /
BLOCK_WIDTH][(nowball->y + nowball->height -
BLOCK_TOP) / BLOCK_HEIGHT];
            draw_box(&hitbox,hitbox.x,hitbox.y,COLOR_BLACK);
            updown++;
            rightleft++;
            delete4++;
```

```
        }

        if(delete1 == 1){
          boxes[nowball->x / BLOCK_WIDTH][(nowball->y -
BLOCK_TOP) / BLOCK_HEIGHT] = null;
          if(flags[nowball->x / BLOCK_WIDTH][(nowball->y -
BLOCK_TOP) / BLOCK_HEIGHT] == 1){
            flags[nowball->x / BLOCK_WIDTH][(nowball->y -
BLOCK_TOP) / BLOCK_HEIGHT] = 0;
            num_blocks--;
          }
        }
        if(delete2 == 1){
          boxes[(nowball->x + nowball->width) /
BLOCK_WIDTH][(nowball->y - BLOCK_TOP) /
BLOCK_HEIGHT] = null;
          if(flags[(nowball->x + nowball->width) /
BLOCK_WIDTH][(nowball->y - BLOCK_TOP) /
BLOCK_HEIGHT] == 1){
            flags[(nowball->x + nowball->width) /
BLOCK_WIDTH][(nowball->y - BLOCK_TOP) /
BLOCK_HEIGHT] = 0;
            num_blocks--;
          }
        }
        if(delete3 == 1){
          boxes[(nowball->x + nowball->width) /
BLOCK_WIDTH][(nowball->y + nowball->height -
BLOCK_TOP) / BLOCK_HEIGHT] = null;
          if(flags[(nowball->x + nowball->width) /
BLOCK_WIDTH][(nowball->y + nowball->height -
BLOCK_TOP) / BLOCK_HEIGHT] == 1){
```

```
            flags[(nowball->x + nowball->width) /
BLOCK_WIDTH][(nowball->y + nowball->height -
BLOCK_TOP) / BLOCK_HEIGHT] = 0;
            num_blocks--;
          }
        }
        if(delete4 == 1){
          boxes[(nowball->x + nowball->width) /
BLOCK_WIDTH][(nowball->y + nowball->height -
BLOCK_TOP) / BLOCK_HEIGHT] = null;
          if(flags[nowball->x / BLOCK_WIDTH][(nowball->y +
nowball->height - BLOCK_TOP) / BLOCK_HEIGHT] == 1){
            flags[nowball->x / BLOCK_WIDTH][(nowball->y +
nowball->height - BLOCK_TOP) / BLOCK_HEIGHT] = 0;
            num_blocks--;
          }
        }

        int rey = ball_get_dy();
        int rex = ball_get_dx();
        if(updown != 0){
          ball_set_dy(-rey);
        }
        if(rightleft != 0){
          ball_set_dx(-rex);
        }
        break;
      case DEAD:
        if(game_get_count() == 2){
          delete_blocks();
        }
        break;
```

```
            case RESTART:
                break;
            case CLEAR:
                break;
            }
}
```

block.h

```
        #ifndef BLOCK_H //二重で include されることを防ぐ
        #define BLOCK_H

        void block_step(void);
        int block_get_num();

#endif
```

game.c

```
#include "gba.h"
#include "box.h"
#include "ball.h"
#include "block.h"
#include "game.h"
#include "8x8.til"

#define COLOR_WHITE BGR(31, 31, 31)
#define COLOR_BLACK 0
#define FONT_SIZE 8

static enum state current_state;   // 現在の状態
int count = 0;
hword *fb = (hword*)VRAM;

/*
* Draw a font of code with color.
* ptr specifies the font's top left corner.
*/
void draw_char(hword *ptr, hword color, int code){
   hword     *p;
   int         i, j;
   unsigned char     *font = char8x8[code];

   for (i = 0; i < FONT_SIZE; i++) {
      p = ptr + LCD_WIDTH * i;
      for (j = FONT_SIZE - 1; j >= 0; j--, p++) {
         if (font[i] & (1 << j)) *p = color;
      }
   }
}
```

```
void game_over(hword color){
   draw_char(fb + (LCD_WIDTH * 10) + 12316, color, 71);
   draw_char(fb + (LCD_WIDTH * 10) + 12324, color, 65);
   draw_char(fb + (LCD_WIDTH * 10) + 12332, color, 77);
   draw_char(fb + (LCD_WIDTH * 10) + 12340, color, 69);
   draw_char(fb + (LCD_WIDTH * 10) + 12348, color, 32);
   draw_char(fb + (LCD_WIDTH * 10) + 12356, color, 79);
   draw_char(fb + (LCD_WIDTH * 10) + 12364, color, 86);
   draw_char(fb + (LCD_WIDTH * 10) + 12372, color, 69);
   draw_char(fb + (LCD_WIDTH * 10) + 12380, color, 82);
   draw_char(fb + (LCD_WIDTH * 10) + 12388, color, 33);
}

void game_clear(hword color){
   draw_char(fb + (LCD_WIDTH * 10) + 12340, color, 67);
   draw_char(fb + (LCD_WIDTH * 10) + 12348, color, 76);
   draw_char(fb + (LCD_WIDTH * 10) + 12356, color, 69);
   draw_char(fb + (LCD_WIDTH * 10) + 12364, color, 65);
   draw_char(fb + (LCD_WIDTH * 10) + 12372, color, 82);
   draw_char(fb + (LCD_WIDTH * 10) + 12380, color, 33);
}

int game_get_count() { return count; }

enum state game_get_state(void) { return current_state; }
void game_set_state(enum state new_state) {
   current_state = new_state;
}

void game_step(void)
{
```

```
int key = gba_register(KEY_STATUS);

switch (game_get_state()) {
case START:
  game_clear(COLOR_BLACK);
  game_over(COLOR_BLACK);
  if (! (key & KEY_START)){
    game_set_state(RUNNING);
  }
  break;

case RUNNING:
  if (ball_get_box()->y + ball_get_box()->height > 160){
    game_set_state(DEAD);
  }
  if(block_get_num() == 0){
    game_set_state(CLEAR);
  }
  break;

case DEAD:
  if(count == 2){
    game_over(COLOR_WHITE);
  }
  if(! (key & KEY_START) && count < 3){
    count++;
    if(count == 3){
      count = 0;
      game_set_state(START);
    }else{
      game_set_state(RESTART);
    }
```

```
            }
            break;

        case RESTART:
            /* 次のティックは RUNNING 状態にする．*/
            if(! (key & KEY_START)){
                game_set_state(RUNNING);
            }
            break;

        case CLEAR:
            game_clear(COLOR_WHITE);
            if(! (key & KEY_START)){
                count = 0;
                game_set_state(START);
            }
            break;
        }
}
```

game.h

```
        #ifndef GAME_H //二重で include されることを防ぐ
        #define GAME_H

        enum state {START, RUNNING, DEAD, RESTART, CLEAR};

        extern int game_get_count();
        extern void game_step(void);                    // 1 ティックの動作
        を行なう.
        extern enum state game_get_state(void);  // 今の状態を問い合
        わせる.
        extern void game_set_state(enum state);  // 状態を変更する.

#endif
```