

テトリス

10 班

2020 年 1 月 26 日

1 要求仕様

目的 娯楽を通して，脳の活性化をはかる

品質目標 読みやすいコードを作成する

使用者の特性 全年齢対象，キーボードが使えること

作るもの テトリス

2 ルール

7種類のブロックからランダムに1つずつ落ちてくる。落ちてくるブロックを操作(回転・左右下移動)する。操作中のブロックが積み上げられたブロックに触れると固定され、次のブロックが落ちてくる。ブロックが固定されたときに1行揃ったらその行は、消える。消えた行より上の行は、1行詰められる。縦20行×横10列のフィールド上で行われる。ブロックは、上から落ちてくる。落ちてくるスピードは、1[block/s]程度。シンプルなテトリスを実装するため、回転によるはめ込み(T-Spin)や対戦機能等は実装しないこととする。ブロックの形は、4つの正方形を組み合わせたものである。スコアアップについては、1行消すごとに1行:1倍, 2行:1.25倍, 3行:1.5倍, 4行:1.75倍のように0.25倍ずつ増えることとする。

3 計画表

4 役割分担

- 物の抽出・振る舞抽出：秋元・尾畑
- データ構造・関数仕様：秋元・尾畑
- テスト計画：本田
- コード作成：篠田
- テスト：宮本・伊崎
- 考察・まとめ：江尻・伊崎
- 資料作成：篠田

5 テスト計画

テスト内容	1 回目	2 回目
落ちてくるブロックがランダムであるか	正常	正常
ブロックが 7 種類全て出現するか	正常	正常
ブロックがフィールドを出ないか	正常	正常
ブロックの操作 (下左右移動ができるか)	正常	正常
操作中のブロックが固定されるか	正常	正常
ブロック固定後に次のブロックが落ちてくるか	正常	正常
ブロックの落ちてくる速度が 1[block/sec] であるか	正常	正常
回転によるはめ込み等の複雑な処理が実装されていないか	正常	正常
ブロックが 1 行揃ったら消えるか	正常	正常
ブロックが消えたら 1 行詰められているか	正常	正常
ポイントアップがルール通りに行われているか	異常	正常

6 物の抽出とふるまい

6.1 物の抽出

- テトリミノ
- フィールド
- スコア

6.2 物の振る舞い

- テトリミノ：回転する，時間経過で落下，横に動く，生成する，表示する
- フィールド：列を消す，列を詰める，テトリミノを固定する
- スコア：加算する，表示する

7 データ構造

- ブロック

```
struct t_mino{  
    int mino[MINO_SIZE][MINO_SIZE]; // MINOSIZE=4 ミノの形状を表す  
    int x,y; // ミノの座標 (左上のマスを規準)  
}
```

- フィールド

```
int field[FIELD_Y][FIELD_X]; // FIELD_X=10, FIELD_Y=20
```

- スコア

```
int score;
```


8 関数仕様

関数名	void init_field();
引数	なし
戻り値	なし
内容	フィールドをすべて 0 で初期化

関数名	bool is_field_over(t_mino mino);
引数	mino : テトリミノの情報
戻り値	true : フィールド内である false : フィールド外である
内容	テトリミノがフィールド内であることを判定

関数名	bool is_field_over_y(t_mino mino);
引数	mino : テトリミノの情報
戻り値	true : フィールド内である false : フィールド外である
内容	テトリミノが y 軸でフィールド内であることを判定

関数名	bool is_field_over_x(t_mino mino);
引数	mino : テトリミノの情報
戻り値	true : フィールド内である false : フィールド外である
内容	テトリミノが x 軸でフィールド内であることを判定

関数名	bool is_side_hit (t_mino mino);
引数	mino : テトリミノの情報
戻り値	true : テトリミノがブロックとかぶっている false : テトリミノがブロックとかぶっていない
内容	テトリミノが x 軸でブロックと被っているかを判定

関数名	bool is_down_hit (t_mino mino);
引数	mino : テトリミノの情報
戻り値	true : テトリミノがブロックとかぶっている false : テトリミノがブロックとかぶっていない
内容	テトリミノが y 軸でブロックと被っているか判定

関数名	void row_check();
引数	なし
戻り値	なし
内容	すべての列に対して揃っているか確認する

関数名	void pack_block(int row);
引数	row : 列の番号
戻り値	なし
内容	指定された列より上の列を 1 段詰める

関数名	void row_clear(int row);
引数	row : 列の番号
戻り値	なし
内容	指定された列を消去する

関数名	void stack_block(t_mino mino);
引数	mino : テトリミノの情報
戻り値	なし
内容	ミノをフィールドに固定する

関数名	void disp_field();
引数	なし
戻り値	なし
内容	フィールドを表示する

関数名	void disp(t_mino mino)
引数	mino : テトリミノの情報
戻り値	なし
内容	ゲーム画面の表示

関数名	void disp_game_over()
引数	なし
戻り値	なし
内容	ゲームオーバー画面の表示

関数名	void initialize(t_mino *mino)
引数	*mino : ミノの情報
戻り値	なし
内容	ゲームの初期化

関数名	void move_block(t_mino *mino, char key[256])
引数	*mino : ミノの情報 key[256] : キーの情報
戻り値	なし
内容	テトリミノ動かす

関数名	void rotate_block(t_mino *mino)
引数	*mino : ミノの情報
戻り値	なし
内容	テトリミノを回転する

関数名	void make_t_mino(t_mino *mino)
引数	*mino : ミノの情報
戻り値	なし
内容	テトリミノを作成する

関数名	void move_right(t_mino *mino)
引数	*mino : ミノの情報
戻り値	なし
内容	テトリミノを右に動かす

関数名	void move_left(t_mino *mino)
引数	*mino : ミノの情報
戻り値	なし
内容	テトリミノを左に動かす

関数名	void move_down(t_mino *mino)
引数	*mino : ミノの情報
戻り値	なし
内容	テトリミノを下に動かす

関数名	void disp_t_mino(t_mino *mino)
引数	*mino : ミノの情報
戻り値	なし
内容	テトリミノを表示する

関数名	void init_score()
引数	なし
戻り値	なし
内容	スコアの初期化

関数名	void add_score(int lines)
引数	lines : 消した列の数
戻り値	なし
内容	消えた列数に応じたスコアの加算

関数名	void disp_score()
引数	なし
戻り値	なし
内容	スコアの表示

9 作成後の批評・考察

10 files/coment

11 プログラムリスト

リスト 1 define.hpp

```
1 #ifndef    DEFINE_HPP
2 #define    DEFINE_HPP
3
4 // スクリーンの大きさ
5 #define    SCREEN_X 240
6 #define    SCREEN_Y 440
7
8 // 表示のオフセット
9 #define    OFFSET_X 20
10 #define    OFFSET_Y 20
11
12 // テトリミノの種類
13 #define    T_MINO_TYPE 7
14 #define    TYPE_1 0
15 #define    TYPE_2 1
16 #define    TYPE_3 2
17 #define    TYPE_4 3
18 #define    TYPE_5 4
19 #define    TYPE_6 5
20 #define    TYPE_7 6
21
22 // テトリミノのサイズ
23 #define    MINO_SIZE 4
24
25 // フィールドの大きさ
26 #define    FIELD_X 10
27 #define    FIELD_Y 20
28
29 // ブロックのサイズ
30 #define    BLOCK_SIZE 20
31
32 // 移動方向
33 #define    RIGHT 1
34 #define    LEFT 2
35 #define    DOWN 3
36 #define    NONE 0
37
38 // ブロックの存在を表す
39 #define    EMPTY 0
40 #define    EXIST 1
41
42 // 加算するスコア
43 #define    ADD_SCORE 20
44 #define    SCORE_BIAS 0.25
45
46 // キー操作確認時間間隔
47 #define    KEY_CHECK_TIME 0.175
48
49 // 落下時間
50 #define    DROP_CHECK_TIME 1.0
51
52 #endif
```

リスト 2 field.cpp

```
1 #include "score.hpp"
2 #include "field.hpp"
3 #include "define.hpp"
4 #include "objects.hpp"
5 #include "DxLib\DxLib.h"
6
7 // フィールドの初期化
8 void init_field(){
9     int x,y;
```

```

10
11 // フィールドを全て空にする
12 for(y = 0; y < FIELD_Y; y++){
13     for(x = 0; x < FIELD_X; x++){
14         field[y][x] = EMPTY;
15     }
16 }
17 }
18
19 /*
20 ミノがフィールドを超えていないか確認する
21 */
22 bool is_field_over(t_mino mino){
23
24     if(is_field_over_x(mino)) return true;
25     if(is_field_over_y(mino)) return true;
26
27     return false;
28 }
29
30 /*
31 y軸(縦)に対してミノが範囲を超えていないか確認する
32 */
33 bool is_field_over_y(t_mino mino){
34     int i, j;
35
36     // ミノを全て調べる
37     for(i = 0; i < MINO_SIZE; i++){
38         for(j = 0; j < MINO_SIZE; j++){
39             if(mino.mino[i][j] == EXIST){ // ミノが存在するところ
40                 if(mino.y + i < 0 || mino.y + i >= FIELD_Y) return true; // 範囲を超
41                     えていないか確認
42             }
43         }
44     }
45     return false;
46 }
47
48 /*
49 x軸(横)に対してミノが範囲を超えていないか確認する
50 */
51 bool is_field_over_x(t_mino mino){
52     int i, j;
53
54     // ミノを全て調べる
55     for(i = 0; i < MINO_SIZE; i++){
56         for(j = 0; j < MINO_SIZE; j++){
57             if(mino.mino[i][j] == EXIST){ // ミノが存在するところ
58                 if(mino.x + j < 0 || mino.x + j >= FIELD_X) return true; // 範囲を超
59                     えていないか
60             }
61         }
62     }
63     return false;
64 }
65
66 /*
67 x軸でミノとフィールドのブロックが被っていないか確認する
68 */
69 bool is_side_hit(t_mino mino){
70     int i, j;
71
72     // ミノの全てを調査
73     for(i = 0; i < MINO_SIZE; i++){
74         for(j = 0; j < MINO_SIZE; j++){
75             if(mino.mino[i][j] == EXIST){ // ミノが存在するところ
76                 if(field[mino.y + i][mino.x + j] == EXIST) return true; // フィールド
77                     と被っていたら
78             }
79         }
80     }
81 }

```

```

80
81     return false;
82 }
83
84 /*
85  y 軸でミノとフィールドのブロックが被っていないか確認する
86 */
87 bool is_down_hit(t_mino mino){
88     int i,j;
89
90     for(i = 0;i < MINO_SIZE;i++){
91         for(j = 0;j < MINO_SIZE;j++){
92             if(mino.mino[i][j] == EXIST){
93                 if(field[mino.y + i][mino.x + j] == EXIST) return true;
94             }
95         }
96     }
97     return false;
98 }
99
100 /*
101   ブロックを固定する
102 */
103 void stack_block(t_mino mino){
104     int i,j;
105
106     // ミノ全てに対して調査
107     for(i = 0;i < MINO_SIZE;i++){
108         for(j = 0;j < MINO_SIZE;j++){
109             if(mino.mino[i][j] == EXIST){ // ミノが存在していたら
110                 field[mino.y + i][mino.x + j] = EXIST; // ミノをフィールドに固定
111             }
112         }
113     }
114 }
115
116 /*
117   列がそろっているかの確認を全ての列に対して行う
118 */
119 void row_check(){
120     int i,j;
121     int count_lines = 0; // そろった列をカウントする用
122
123     for(i = FIELD_Y-1;i >= 0;){
124         for(j = 0;j < FIELD_X;j++){
125             if (field[i][j] == EMPTY) break; // 1つでも空があったら抜ける
126         }
127         if(j == FIELD_X){ // 1列全てが埋まっていたら
128             row_clear(i); // 列を削除
129             pack_block(i); // 列を詰める
130             count_lines++; // そろった列のカウントを進める
131         }else{
132             i--;
133         }
134     }
135
136     if(count_lines != 0){
137         add_score(count_lines);
138     }
139 }
140
141 /*
142   空になった列を詰める
143 */
144 void pack_block(int row){
145     int i,j;
146
147     // 現在の列よりも上の列を1つ落とす
148     for(i = row-1;i >= 0;i--){
149         for(j = 0;j < FIELD_X;j++){
150             field[i+1][j] = field[i][j];
151         }
152     }

```



```

153 }
154
155 /*
156   そろった列を消す
157 */
158 void row_clear(int row){
159     int i;
160
161     // 指定された列を空にする
162     for(i = 0; i < FIELD_X; i++){
163         field[row][i] = EMPTY;
164     }
165 }
166
167 /*
168   フィールド表示用
169 */
170 void disp_field(){
171     int x,y;
172     unsigned int Color = GetColor(255,255,255); // フィールドの色
173
174     for(y = 0; y < FIELD_Y; y++){
175         for(x = 0; x < FIELD_X; x++){
176             if(field[y][x] == EMPTY){ // フィールドが空だったら塗りつぶさない
177                 // 指定座標にブロックを描画
178                 DrawBox(x*BLOCK_SIZE+OFFSET_X, y*BLOCK_SIZE+OFFSET_Y,
179                     (x+1)*BLOCK_SIZE+OFFSET_X, (y+1)*BLOCK_SIZE+OFFSET_Y, Color, false);
180             } else{ // フィールドが空でなかったら塗りつぶす
181                 // 指定座標にブロックを描画
182                 DrawBox(x*BLOCK_SIZE+OFFSET_X, y*BLOCK_SIZE+OFFSET_Y,
183                     (x+1)*BLOCK_SIZE+OFFSET_X, (y+1)*BLOCK_SIZE+OFFSET_Y, Color, true);
184             }
185         }
186     }
187 }

```

リスト 3 field.hpp

```

1 #ifndef FIELD_HPP
2 #define FIELD_HPP
3
4 #include "objects.hpp"
5
6 // フィールド初期化
7 void init_field();
8
9 // ミノがフィールド内か確認
10 bool is_field_over(t_mino mino);
11
12 // ミノが y 軸でフィールド内か確認
13 bool is_field_over_y(t_mino mino);
14
15 // ミノが x 軸でフィールド内か確認
16 bool is_field_over_x(t_mino mino);
17
18 // x 軸でミノがフィールドのブロックと被っていないか確認
19 bool is_side_hit(t_mino mino);
20
21 // y 軸でミノがフィールドのブロックと被っていないか確認
22 bool is_down_hit(t_mino mino);
23
24 // 列がそろっているのかの確認
25 void row_check();
26
27 // 列を1段下に詰める
28 void pack_block(int row);
29
30 // 指定された列を消去する
31 void row_clear(int row);
32
33 // ブロックを固定化する
34 void stack_block(t_mino mino);

```

```

35 // フィールドを表示する
36 void disp_field();
37
38
39 #endif // DISPLAY_HPP

```

リスト 4 game_screen.cpp

```

1 #include "objects.hpp"
2 #include "score.hpp"
3 #include "mino.hpp"
4 #include "field.hpp"
5 #include "DxLib/DxLib.h"
6
7 // ゲーム画面描画
8 void disp(t_mino mino){
9     // フィールドの描画
10    disp_field();
11    // ミノの描画
12    disp_t_mino(&mino);
13    // スコアの描画
14    disp_score();
15 }
16
17 // ゲームオーバー画面
18 void disp_game_over(){
19     TCHAR gameover_str[] = _T("GAME OVER"); // ゲームオーバーの文字列
20     int Green = GetColor( 0, 255, 0 ); // 文字色
21
22     DrawString( SCREEN_X/2-40, SCREEN_Y/2, gameover_str, Green); // ゲームオーバー文
        字の表示
23 }

```

リスト 5 game_screen.hpp

```

1 #ifndef GAME_SCREEN_HPP
2 #define GAME_SCREEN_HPP
3
4 #include "objects.hpp"
5
6 // ゲーム画面の描画
7 void disp(t_mino mino);
8
9 // ゲームオーバー画面描画
10 void disp_game_over();
11
12 #endif

```

リスト 6 main.cpp

```

1 #include <stdlib.h>
2 #include <time.h>
3 #include "DxLib/DxLib.h"
4 #include "field.hpp"
5 #include "mino.hpp"
6 #include "objects.hpp"
7 #include "define.hpp"
8 #include "score.hpp"
9 #include "game_screen.hpp"
10
11 // 初期化処理
12 void initialize(t_mino *mino){
13     // seed値の設定
14     srand((unsigned)time(NULL));
15
16     // フィールドの初期化
17     init_field();
18     // ミノを作成
19     make_t_mino(mino);

```

```

20 // スコアの初期化
21 init_score();
22 }
23
24 // プログラムは WinMain から
25 int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine,
26 int nCmdShow )
27 {
28     t_mino mino; // 操作対象のテトリミノ
29     clock_t key_time = clock(); // キー操作の時間計測用
30     clock_t drop_time = clock(); // 自動落下の時間計測用
31     double time = 0; // 現在の時間計算用
32     char key[256]; // キー取得用
33
34     ChangeWindowMode(TRUE); // 非全画面モードに
35     SetGraphMode(SCREEN_X, SCREEN_Y, 32); // 画面サイズ指定
36     SetOutApplicationLogValidFlag(FALSE); // Log.txtを生成しないように設定
37     if( DxLib.Init() == -1 ){return -1 ;} // エラーが起きたら直ちに終了
38
39 // 初期化
40 initialize(&mino);
41
42 // mainループ
43 while(ProcessMessage() == 0){
44     ClearDrawScreen(); // 裏画面の消去
45     SetDrawScreen(DX_SCREEN_BACK); // 描画先を裏画面に
46
47     // キー入力
48     GetHitKeyStateAll(key);
49     time = (double)(clock() - key_time) / CLOCKS_PER_SEC;
50     if(time >= KEY_CHECK.TIME){ // 一定の時間間隔でキーをチェック
51         key_time = clock();
52         move_block(&mino, key); // キーの方向にブロックを移動
53     }
54     // 自動落下判定
55     time = (double)(clock() - drop_time) / CLOCKS_PER_SEC;
56     if(time >= DROP_CHECK.TIME){ // 一定の時間間隔で自動落下を行う
57         drop_time = clock();
58         move_down(&mino); // 1つ下にブロックを移動
59     }
60
61     // 表示
62     // ゲームオーバーになったらゲームオーバー画面を表示しループを抜ける
63     if(!gameover){
64         disp(mino); // ミノやフィールドを表示
65     }else{
66         disp_game_over(); // ゲームオーバー画面を表示
67         ScreenFlip(); // 裏画面を表画面に
68         exit;
69     }
70
71     ScreenFlip(); // 裏画面を表画面に描画
72 }
73
74 WaitKey(); // キーが押されるまで待機
75
76 DxLib.End(); // DXライブラリ使用の終了処理
77 return 0; // ソフトの終了
78 }

```

リスト 7 mino.cpp

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 #include "objects.hpp"
5 #include "field.hpp"
6 #include "DxLib\DxLib.h"
7 #include "mino.hpp"
8 #include "game_screen.hpp"
9
10 /*

```

```

11     ブロックを指定の方向に動かす
12 */
13 void move_block(t_mino *mino, char key[256]){
14     if(key[KEY_INPUT_LEFT]){ // 左方向に移動
15         move_left(mino);
16     }
17     if(key[KEY_INPUT_RIGHT]){ // 右方向に移動
18         move_right(mino);
19     }
20     if(key[KEY_INPUT_DOWN]){ // 下方向に移動
21         move_down(mino);
22     }
23     if(key[KEY_INPUT_UP]){ // ブロックの回転
24         rotate_block(mino);
25     }
26 }
27
28 /*
29     ブロックの回転
30 */
31 void rotate_block(t_mino *mino){
32     t_mino tmp_mino;
33     int i, j;
34
35     // 仮のミノを作成
36     tmp_mino.x = mino->x;
37     tmp_mino.y = mino->y;
38     for(i = 0; i < MINO_SIZE; i++){
39         for(j = 0; j < MINO_SIZE; j++){
40             tmp_mino.mino[j][MINO_SIZE - i - 1] = mino->mino[i][j];
41         }
42     }
43
44     // 他のブロックに当たらず、範囲外でないなら90度回転
45     if(!is_field_over(tmp_mino) && !is_side_hit(tmp_mino)){
46         for(i = 0; i < MINO_SIZE; i++){
47             for(j = 0; j < MINO_SIZE; j++){
48                 mino->mino[i][j] = tmp_mino.mino[i][j];
49             }
50         }
51     }
52 }
53
54 /*
55     ミノを作成
56 */
57 void make_t_mino(t_mino *mino){
58     int type = rand() % T_MINO_TYPE; // ランダムにミノを選択
59     int x, y;
60
61     // あらかじめ決めた形から読み込み
62     for(y = 0; y < MINO_SIZE; y++){
63         for(x = 0; x < MINO_SIZE; x++){
64             mino->mino[y][x] = t_mino_buff[type][y][x];
65         }
66     }
67     mino->y = 0; // y座標の設定
68     mino->x = 3; // x座標の設定
69
70     // 作った直後にブロックがあったらゲームオーバー
71     if(is_down_hit(*mino)){
72         gameover = true;
73     }
74 }
75
76 /*
77     ミノを右に移動させる
78 */
79 void move_right(t_mino *mino){
80     mino->x++;
81     if(is_field_over(*mino) || is_side_hit(*mino)) mino->x--;
82 }
83

```

```

84
85 /*
86 ミノを左に移動させる
87 */
88 void move_left(t_mino *mino){
89     mino->x--;
90     if(is_field_over(*mino) || is_side_hit(*mino)) mino->x++;
91 }
92
93 /*
94 ミノを1つ下に移動させる
95 */
96 void move_down(t_mino *mino){
97     mino->y++;
98     if(is_down_hit(*mino) || is_field_over_y(*mino)){
99         mino->y--;
100         // 次のブロックへ
101         stack_block(*mino); // ミノを固定化
102         row_check(); // そろった列の確認
103         make_t_mino(mino); // 新しくミノを作る
104         Sleep(400);
105     }
106 }
107
108 /*
109 ミノを表示する
110 */
111 void disp_t_mino(t_mino *mino){
112     int x,y;
113     int plot_x, plot_y; // ミノの座標
114     unsigned int Color = GetColor(100,255,100); // ミノの色
115
116     // ミノを表示する
117     for(y = 0; y < MINO_SIZE;y++){
118         for(x = 0; x < MINO_SIZE;x++){
119             if(mino->mino[y][x] == EXIST){ // ミノが存在していたら描画
120                 plot_x = x + mino->x;
121                 plot_y = y + mino->y;
122                 DrawBox(plot_x*BLOCK_SIZE+OFFSET_X, plot_y*BLOCK_SIZE+OFFSET_Y,
123                     (plot_x+1)*BLOCK_SIZE+OFFSET_X, (plot_y+1)*BLOCK_SIZE+OFFSET_Y, Color,
124                     true);
125             }
126         }
127     }

```

リスト 8 mino.hpp

```

1 #ifndef MINO_HPP
2 #define MINO_HPP
3
4 #include "objects.hpp"
5
6 // 指定方向にミノを移動
7 void move_block(t_mino *mino, char key[256]);
8
9 // ミノを回転する
10 void rotate_block(t_mino *mino);
11
12 // ミノを新しく作成する
13 void make_t_mino(t_mino *mino);
14
15 // ミノを右方向に移動する
16 void move_right(t_mino *mino);
17
18 // ミノを左方向に移動する
19 void move_left(t_mino *mino);
20
21 // ミノを下に移動する
22 void move_down(t_mino *mino);
23
24 // ミノを描画する

```

```

25 void disp_t_mino(t_mino *mino);
26
27 #endif // CONTROL_MINO_HPP

```

リスト 9 objects.cpp

```

1 #include "objects.hpp"
2 #include "define.hpp"
3 /*
4  グローバル変数を定義
5 */
6
7 // フィールドオブジェクト
8 int field[FIELD_Y][FIELD_X];
9 // スコア
10 int score = 0;
11 // ゲームオーバーフラグ
12 bool gameover = false;
13 // ミノの種類を保持するオブジェクト
14 int t_mino_buff[T_MINO_TYPE][MINO_SIZE][MINO_SIZE] = {
15     {{0,0,0,0},
16      {1,1,1,1},
17      {0,0,0,0},
18      {0,0,0,0}},
19     {{0,0,0,0},
20      {0,1,1,0},
21      {0,1,1,0},
22      {0,0,0,0}},
23     {{0,0,0,0},
24      {0,1,1,0},
25      {1,1,0,0},
26      {0,0,0,0}},
27     {{0,0,0,0},
28      {1,1,0,0},
29      {0,1,1,0},
30      {0,0,0,0}},
31     {{0,0,0,0},
32      {0,1,0,0},
33      {0,1,1,1},
34      {0,0,0,0}},
35     {{0,0,0,0},
36      {0,0,1,0},
37      {1,1,1,0},
38      {0,0,0,0}},
39     {{0,0,0,0},
40      {0,1,0,0},
41      {1,1,1,0},
42      {0,0,0,0}},
43     {{0,0,0,0},
44      {0,1,0,0},
45      {1,1,1,0},
46      {0,0,0,0}}};
47

```

リスト 10 objects.hpp

```

1 #ifndef OBJECTS_HPP
2 #define OBJECTS_HPP
3
4 #include "define.hpp"
5
6 // テトリミノの構造体
7 typedef struct{
8     int mino[MINO_SIZE][MINO_SIZE]; // ミノ本体
9     int x,y; // テトリミノの座標
10 } t_mino;
11
12 // フィールド
13 extern int field[FIELD_Y][FIELD_X];
14 // スコア

```

```

15 extern int score;
16 // ゲームオーバーフラグ
17 extern bool gameover;
18 // テトリミノの種類
19 extern int t_minos_buff[T_MINO_TYPE][MINO_SIZE][MINO_SIZE];
20
21 #endif

```

リスト 11 score.cpp

```

1 #include "score.hpp"
2 #include "objects.hpp"
3 #include "Dxlib/DxLib.h"
4
5 // スコアの初期化
6 void init_score(){
7     score = 0;
8 }
9
10 // スコアの加算
11 void add_score(int lines){
12     score += ADD_SCORE * (1 + (lines - 1) * SCORE_BIAS);
13 }
14
15 // スコアの表示
16 void disp_score(){
17     int color = GetColor(100,255,255);
18     DrawFormatString(0,0,color,_T("SCORE : %d"),score);
19 }

```

リスト 12 score.hpp

```

1 #ifndef SCORE_HPP
2 #define SCORE_HPP
3
4 // スコアの初期化
5 void init_score();
6
7 // スコアの加算
8 void add_score(int lines);
9
10 // スコアを表示する
11 void disp_score();
12
13 #endif

```