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。

1. 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。

1. 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\_WHITE);  }  }  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\_BLACK);  }  }  }  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 |