平成 30 年度ソフトウェア設計
テトリスの作成

2020 年 2 月 13 日

班名：10 班
篠田拓樹 (班長) 伊崎龍河 (副班長) 秋元駿
江尻姫花 尾畑賢太 本田歩 宮本雅也

目次

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，画面を視認できること
作るもの　テトリス
機能的要求　テトリスとして動作すること
動作環境・OS　Windows10
使用言語　C 言語/C++
外部ライブラリ　DxLibrary(GUI 表示用)
インターフェース　標準キーボード，ディスプレイ
ネットワーク　なし

2　用語定義

テトリスにおける用語の定義を以下に示す．

- テトリミノ プレーヤーが操作する対象物．7 種類の形状がある．
- フィールド テトリミノの可動出来る範囲．ブロックの設置を行う場所．
- ブロック フィールドに固定されたテトリミノのこと．

3　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．

4　計画表

以下に作成した計画表を示す．上の表は計画段階での表であり，下の表は実際に行った作業の日程である．表中の黒い部分が日程での作業である．

	10/28	11/7	11/11	11/18	11/25	11/29	12/2	12/9	12/16	12/23	1/6	1/20	1/27	1/31	2/6	2/10
ルール決定	■															
物の抽出	■	■														
物の振る舞い	■	■														
データ構造		■	■													
関数仕様		■	■	■												
テスト計画				■	■	■	■									
コード作成				■	■	■	■	■								
テスト									■	■						
考察											■	■				
予備日													■	■	■	■

	10/28	11/7	11/11	11/18	11/25	11/29	12/2	12/9	12/16	12/23	1/6	1/20	1/27	1/31	2/6	2/10
ルール決定	■															
物の抽出	■	■														
物の振る舞い	■	■														
データ構造		■	■													
関数仕様		■	■	■												
テスト計画				■	■	■	■									
コード作成1				■	■	■	■	■								
テスト1									■							
仕様見直し										■						
コード作成2										■	■					
テスト2												■	■			
考察													■	■		

5　役割分担

- 物の抽出・振る舞い抽出：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

6　テスト計画

6.1　要件仕様と設計仕様書の相互確認

要件仕様と設計仕様において誤りや抜けがないかどうかについて確認し，設計仕様についても細かく設計できていることを確認した．

6.2　関数ごとの単体テスト

関数毎の機能が設計仕様を満たしていることを確認した．

6.3　全体のシステムテスト

以下のテスト項目に従ってシステムの確認を行った．1 回目のテストで不具合を発見したため，仕様の確認を行いその後 2 回目のテストを行った．2 回目のテストでは，全ての項目が正常な動作であった．

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

7　物の抽出とふるまい

7.1　物の抽出

- テトリミノ
- テトリミノのバッファ
- フィールド
- スコア

7.2　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
- テトリミノのバッファ：テトリミノ生成時の高速化のため
- 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

8　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)
  }
- テトリミノのバッファ

  int t_mino_buf[T_MINO_TYPE][MINO_SIZE][MINO_SIZE];
- フィールド

  int field[FIELD_Y][FIELD_X]; // FIELD_X=10,FIELD_Y=20
- スコア

  int score;

9　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かを判定

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かを判定

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かを判定

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る
内容	テトリミノが x 軸でブロックと被っているか判定

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る
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

10　状態遷移表

状態遷移表を表 1 に示す．状態遷移表を作成することでシステムにおいてやってはならないことを可視化することが出来た．また，今回作成することが出来なかったが状態遷移図を作成することによってシステムの状態の流れを可視化することが出来，より具体的に画面遷移等の状態の変化を見ることが出来るようになる．

表 1　状態遷移表

状態番号	状態名	上下左右ボタン	テトリミノが ブロックに触れた	テトリミノが 下の壁に触れた	テトリミノが 下左右の壁を超えた	テトリミノが ブロックに被った	行が揃っていた	1 秒経過	なし
S0	テトリミノ操作可能	S12	S2	S2				S10	S0
S1	テトリミノの生成		S13						S0
S2	テトリミノを固定								S3
S3	行が何列揃っているか確認						S4		
S4	行の削除								S5
S5	行を詰める								S6
S6	スコアの倍率計算								S7
S7	スコアの加算								S1
S8	テトリミノを右に移動								S0
S9	テトリミノを左に移動								S0
S10	テトリミノを下に移動								S0
S11	テトリミノを回転								S0
S12	範囲外判定				S0	S0			{S8,S9,S10,S11}
S13	ゲームオーバー (終了)								S13

11　操作方法

11.1　ゲーム開始操作

実行ファイル (main.exe) をダウンロードする．実行ファイルを実行する．ゲームが開始されたらゲーム内操作に従って操作を行う．

11.2　ゲーム終了操作

GameOver 画面が表示されたらウィンドウの×ボタンを押してウィンドウを閉じる．

11.3　ゲーム内操作方法

操作方法を以下に示す．

右矢印 テトリミノを右に移動する
左矢印 テトリミノを左に移動する
上矢印 テトリミノを時計回りに回転する
下矢印 テトリミノを下に移動する

12　実行画面

12.1　ゲーム画面

初期画面を図 1 に示す．ゲームを開始すると同時にテトリミノが落下を始める．

12.2　テトリミノの落下

テトリミノの落下を図 2 に示す．テトリミノは，毎秒 1 ブロックの速さで落下する．

12.3　テトリミノの固定

テトリミノの固定を図 3 に示す．テトリミノは，他のブロックや下の範囲に触れると固定される．

12.4　テトリミノの新規作成

テトリミノの新規作成を図 4 に示す．テトリミノが固定されると新しく操作可能なブロックが作成される．

12.5　1 行消去のスコア加算

1 行消去したときのスコア加算を図 5,6 に示す．1 行消去されるとスコアが 100 加算される．

12.6　2 行消去のスコア加算

2 行消去したときのスコア加算を図 7,8 に示す．2 行消去されるとスコアが 100 加算される．

12.7　3 行消去のスコア加算

3 行消去したときのスコア加算を図 9,10 に示す．3 行消去されるとスコアが 100 加算される．

12.8　4 行消去のスコア加算

4 行消去したときのスコア加算を図 11,12 に示す．4 行消去されるとスコアが 100 加算される．

12.9　ゲームオーバー画面

ゲームオーバー画面を図 13 に示す．ブロックが新しく出たときに操作不可能だった場合固定される．

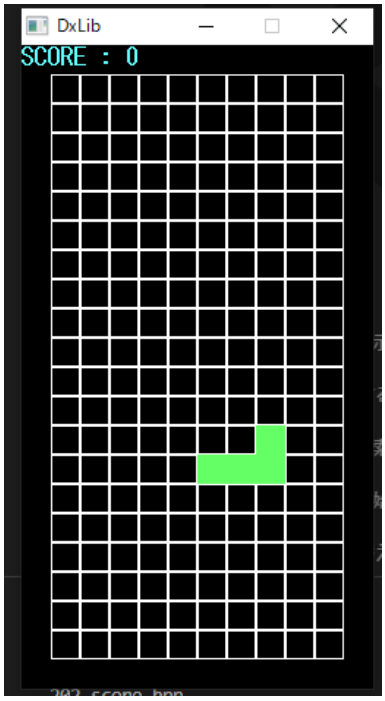

図１　ゲーム画面

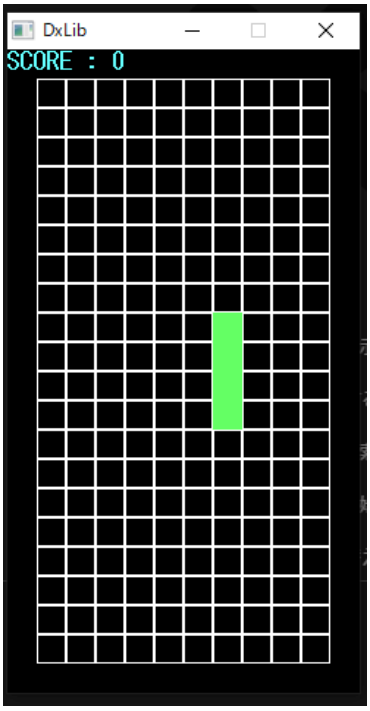

図２　テトリミノの落下

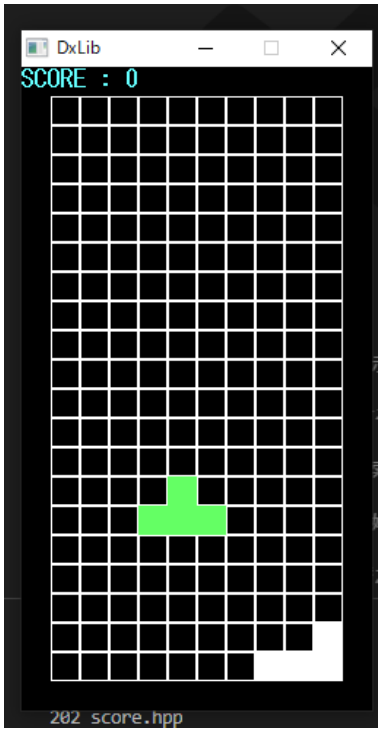

図３　テトリミノの固定

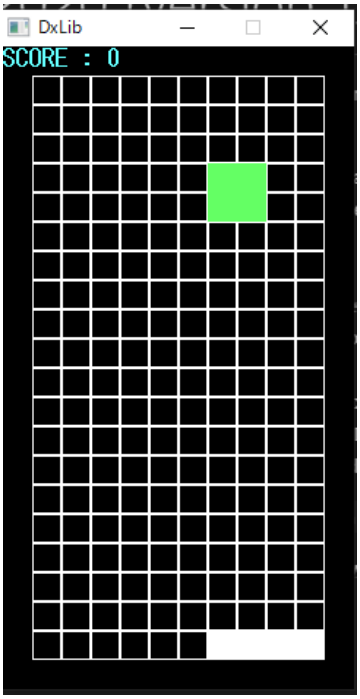

図４　テトリミノの新規作成

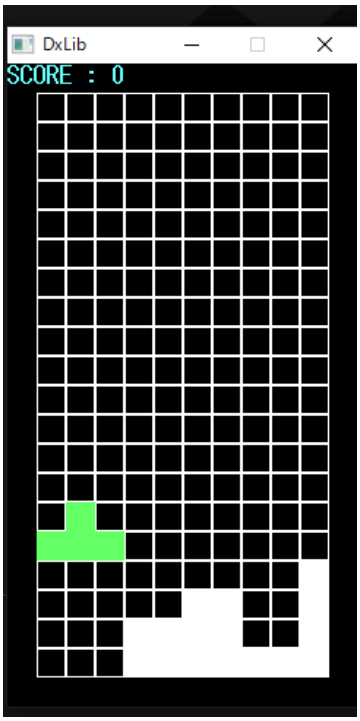

図 5　1 行消去前の画面

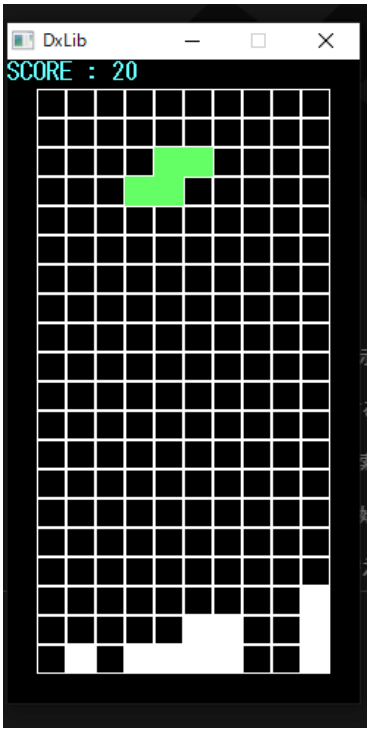

図 6　1 行消去後の画面

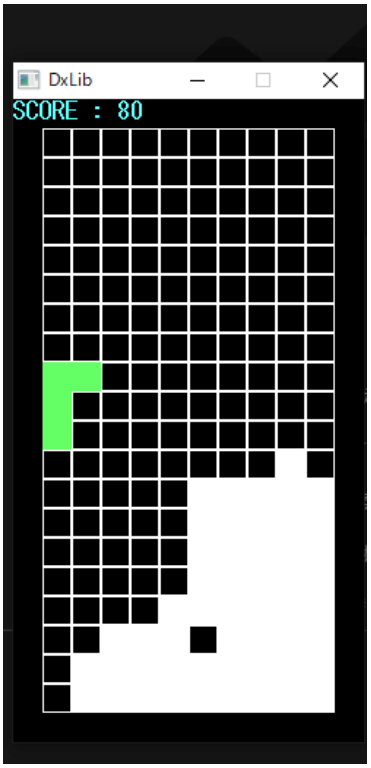

図 7　2 行消去前の画面

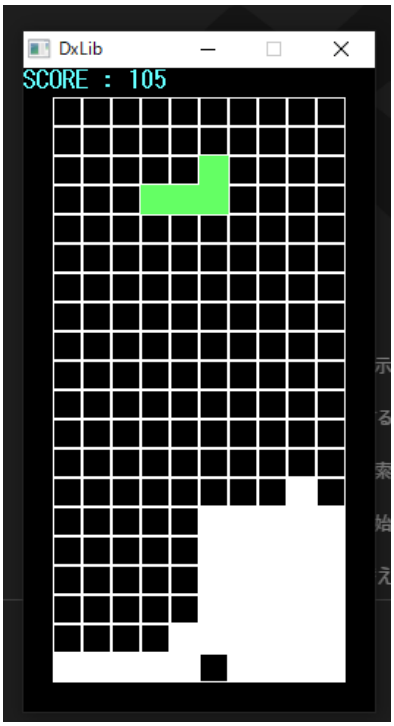

図 8　2 行消去後の画面

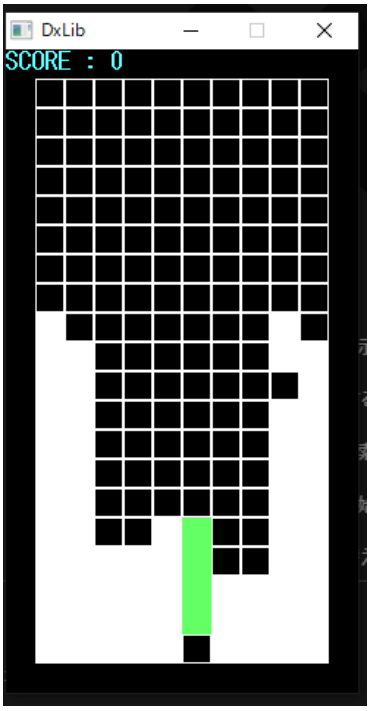

図 9　3 行消去前の画面

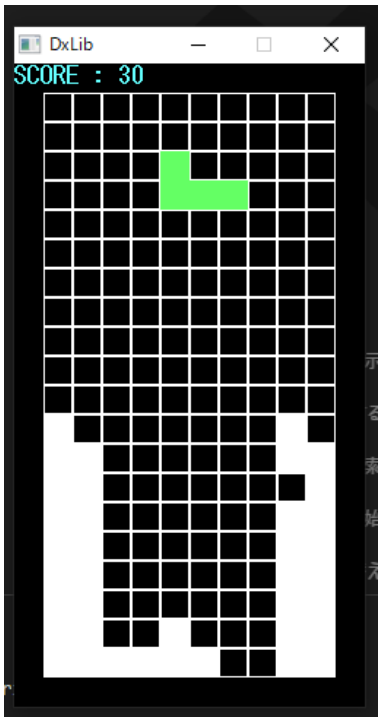

図 10　3 行消去後の画面

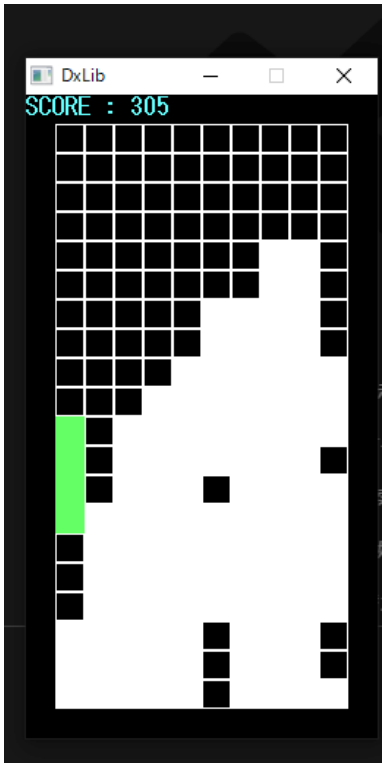

図 11　4 行消去前の画面

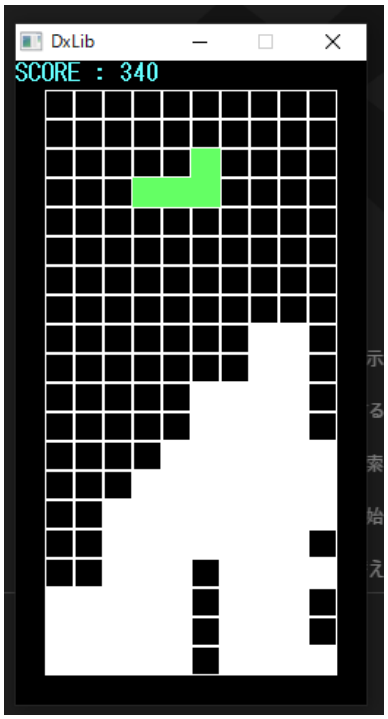

図 12　4 行消去後の画面

図 13　ゲームオーバー画面

13　作成後の批評・考察

13.1　プレイした人の感想

- ブロックが 1 色じゃないほうが楽しい.
- 次に落ちてくるブロックが分かるほうが使いやすい.
- ホールド機能が欲しい.
- スコアによってスピードが変化するほうが良い.

13.2　改善点

- ブロックを多色にする.
- 次に落ちてくるブロックの表示を行う.
- ホールド機能を実装する.
- 速度の変化を実装する.

13.3　設計・作成に関する反省

今回は，ウォーターフォール設計を用いてソフトウェアを作成した．1 つ 1 つの工程で成果が文章化出来た点では，資料作成や日程の管理がやりやすかった．しかし，バグを発見した際に仕様まで戻る必要があったため予想以上の時間がかかった．また，最終形にならないとシステムの形が見えてこないので仕様の変更 (ユーザーの要求) には対応しづらいと思った.

14　プログラムリスト

リスト 1　define.hpp

```
#ifndef    DEFINE_HPP
#define    DEFINE_HPP

// スクリーンの大きさ
#define SCREEN_X 240
#define SCREEN_Y 440

// 表示のオフセット
#define OFFSET_X 20
#define OFFSET_Y 20

// テトリミノ種類
#define T_MINO_TYPE 7
#define TYPE_1 0
#define TYPE_2 1
#define TYPE_3 2
#define TYPE_4 3
#define TYPE_5 4
#define TYPE_6 5
#define TYPE_7 6

// テトリミノのサイズ
#define MINO_SIZE 4

// フィールドの大きさ
#define FIELD_X 10
#define FIELD_Y 20

// ブロックのサイズ
#define BLOCK_SIZE 20

// 移動方向
#define RIGHT 1
#define LEFT  2
#define DOWN  3
#define NONE  0

// ブロックの存在を表す
#define EMPTY 0
#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;
                    // 範囲を超えていないか確認
41             }
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;
                     // 範囲を超えていないか
59              }
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;
                     // フィールドと被っていたら
77              }
78          }
79      }
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

```
#ifndef    FIELD_HPP
#define    FILED_HPP

#include "objects.hpp"

// フィールド初期化
void init_field();

// ミノがフィールド内か確認
bool is_field_over(t_mino mino);

// ミノが y 軸でフィールド内か確認
bool is_field_over_y(t_mino mino);

// ミノが x 軸でフィールド内か確認
bool is_field_over_x(t_mino mino);

// x 軸でミノがフィールドのブロックと被ってないか確認
bool is_side_hit(t_mino mino);

// y 軸でミノがフィールドのブロックと被ってないか確認
bool is_down_hit(t_mino mino);

// 列がそろっているのかの確認
void row_check();

// 列を 1 段下に詰める
void pack_block(int row);

// 指定された列を消去する
void row_clear(int row);

// ブロックを固定化する
void stack_block(t_mino mino);

// フィールドを表示する
void disp_field();

#endif    // DISPLAY_HPP
```

リスト 4　game_screen.cpp

```
#include "objects.hpp"
#include "score.hpp"
#include "mino.hpp"
#include "field.hpp"
#include "DxLib/DxLib.h"

// ゲーム画面描画
void disp(t_mino mino){
    // フィールドの描画
    disp_field();
    // ミノの描画
    disp_t_mino(&mino);
    // スコアの描画
    disp_score();
}

// ゲームオーバー画面
void disp_game_over(){
    TCHAR gameover_str[] = _T("GAME OVER"); // ゲームオーバーの文字列
    int Green = GetColor( 0, 255, 0 ); // 文字色

    DrawString( SCREEN_X/2-40, SCREEN_Y/2, gameover_str, Green); // ゲームオーバー文字の表示
}
```

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, int nCmdShow )
26 {
27     t_mino mino; // 操作対象のテトリミノ
28     clock_t key_time = clock(); // キー操作の時間計測用
29     clock_t drop_time = clock(); // 自動落下の時間計測用
30     double time = 0; // 現在の時間計算用
31     char key[256]; // キー取得用
32
33     ChangeWindowMode(TRUE); // 非全画面モードに
34     SetGraphMode(SCREEN_X, SCREEN_Y, 32); // 画面サイズ指定
35     SetOutApplicationLogValidFlag(FALSE); // Log.
         txtを生成しないように設定
36     if( DxLib_Init() == -1 ){return -1 ;} // エラーが起きたら直ちに終了
37
38     // 初期化
39     initialize(&mino);
40
41     // mainループ
42     while(ProcessMessage() == 0){
```

```
43          ClearDrawScreen(); // 裏画面の消去
44          SetDrawScreen(DX_SCREEN_BACK); // 描画先を裏画面に
45
46          // キー入力
47          GetHitKeyStateAll(key);
48          time = (double)(clock() - key_time) / CLOCKS_PER_SEC;
49          if(time >= KEY_CHECK_TIME){ // 一定の時間間隔でキーをチェック
50              key_time = clock();
51              move_block(&mino, key); // キーの方向にブロックを移動
52          }
53          // 自動落下判定
54          time = (double)(clock() - drop_time) / CLOCKS_PER_SEC;
55          if(time >= DROP_CHECK_TIME){ // 一定の時間間隔で自動落下を行う
56              drop_time = clock();
57              move_down(&mino); // 1つ下にブロックを移動
58          }
59
60          // 表示
61          // 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
62          if(!gameover){
63              disp(mino); // ミノやフィールドを表示
64          }else{
65              disp_game_over(); // ゲームオーバー画面を表示
66              ScreenFlip(); // 裏画面を表画面に
67              exit;
68          }
69
70          ScreenFlip(); // 裏画面を表画面に描画
71      }
72
73      WaitKey(); // キーが押されるまで待機
74
75      DxLib_End() ; // DXライブラリ使用の終了処理
76      return 0 ; // ソフトの終了
77  }
```

リスト 7　mino.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include "objects.hpp"
#include "field.hpp"
#include "DxLib\DxLib.h"
#include "mino.hpp"
#include "game_screen.hpp"

/*
    ブロックを指定の方向に動かす
*/
void move_block(t_mino *mino,char key[256]){
    if(key[KEY_INPUT_LEFT]){ // 左方向に移動
        move_left(mino);
    }
    if(key[KEY_INPUT_RIGHT]){ // 右方向に移動
        move_right(mino);
    }
    if(key[KEY_INPUT_DOWN]){ // 下方向に移動
        move_down(mino);
    }
    if(key[KEY_INPUT_UP]){ // ブロックの回転
        rotate_block(mino);
    }
}

/*
    ブロックの回転
*/
void rotate_block(t_mino *mino){
    t_mino tmp_mino;
    int i,j;

    // 仮のミノを作成
    tmp_mino.x = mino->x;
    tmp_mino.y = mino->y;
    for(i = 0;i < MINO_SIZE;i++){
        for(j = 0;j < MINO_SIZE;j++){
            tmp_mino.mino[j][MINO_SIZE - i - 1] = mino->mino[i][j];
        }
    }

    // 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  /*
56      ミノを作成
57  */
58  void make_t_mino(t_mino *mino){
59      int type = rand() % T_MINO_TYPE; // ランダムにミノを選択
60      int x,y;
61
62      // あらかじめ決めた形から読み込み
63      for(y = 0;y < MINO_SIZE;y++){
64          for(x = 0;x < MINO_SIZE;x++){
65              mino->mino[y][x] = t_mino_buff[type][y][x];
66          }
67      }
68      mino->y = 0; // y座標の設定
69      mino->x = 3; // x座標の設定
70
71      // 作った直後にブロックがあったらゲームオーバー
72      if(is_down_hit(*mino)){
73          gameover = true;
74      }
75  }
76
77  /*
78      ミノを右に移動させる
79  */
80  void move_right(t_mino *mino){
81      mino->x++;
82      if(is_field_over(*mino) || is_side_hit(*mino)) mino->x--;
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,true);
124             }
125         }
126     }
127 }
```

リスト 8　mino.hpp

```
#ifndef    MINO_HPP
#define    MINO_HPP

#include "objects.hpp"

// 指定方向にミノを移動
void move_block(t_mino *mino,char key[256]);

// ミノを回転する
void rotate_block(t_mino *mino);

// ミノを新しく作成する
void make_t_mino(t_mino *mino);

// ミノを右方向に移動する
void move_right(t_mino *mino);

// ミノを左方向に移動する
void move_left(t_mino *mino);

// ミノを下に移動する
void move_down(t_mino *mino);

// ミノを描画する
void disp_t_mino(t_mino *mino);

#endif     // CONTROL_MINO_HPP
```

リスト 9　objects.cpp

```
#include "objects.hpp"
#include "define.hpp"
/*
    グローバル変数を定義
*/

// フィールドオブジェクト
int field[FIELD_Y][FIELD_X];
// スコア
int score = 0;
// ゲームオーバーフラグ
bool gameover = false;
// ミノの種類を保持するオブジェクト
int t_mino_buff[T_MINO_TYPE][MINO_SIZE][MINO_SIZE] = {{{0,0,0,0},
                                                       {1,1,1,1},
                                                       {0,0,0,0},
                                                       {0,0,0,0}},

                                                      {{0,0,0,0},
                                                       {0,1,1,0},
                                                       {0,1,1,0},
                                                       {0,0,0,0}},

                                                      {{0,0,0,0},
                                                       {0,1,1,0},
                                                       {1,1,0,0},
                                                       {0,0,0,0}},

                                                      {{0,0,0,0},
                                                       {1,1,0,0},
                                                       {0,1,1,0},
                                                       {0,0,0,0}},

                                                      {{0,0,0,0},
                                                       {0,1,0,0},
                                                       {0,1,1,1},
                                                       {0,0,0,0}},

                                                      {{0,0,0,0},
                                                       {0,0,1,0},
                                                       {1,1,1,0},
                                                       {0,0,0,0}},

                                                      {{0,0,0,0},
```

```
45                                                    {0,1,0,0},
46                                                    {1,1,1,0},
47                                                    {0,0,0,0}}};
```

リスト 10　objects.hpp

```
#ifndef    OBJECTS_HPP
#define    OBJECTS_HPP

#include "define.hpp"

// テトリミノの構造体
typedef struct{
    int mino[MINO_SIZE][MINO_SIZE]; // ミノ本体
    int x,y; // テトリミノの座標
} t_mino;

// フィールド
extern int field[FIELD_Y][FIELD_X];
// スコア
extern int score;
// ゲームオーバーフラグ
extern bool gameover;
// テトリミノの種類
extern int t_mino_buff[T_MINO_TYPE][MINO_SIZE][MINO_SIZE];

#endif
```

リスト 11　score.cpp

```
#include "score.hpp"
#include "objects.hpp"
#include "Dxlib/DxLib.h"

// スコアの初期化
void init_score(){
    score = 0;
}

// スコアの加算
void add_score(int lines){
    score += ADD_SCORE * (1 + (lines-1) * SCORE_BIAS);
}

// スコアの表示
void disp_score(){
    int color = GetColor(100,255,255);
    DrawFormatString(0,0,color,_T("SCORE : %d"),score);
}
```

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
```