テトリスを作ろう!

リアルタイムゲームプログラム入門 Ver.2

2014 年 5 月 28 日

日本工業大学創造システム工学科
ロボット創造コースロボットビジョン研究室　田村仁

内容

1. はじめに

1.1 本稿の対象について

本稿では，リアルタイムプログラミングの基礎として，リアルタイムパズルゲームの代表格であるテトリスを，C 言語を用いて作成する方法について解説する．

対象とする読者は，C 言語の初歩を修得済みの大学生である．創造システム工学科では「C 言語入門」の単位取得者(およびそれと同等の能力を有する者)であり，具体的にいえば，

① C 言語の 10 行程度の初歩的なサンプルプログラムの全体構成が分かる．
② #include や#define が使える
③ ++,+=,==,!=,||, などの演算子を理解している．
④ if 文が分かる．
⑤ for 文が分かる．
⑥ switch 文が分かる．
⑦ 関数が使える．(作れる)

といった能力を有する人を対象とする．

これらがうろ覚えな人は，あらかじめ C 言語入門の教科書など C 言語の入門書を復習しておくこと．また，とりあえず本稿を読み進めて，これらがよくわかっていなかったことに気づいた場合でも，すぐ入門書に戻って自分で確認すること．

C 言語の基礎はわかったが，実際に何か実用的なプログラムを作成しようと思ったときに，どのように開発していくのかが分からない人が多い．本稿はそうした人のために，どのように考えて一からプログラムを作り上げていくのか，全手順を説明している．

本稿をよく読んで，各自自習して見よう．想定している時間数は，週 1 コマとして 6 週間のコース，丸一日集中すれば 2〜3 日程度の時間がかかる．ただし，不慣れな人はもう少し時間がかかるだろう．また，途中脇道にそれて色々試してみるには，さらに時間が必要である．自分の PC で自習できるようにして，なるべく色々試してほしい．

1.2 テトリスについて

本稿で作成するテトリスは，1980 年代に世界的に流行したリアルタイムパズルゲームであり，旧ソ連のアレクセイ・パジトノフらが開発したとされる．

図 のように画面上部から数種類のブロックが落ちてくるので，それを操作して隙間ができないように積み上げるパズルである．横一行がきれいに埋まるとその行は消滅するが，隙間がある行はそのままブロックが積み上がる．最終的に天井まで積みあがればゲームオーバーとなる．多くのブロックをきれいに配置し，消滅させたライン数が多いほど高得点になる．

テトリスはいわゆる「落ちものゲーム」の元祖であり，比較的単純なプログラムで作成可能である．実際 C 言語で記述すれば，プログラム部分で 400 行程度，データ部分を含めても 600 行程度で完成する．

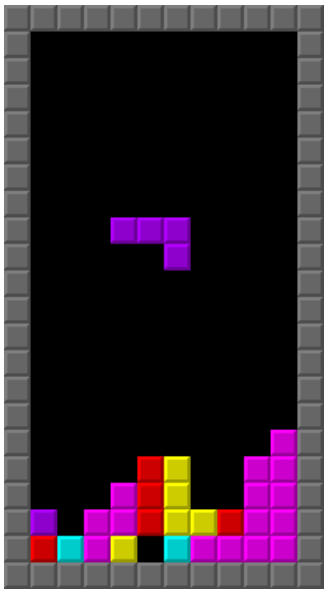


図 1 テトリスゲーム画面(引用元:Wikipedia)

単純なゲームだが，これを実現するためには，ゲームの任意の瞬間にキーボードで押されているキーをリアルタイムで取得し，タイマーを使用して一定時間でブロックを落下させる表示を行い，ブロックの消滅判定手続きや得点表示などを，すべて「同時に行っているように(見える)」プログラムを作成する必要がある．

テトリスを実現できれば，こうした入力装置や出力装置をリアルタイムに操作するプログラム，つまりロボット等の組み込みプログラムを作成する練習となる．

2. プログラムの開発環境

本稿では，プログラムの開発環境として Ubuntu Linux に最初から導入されている cc コマンドと，標準エディタの gedit を用いる．

4-108 教室の PC は，Windows であるが，Ubuntu Linux も利用できるように VMware player が導入されている．VMware player は VMware 社が提供している仮想 PC ソフトウェアである．少々ややこしいが，PC の中にもう一台別の PC を仮想的に用意するソフトウェアである．VMware player は個人利用・教育目的利用において無料で利用できる．

2.1 Ubuntu Linux の起動

4-108 室では，アイコンなどから VMware Player を起動すると，図 2 のように初期画面から Ubuntu を選択して，そのまま Ubuntu Linux を起動できる．

図 2 VMware player のアイコンと初期画面

起動すると，まず仮想 PC が通常の PC のように電源 ON 後の起動画面を表示して，そのまま Ubuntu が起動する(図 3)．4-108 室の Ubuntu ではユーザ名とパスワードとも「iseuser」でログイン可能である．

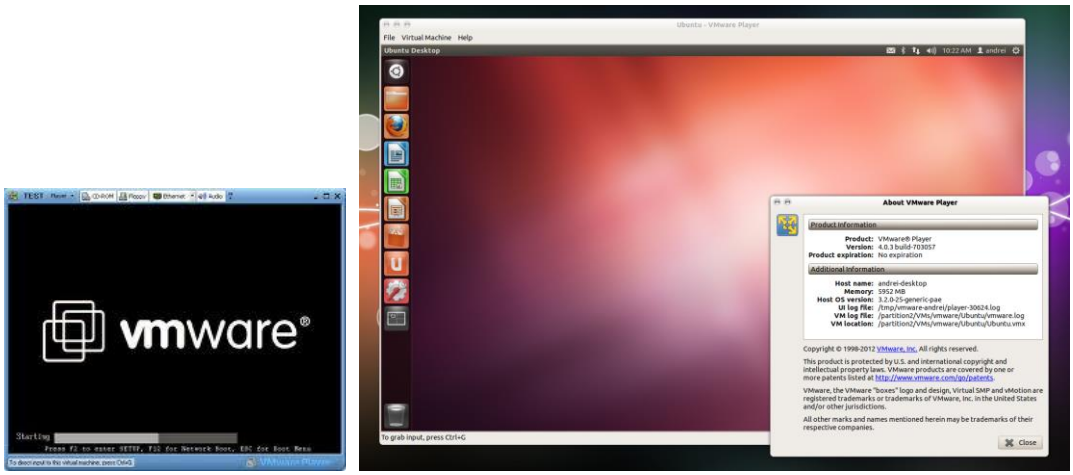


図 3 仮想 PC 起動画面と起動した Ubuntu のデスクトップ画面

2.2 端末の起動

本稿で利用するプログラミングでは，図 4 のように Ubuntu のデスクトップの左上にある Dash メニューをクリックして「terminal」と入力する(最初の数文字だけでもよい)．すると，すぐにコマンドが検索されて端末のアイコンが表示される．そのまま端末のアイコンをダブルクリックして起動する．

図 4　Dash メニューと端末の起動

この端末で，cc コマンドによりプログラムのコンパイルと実行ができる．

2.3 gedit の起動

プログラムの入力には gedit を起動する．同じく左上の Dash メニューから起動する．か，今起動した端末に「gedit&(エンターキー)」と入力してもよい．

起動した gedit(図 5)は，そのままプログラムを入力して保存することができる．

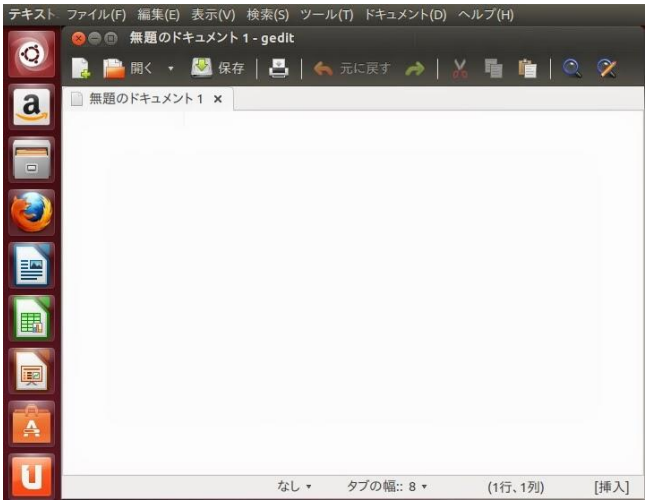


図 5 gedit(テキストエディター)

保存時には，図 6 のフォルダの中に保存(F)を，自分の USB メモリに指定することを忘れないこと．また名前は末尾の拡張子を必ず「.c」とすること．自分の USB メモリは，VMware のウィンドウをアクティブにした状態で，USB メモリを PC に挿すだけで自動的に認識され，ウィンドウが開き，左端のランチャーの中にアイコンが表示される．

図 6 プログラムの保存

2.4 保存したプログラムのコンパイルと実行

保存したプログラムは，端末の gcc コマンドでコンパイルして実行できる．コンパイルするためには，まず端末の中で，保存したプログラムファイルの場所へ移動する．まず端末で「cd　」と(空白も)入力した後，図 7 のように USB フォルダを端末にドラッグして入力する．

例えばこれで

```
cd '/media/iseuser/trancend'
```

のように，移動先をコマンド入力できる．cd は change directory の略で，現在開いてい

るフォルダ(カレントディレクトリ)を移動するコマンドである．最後にエンターキーを押してコマンドを実行する．

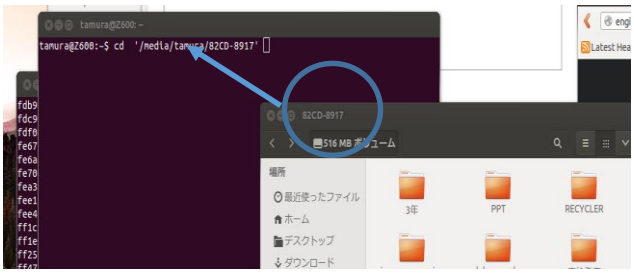


図 7 cd コマンドでフォルダへ移動

次に，ここで先ほど保存したプログラムをコンパイルする．「cc」は C Compiler(C 言語コンパイラ)の略である．「-o」(マイナス　オー)の前後に空白がある点，その次にある「~」(チルダ)はキーボードの SHIFT キーを押しながら「^」で入力する点に注意．

```
cc hello.c  -o ~/a.out
```

エラーがなければ(何も表示されなければ)，「a.out」という名前のファイルが，ホームフォルダに作成されている．これが実行可能ファイルである．もしも，コンパイル中にエラーがあると，エラーメッセージが端末に表示されて，このファイルは作成されない．メッセージの英語をよく読んで，エラーが出ている行番号を確認して，gedit で再度修正して保存し直す．

うまく a.out が作成されたら，それを実行してみよう．a.out の実行は，端末上で

```
~/a.out
```

として実行することができる．

以後，作成したプログラムは毎回この方法でコンパイルして実行すること．ちなみに，端末では，前に一度入力したことのあるコマンドは，上矢印↑キーを何回か押して選択し，エンターキーを押すだけで，再入力できるようになっている．

[課題1]

(1) gedit に，次のコード 1 を入力して保存し，コンパイルし，実行してみよ．

```
#include <stdio.h>
int main(void)
{
  printf("Hello C¥n");
}
```

コード 1 練習プログラム

(2) 本稿末尾にある付録の，テトリスプログラムの完成版を動かして，テトリスの動作を再確認してみよ．完成版テトリスプログラムは田村研 Web ページからもダウンロードできる．ただし必要以上は遊ばないこと．本稿の目的はあくまでプログラムを作成することである．
もしも，この完成版プログラムが理解できるならば，これ以上本稿を読み進める必要はない．完成版プログ

ラムを自由に改造して，例えば，通信対戦化やコラムスなどの別のパズルゲームに変更するなど，自由にプログラムを作成してよい．

2.5 (参考)自分の PC で作業するには

VMware player と Ubuntu Linux はともにフリーソフトであるので，自宅の PC やノート PC にインストールすることができる．可能ならば自宅などで自習して，作業を進めることを強く勧める．

VMware Player のダウンロードとインストールなどはインターネットで解説するサイトがたくさんある．例えば次のようなサイトを参考に自分で作業してみよ．

(1) VMWare Player のダウンロードとインストール方法
http://www.websec-room.com/2013/03/26/767

また，同様にその VMware に Ubuntu を入れる方法も解説サイトがたくさん存在している．

(2) 自分の PC の VMware player に Ubuntu を入れる方法
http://kingpcfx.seesaa.net/article/268322741.html

また，2014 年 5 月の段階で，Ubuntu の最新版は 14.04 となっている．上記紹介したサイトでは 12.04 をダウンロードするようになっているが，ダウンロード先のフォルダの親フォルダを見ると，それ以外のバージョンも用意されており，最新版をダウンロードすることもできる．

3. プログラム内容

ここで完成版プログラムのことは一旦忘れて，テトリスを最初の一歩から順に作成していこう．これは，何か実用的なプログラムを作成するための手順を練習することになる．今回は STEP1 から STEP14 までの 14 段階に分けた手順で進める．

モダンな開発環境では，ゲーム画面にグラフィック機能を用いることになるが，そのためにはウィンドウの設定や初期化など，プログラムの本筋とは関係ない記述が増える．

そこで，今回は文字表示を使ってテトリスを表示してみる．そのためには，まず好きな文字を，画面の任意の場所に表示する機能が必要なことに気付くはずだ．

では，そこから説明しよう．

3.1 STEP1 端末の任意の場所に文字に任意の色で表示する方法

あなたが，好きな文字を，好きな位置に表示する方法を知らなければ，ネットや文献でその方法を調べることから始めなければならない．実際に調べて様々な方法を試行錯誤することも勉強ではあるが，ここでは時間がないので，数ある方法の中からお勧めの方法を紹介する．

ここでは Linux などのコンピュータ端末機能にあるエスケープシーケンスを用いて，比較的簡便に，任意の場所に文字を表示したり，文字色を変更したりすることを行う．

エスケープシーケンスは，特殊な一連の記号を端末に表示すると，その記号にあらかじめ割り当てられた機能，例えばカーソル位置を変更したり，文字色を変更したり，様々な動作を行うものである．

エスケープシーケンスのうち例えば文字色に関わるものは表 1 などが用意されている．

表 1 端末制御用エスケープシーケンス

¥033[2J	画面をクリアします。		¥033[m	既定の文字色に戻します。	
¥033[30m	文字色を黒にします。		¥033[40m	¥033[30m	の反転表示。
¥033[31m	〃	赤 〃	¥033[41m	¥033[31m	〃
¥033[32m	〃	緑 〃	¥033[42m	¥033[32m	〃
¥033[33m	〃	黄色 〃	¥033[43m	¥033[33m	〃
¥033[34m	〃	青 〃	¥033[44m	¥033[34m	〃
¥033[35m	〃	紫 〃	¥033[45m	¥033[35m	〃
¥033[36m	〃	水色 〃	¥033[46m	¥033[36m	〃
¥033[37m	〃	白 〃	¥033[47m	¥033[37m	〃

エスケープシーケンスを用いたサンプルプログラムはコード 2 のようなものである．ここで，#define は，マクロの定義，定数の定義に使用している．※注意：コード 2 を，こ

のテキストからコピーペーストする場合，「¥033」という箇所がいくつか存在するが，それをすべて「\033」(「バックスラッシュ」，キーボード下の方のキー)に置き換えること．

```
#include <stdio.h>
#define clearScreen()  printf("¥033[2J")
#define setPosition(x,y) printf("¥033[%d;%dH",(y),(x))
#define setCharColor(n) printf("¥033[3%dm",(n))
#define setBackColor(n) printf("¥033[4%dm",(n))
#define BLACK 0
#define RED 1
#define GREEN 2
#define YELLOW 3
#define BLUE 4
#define MAGENTA 5
#define CYAN 6
#define WHITE 7
#define DEFAULT 9

int main(int argc, char *argv[])
{
  clearScreen(); //画面クリア
  setBackColor(BLUE); //文字色設定
  setCharColor(YELLOW); //文字の背景色設定
  setPosition(5,10); //位置指定
  printf("hello!"); fflush(stdout); //表示※
  setBackColor(DEFAULT);//文字背景色を元に戻す
  setCharColor(DEFAULT);//文字色を元に戻す
}
```

コード 2 エスケープシーケンスを用いた特定位置への文字表示プログラム例(tetris1.c)

(1) 定数

```
#define 名前　定数
```

とするとプログラム中に現れた「名前」を「定数」に自動的に置換してくれる．C 言語では伝統的に定数の「名前」には大文字を使用する．以後のプログラムで大文字の単語を見たら，定数を意味していると思って間違いない．

コード 2 では，例えば次のように色の指定に定数を利用しており，プログラムで「1」の代わりに「RED」と表記できるようになる．

```
#define RED 1
```

#define を利用する理由は二つある．

①人間が読みやすい．人間にとっては数字よりも，その数字が意味する「名前」を使った方がわかりやすい．

②プログラム中の複数の場所で同じ数値を利用している時に，その数値を変更するには全部の箇所で書き換えが必要となり，ミスしやすい．#define を使っていれば，その一か所を書き換えるだけで，全部の箇所の修正が自動的に完了する．

(2) マクロ

```
#define 名前(引数)　引数を使った置き換え後のプログラム
```

マクロは，1 行で表現できる簡単なプログラム処理で，これも人間がわかりやすいように「名前」を付けて置換する．

```
#define setCharColor(n) printf("¥033[3%dm",(n))
```

このマクロは，コード 2 の例にあるように

```
  setCharColor(YELLOW);
```

とプログラム中で利用すると，実際には

```
printf("¥033[3%dm",(3));
```

と置き換えられて実行される．つまり黄色の文字色に変更するためのエスケープシーケンスが出力される．

この定数やマクロの置換を使用することで，一度定義した後は深く考えずに「setCharColor すれば表示文字色が変更できる」と考えてプログラムすればよい．

(3) 文字表示部分

コード 2 で一番肝心な部分が，表示※で示したメッセージを表示する部分である．

```
  printf("hello!"); fflush(stdout); //表示※
```

printf はそのまま文字列を画面表示するだけだが，fflush(stdout)は，stdout(標準出力，画面表示のこと)のバッファをフラッシュする．

ここでいうバッファは，標準出力へ出力する内容の文字を一時的に貯めておく機構のことで，貯めていた内容を実際に画面に出力することを，フラッシュと呼ぶ．

CPU の速度は，モニタ画面等の出力装置の速度に対して圧倒的に速い．このため，直接画面出力させると，出力装置の動作を CPU が待つための待ち時間が生じて，結果的にシステム全体の性能低下を招く．そこで，通常は pintf の出力を一度バッファに貯めて，ある程度出力文字が貯まってから，まとめて画面に出力させることで，画面出力回数を減らして待ち時間の軽減を図っている．

しかし，リアルタイムゲームにとっては致命的な，画面出力に時間的な遅れが生じることになる．そこで，printf 直後に明示的に fflush で遅延なしに画面出力ざせている．

[課題2]

(1) gedit で，コード 2 のプログラムを入力して tetris1.c として保存し，コンパイルし，実行してみよ．
(2) このプログラムを修正して，例えば画面上の 10 行目の 15 桁目に文字を赤く表示するにはどうしたらいいか，実際に試してみよ．
(3) ここでいう「10 行目」，「15 桁目」とはどこから数え始めた数字か確認せよ．(1 行目 1 桁目と 2 行目 2 桁目を表示して比較したりすること．)

3.2 STEP2 文字を時間とともに移動させる方法

(1) 失敗例

テトリスのブロックは時間とともに落下する．そこで，時間とともに任意の文字を上から下に移動させてみよう．

上から下への移動は，「一段階ずつ場所を移動させる」⇒「x 座標は同じで y 座標だけ一

つ下にずらす作業を繰り返す」と考えて実装する．繰り返しは for 文で書ける．for 文に関しては C 言語入門で説明した．

```
int main(int argc, char *argv[])
{
  int y;
  clearScreen();
  setBackColor(BLUE);
  setCharColor(YELLOW);
  for(y=1;y<23;y++)              //一行ずつ繰り返す(23 回)
  {
    setPosition(5,y);
    printf("$"); fflush(stdout);
  }
  setBackColor(DEFAULT);
  setCharColor(DEFAULT);
}
```

コード 3 一行ずつ下に文字を表示するプログラム

コード 3 は，コード 2 のうち変更した main 部分だけを抜書きしたものである．つまりコード 3 はこれだけでは動かなくて，コード 1 の残り部分(#define など)を追加して完成する．実際にコード 3 を動かしてみよ．残念ながらこれは期待した通りにはならない．

コード 3 に期待したのと違う動作は，

① 一瞬でプログラムが終了して文字が移動したようには見えない
② 上の行の文字が消えていないので移動したのではなく，分身したようだ

ということであろう．①の理由は PC の速度が速すぎるため，一番上の行から一番下の行の文字までほぼ同時表示して，一瞬でプログラムが終了してしまうためである．②は，そもそも上の行の文字を消すプログラムになっていないからである．

(2) 正しい方法

まず①から解決する．PC が速すぎて人間の目が追えないのであれば，時間を待たせればよい．例えば一段表示するたびに 0.5 秒何もせず待たせるプログラムを書く．例えば，

```
    wait(500); //ここで 500 ミリ秒(=0.5 秒)待つ
```

のようなプログラムを，文字を表示した後にはさんで，表示を一時停止できればよい．

時間を測るタイマーなどは，OS の機能である．そのため Windows と Linux ではプログラムの書き方が異なる．Linux では nanosleep 関数が用意されている．この関数は，指定したナノ秒単位でプログラムを一時停止(スリープ)させる機能を持つ．

nanosleep は，使用するために time.h をインクルードしておく必要があり，また引数には timespec 構造体(time.h の中で定義されている)を使用する．構造体に関しては「STEP3 ブロックのセルを落下させるようにする」で説明するので，ここではとりあえず次の wait 関数にラップして利用する．

```
#include <time.h>   //①必要なヘッダファイル
int wait(int msec); //①関数プロトタイプ宣言
int wait(int msec)  //②関数の本体
{
  struct timesepc r = {0, msec * 1000L * 1000L};
  return nanosleep( &r, NULL );
}
```

コード 4 wait 関数

この関数をプログラムに組み込むには，コード 5 のように 3 か所を追加する．①必要なヘッダファイルと関数プロトタイプ宣言を，プログラムの先頭付近に追加する．②関数本体を，main 後に追加する．③関数の利用部分を，main 中に追加する．

```
#include <stdio.h>
#include <time.h> //①のうち必要なヘッダファイルの部分
#define clearScreen()  printf("¥033[2J")
#define setPosition(x,y) printf("¥033[%d;%dH",(y),(x))
#define setCharColor(n) printf("¥033[3%dm",(n))
#define setBackColor(n) printf("¥033[4%dm",(n))
#define BLACK 0
#define RED 1
#define GREEN 2
#define YELLOW 3
#define BLUE 4
#define MAGENTA 5
#define CYAN 6
#define WHITE 7
#define DEFAULT 9
int wait(int msec); //①のうち関数プロトタイプの部分

int main(int argc, char *argv[])
{
  int y;
  clearScreen();
  setBackColor(BLUE);
  setCharColor(YELLOW);
  for(y=1;y<23;y++)
  {
    setPosition(5,y);
    printf("$"); fflush(stdout);
    wait(500); //③関数の利用．ここで 500 ミリ秒(=0.5 秒)待つ
  }
  setBackColor(DEFAULT);
  setCharColor(DEFAULT);
}

int wait(int msec) //②関数本体
{
  struct timespec r = {0, msec * 1000L * 1000L};
  return nanosleep( &r, NULL );
}
```

コード 5 wait を組み込んだ文字落下プログラム(tetris2.c)

ここでは，丁寧に全プログラムを示したが，これ以降では，コード 4 のように利用する関数だけを表記するので，それをコード 5 のように自分でプログラムに組み込めるようにしておくこと．

これで「①一瞬でプログラムが終了して文字が移動したようには見えない」は解決できたので，次の「②上の行の文字が消えていないので移動したのではなく，分身したようだ」を解決する．

分身を解決するのは簡単で，表示し終わった文字は画面から消去すればよい．これで「分身」ではなく，ちゃんと「移動」できる．画面消去は，すでに紹介している画面クリアを使うと簡単である．

```
clearScreen(); //画面クリア
```

これを，どのタイミングに(どの行に)挿入すれば，きちんと動くだろうか．

[課題3]

(1) コード 5 に clearScreen()を追加して tetris2.c として保存し，ちゃんと文字が下に移動するようにしてみよ．
(2) 落下速度を調節してみよ．
(3) 落下距離を変更してみよ．
(4) x 座標も同時に変化させて，斜めに移動させてみよ．(確認した後で，元の垂直落下に戻しておくこと)

3.3 STEP3 ブロックのセルを落下させるようにする

ここまでで，文字を落下させることができるようになった．次は，文字でなくテトリスのブロックを落としたい．テトリスのブロックは，図 8 のように複数の正方形から構成されている．この正方形を，ここではセルと呼ぼう．

図 8　テトリスのブロックとセル

まず，セル一個を落下させてみる．そもそもセルを表示するためには，文字を使ってセルのように見える表示を作成する．

Linux の端末は初期状態で文字が等幅フォントに設定され，一文字が固定幅で表示される．アルファベット等はいわゆる半角文字と呼ばれ，全角文字である漢字に対してちょうど半分の幅で画面表示される．日本語の全角文字のフォントは基本的に正方形サイズで作成されているので，半角文字を二文字並べると正方形サイズになる．

また，エスケープシーケンスで文字は明暗の反転表示ができる．そこで，空白文字「 」の明暗反転表示したものを二つ使ってセルを作成する．

文字の反転表示には，コード 6 のような文字属性のエスケープシーケンスを使用する．

```
#define setAttribute(n) printf("¥033[%dm",(n))
#define NORMAL 0   //通常
#define BLIGHT 1   //明るく
#define DIM 2              //暗く
#define UNDERBAR 4 //下線
```

```
#define BLINK 5    //点滅
#define REVERSE 7  //明暗反転
#define HIDE 8              //隠れ(見えない)
#define STRIKE 9   //取り消し線
```

コード 6 文字属性のエスケープシーケンス

このコード 6 を追加した上で，プログラム中で次のようにセルを表示する．

```
setCharColor(WHITE);
setBackColor(BLACK);
setAttribute(REVERSE);
printf("  "); //空白二文字
```

コード 7 セルの表示

逆にセルを消去するためには，同じ場所に今度は通常状態で空白文字を上書きすればよい．前のコード 5 のように clearScreen()で画面全体を消去すると，必要ない部分も消去されてゲーム画面を作れなくなる．ここでは必要なセルの部分だけを消去する．

```
setCharColor(WHITE);
setBackColor(BLACK);
setAttribute(NORMAL);
printf("  "); //空白二文字
```

コード 8 セルの消去

ここで，コード 5 の tetris2.c をコード 6 からコード 8 を使って書き換えると，コード 9 になる．(コード 9 は別名 tetris3.c で保存すること)

```
int main(int argc, char *argv[])
{
  int y;
  clearScreen();
  for(y=1;y<23;y++)
  {
    //セルを表示
    setPosition(5,y);
    setCharColor(WHITE);
    setBackColor(BLACK);
    setAttribute(REVERSE);
    printf("  "); //空白二文字
    fflush(stdout);

    //そのまま待つ
    wait(500);

    //セルを消去
    setPosition(5,y);
    setCharColor(WHITE);
    setBackColor(BLACK);
    setAttribute(NORMAL);
    printf("  "); //空白二文字
    fflush(stdout);
  }
  setBackColor(DEFAULT);
  setCharColor(DEFAULT);
```

```
  clearScreen();
}
```

コード 9 セルを落下させるプログラムの main 部分(tetris3.c)

[課題4]

(1) コード 9 を入力して, tetris3.c としてセルを落下させる動作を確認せよ.
(2) setAttribut を使った別の文字属性も表示してみてどうなるか試してみよ.

3.4 STEP4 セル落下プログラムの整理

今後のことを考えて，機能はほぼこのままで，少しプログラムを整理しておこう．次の四点を整理してプログラムをわかりやすく改良する．

(1) マクロと定数の追加
(2) 画面初期化と復元関数の用意
(3) セル構造体の利用
(4) セル表示関数とセル消去関数の追加

(1) マクロと定数の追加

まずマクロと定数をいくつかコード 10 に示すように追加しておく．

```
#define setPosition(x,y) printf("¥033[%d;%dH",(y)+1,(x)*2+1)
#define cursolOn()  printf("¥033[?25h")    //カーソルを表示
#define cursolOff()  printf("¥033[?25l")   //カーソルを非表示
#define WIDTH 10   //フィールドの幅
#define HEIGHT 20  //フィールドの高さ
```

コード 10 マクロの修正追加部分

今後はセル単位で表示すればよいので，setPosition はセル単位で表示できるように 2 桁ずつ計算する．また課題 2 で確かめたように座標が(1,1)からスタートするので，ちょっと気持ち悪い．後で使用する C 言語の配列とも相性が悪いので，座標(0,0)から指定できるように改良する．(次のマクロで座標(x,y)として座標(0,0)を指定すると，コード 9 に改良したように座標((y)+1, (x)*2+1)で示される座標はいくつになるか計算してみよ．また(1,1)は同様にどうなるか)

また，今までカーソルが表示されたままで気になっていたので，プログラム動作中にはカーソルを非表示にできるエスケープシーケンスを用意しておく．

最後に，ブロックを積み上げるフィールドの幅と高さも定数としておく．ここではテトリスの標準サイズとして幅 10 セル，高さ 20 セルとしておく

(2) 画面初期化と復元関数の用意

次に，端末画面の初期化と元の状態に復元する部分をちゃんと整理してコード 11 のような関数にしておこう．

```
void initialize(void); //画面の初期化（①関数プロトタイプ部分）
void reset(void);      //画面の復元（①関数プロトタイプ部分）

void initialize(void)  //画面の初期化（②関数本体）
{
  setBackColor(BLACK);
  setCharColor(WHITE);
  setAttribute(NORMAL);
  clearScreen();
  cursolOff();
}

void reset(void)       //画面の復元（②関数本体）
{
  setBackColor(BLACK);
  setCharColor(WHITE);
  setAttribute(NORMAL);
  clearScreen();
  cursolOn();
}
```

コード 11 画面の初期化と復元関数

STEP2 のコード 5 で wait 関数を組み込んだ時と同様に，この initialize と reset もプログラムに組み込む．それぞれの関数の③(関数の利用)として，main の中の最初と最後の部分にコード 12 のように挿入する．

```
int main(int argc, char *argv[])
{
  int y;
  initialize(); //画面の初期化③利用部分　その代り clearScreen()不要
…中略
  reset(); //画面の復元③利用部分　その代り setCharColor()setBackColor()clearSCreen()不要
}
```

コード 12 main での画面の初期化と復元関数の呼び出し方

(3) セル構造体の利用

コード 7 やコード 8 のように，セルを表示したり消去したりする時には必ず，表示する文字，表示する色，表示する文字背景色，表示する文字属性の 4 つを同時に printf で出力している．つまり，「セルは 4 つのプロパティから構成される」と考えよう．

今後多数のセルを管理する上で，この 4 つのプロパティは一体としてまとめて管理したい．C 言語では，このような複数のプロパティを一つのデータにまとめるために，構造体と呼ぶ仕組みを持っている．

この 4 つのプロパティを構造体として定義すると次のようになる．

```
typedef struct cell {
  char c;         //表示文字
  int charcolor;  //表示色
  int backcolor;  //背景色
  int attribute;  //文字属性
```

```
}   Cell;
```

コード 13 セル構造体

これは，文法的にはコード 14 のように意味している．

```
typedef struct 構造体名 {
  型 メンバ名; //構造体の要素 1
  型 メンバ名; //構造体の要素 2
  型 メンバ名; //構造体の要素 3
  型 メンバ名; //構造体の要素 4
}   構造体の型名;
```

コード 14 構造体の文法

「char c;」や「int charcolor;」といった普通の変数宣言文を，中括弧{}で括ったものである．その中括弧{}の部分に，構造体としての名前(構造体型の型名)を，最後に付けている．一行目の構造体名と最後の構造体の型名は違うものであることに注意．ここでは，最後の構造体の型名の方を主に利用する．

コード 13 は，あくまで構造体自体の宣言で，構造体を利用した変数数宣を，別にする必要がある．コード 13 の場合，構造体の型名として「Cell」を宣言したので，それを使った変数「a」を宣言するには，

```
Cell a = {' ', WHITE, BLACK, REVERSE};
```

のように記述する．この例では a の中の要素(メンバ)c には(空白)が，charcolor には WHITE が，backcolor には BLACK が，attribute には REVERSE が代入された状態で初期化される．初期化が必要ない場合には，初期化部分(=より右の部分)は省略可能である．

構造体型と変数の関係を，人間に例えると，構造体の型名が「人間」だとすれば変数の名前が個人名「日工太郎」や「宮代花子」となる．人間以外に「犬」「猫」等の型もあるだろう．コード 15 に，その比喩をプログラム形式で表した例を示す．ただし「犬」構造体「猫」構造体は省略している．

```
typedef struct human {
  int 身長;
  int 体重;
  int 性格;
  int 所属;
} 人間;

人間 日工太郎;
人間 宮代花子;
犬   ポチ;
猫   たま;
```

コード 15 構造体と変数の関係を説明するための比喩

テトリスでは，多数のセルを使用する．その一個一個のセルは個別の存在であり，名前を付けて管理する必要がある．その名前が，例えば変数「a」，人間の例では「日工太郎」

にあたる．セル構造を表す名前としての「Cell」とは別に，変数宣言が必要になる．

変数 a の中身(プロパティ)は，同じ Cell 型の別の変数とは異なって管理される．変数 a の charcolor と，別の名前を持った変数の charcolor には異なった値を代入できる．人間の比喩でいえば，変数「日工太郎」の中身の「身長」や「性格」は，変数「宮代花子」のそれとは異なる値を持っている．

また，C 言語の構造体では，プロパティに相当するものを，「要素(メンバ)」と呼ぶ．

(4) セル表示関数とセル消去関数の追加

セル構造体を定義できたので，そのセル型の変数を受け取って，変数のメンバを表示したり，消去したりする関数を作成しよう．

この関数の欲しい機能としては，「指定したセルを，指定した位置に表示する」と考える．つまり次のように利用できる関数にしよう．

```
printCell(a, 5, 10); //座標(5,10)にセル a を表示
clearCell(a, 5, 10); //座標(5,10)のセル a を消去
```

そしてこの時，指定した位置が正しいか(表示可能な座標か)もチェックして欲しい．そのチェック機能は，この二つの関数それぞれ内部で共通の機能と考えられるので，表示位置が正しいかどうかを判断する関数 checkRange も独立して用意する．そうすれば printCell も clearCell も，内部で checkRange を利用してチェックができるからである．

こういう場合，内部の checkRange から作成する．checkRange はコード 16 になる．

```
int checkRenge(Cell a, int x, int y);

int checkRange(Cell a, int x, int y)
{
  if(a.c=='¥0' || x<0 || y<0 || x>=WIDTH || y>=HEIGHT )
    return -1; //失敗
  else
    return 0;  //成功
}
```

コード 16 checkRange 関数

checkRange は単純な if 文で，引数 a として受け取ったセルの中身が表示可能かどうか，引数 x と y として受け取った座標が表示範囲内かどうかを条件判定する．

引数 a が表示可能かどうかは，a の中身(メンバ)の一つである表示文字 c が NULL 文字¥0 でないかどうかで判断する．本当は NULL 文字以外にも表示できない特殊文字はたくさんあるが，ここではちょっとサボっている．表示ができない¥0 の時は，例え x,y が表示可能範囲であっても失敗と見なす．

セルの構造体の変数 a のメンバを参照するには，ドット.演算子を使用する．メンバ c をアクセスするには

```
a.c
```

のように記述する．同様にメンバ charcolor を参照するには

```
a.charcolor
```

と記述する．

また，表示範囲は，座標 x,y がゲーム画面の幅と高さの範囲内かどうかチェックしている．範囲は (1)マクロと定数の追加で説明したように，定数 WIDTH と HEIGHT を定義したのでそれを利用した条件式を書いている．

判定結果は，失敗の場合-1 を，成功の場合 0 を返す(C 言語の伝統的な流儀)．

この checkRange を利用すれば，printCell と clearCell はコード 17 のように書ける．

```
int printCell(Cell c, int x, int y);
int clearCell(Cell c,int x, int y);

int printCell(Cell c, int x, int y)
{
  if(checkRange(c,x,y)==-1)
    return -1;
  setPosition(x, y);
  setAttribute(c.attribute);
  setBackColor(c.backcolor);
  setCharColor(c.charcolor);
  printf("%c%c", c.c, c.c);
  fflush(stdout);
  return 0;
}

int clearCell(Cell c, int x,int y)
{
  if(checkRange(c,x,y)==-1)
    return -1;
  setPosition(x, y);
  setAttribute(NORMAL);
  setBackColor(BLACK);
  setCharColor(BLACK);
  printf("  "); //空白 2 文字
  fflush(stdout);
  return 0;
}
```

コード 17 printCell 関数と clearCell 関数

この printCell と clearCell を使って，コード 9 (tetris2.c)の main を簡単にする．

(1)から(4)までを全部まとめるとコード 18 になる．ただし，コード 18 を見る前に，自分で(1)から(4)までをよく読んで tetris4.c として作成してほしい．コード 18 と一致しただろうか? #define の順番は気にしないが，必要な記述が全部そろっていただろうか．また Cell 構造体の宣言は Cell を使った関数プロトタイプの前にしないとエラーになるなど，変数や構造体の宣言の順番が重要になる箇所がある．

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

#define clearScreen()  printf("¥033[2J")
#define setPosition(x,y) printf("¥033[%d;%dH",(y)+1,(x)*2+1)
#define cursolOn()  printf("¥033[?25h")    //カーソルを表示
#define cursolOff()  printf("¥033[?25l")   //カーソルを非表示
```

```
#define WIDTH 10   //フィールドの幅
#define HEIGHT 20  //フィールドの高さ

#define setCharColor(n) printf("¥033[3%dm",(n))
#define setBackColor(n) printf("¥033[4%dm",(n))
#define BLACK 0
#define RED 1
#define GREEN 2
#define YELLOW 3
#define BLUE 4
#define MAGENTA 5
#define CYAN 6
#define WHITE 7
#define DEFAULT 9

#define setAttribute(n) printf("¥033[%dm",(n))
#define NORMAL 0   //通常
#define BLIGHT 1   //明るく
#define DIM 2                //暗く
#define UNDERBAR 4 //下線
#define BLINK 5    //点滅
#define REVERSE 7  //明暗反転
#define HIDE 8               //隠れ(見えない)
#define STRIKE 9   //取り消し線

typedef struct cell {
  char c;         //表示文字
  int charcolor;  //表示色
  int backcolor;  //背景色
  int attribute;  //文字属性
}   Cell;

int wait(int msec);      //待ち関数
void initialize(void);   //画面の初期化関数
void reset(void);        //画面の復元関数
int checkRenge(Cell a, int x, int y); //座標範囲チェック
int printCell(Cell c, int x, int y); //セル表示
int clearCell(Cell c,int x, int y);  //セル消去

int main(int argc, char *argv[])
{
  int y;
  Cell a = {' ', WHITE, BLACK, REVERSE};
  initialize();
  for(y=1;y<HEIGHT;y++)
  {
    printCell(a, 5, y); //セルを表示
    wait(500);          //そのまま待つ
    clearCell(a, 5, y); //セルを消去
  }
  reset();
}

int wait(int msec) //②関数の本体
{
  struct timespec r = {0, msec * 1000L * 1000L};
  return nanosleep( &r, NULL );
}

void initialize(void)  //画面の初期化 (②関数本体)
{
  setAttribute(NORMAL);
  setBackColor(BLACK);
  setCharColor(WHITE);
  clearScreen();
```

```
  cursolOff();
}

void reset(void)        //画面の復元 (②関数本体)
{
  setBackColor(BLACK);
  setCharColor(WHITE);
  setAttribute(NORMAL);
  clearScreen();
  cursolOn();
}

int checkRange(Cell a, int x, int y)
{
  if(a.c=='¥0' || x<0 || y<0 || x>=WIDTH || y>=HEIGHT )
    return -1; //失敗
  else
    return 0;  //成功
}

int printCell(Cell c, int x, int y)
{
  if(checkRange(c,x,y)==-1)
    return -1;
  setPosition(x, y);
  setAttribute(c.attribute);
  setBackColor(c.backcolor);
  setCharColor(c.charcolor);
  printf("%c%c", c.c, c.c);
  fflush(stdout);
  return 0;
}

int clearCell(Cell c, int x,int y)
{
  if(checkRange(c,x,y)==-1)
    return -1;
  setPosition(x, y);
  setAttribute(NORMAL);
  setBackColor(BLACK);
  setCharColor(BLACK);
  printf("  "); //空白 2 文字
  fflush(stdout);
  return 0;
}
```

コード 18 セル落下プログラム改良版(tetris4.c)

かなり長くなった印象があるが，関数はその機能を理解して正しく動作することを確認すれば，以降は関数の中身を気にしなくてよい．肝心の main は非常に単純になった．

[課題5]

(1) 作成した tetris4.c をきちんと動作させよ．うまく動かない場合コード 16 とよく見比べてみること．
(2) セルの色を変更してみよ．
(3) 初期表示位置を変更してみよ．
(4) フィールドの高さを変更してみよ．

3.5 STEP5 ブロックを落下させる

今度はセル一個ではなくて，複数のセルから構成されるブロックを落下させてみよう．ブロックをどのように構成するか考える．いろいろな方法あるが，単純にセルを二次元に並べて配置する方法が簡単そうだ．

(1) ブロックを配列として宣言する

オリジナルのテトリスのブロックは全部で 7 種類ある．図 9 のようにいずれも 4×4 のサイズで配置できる．

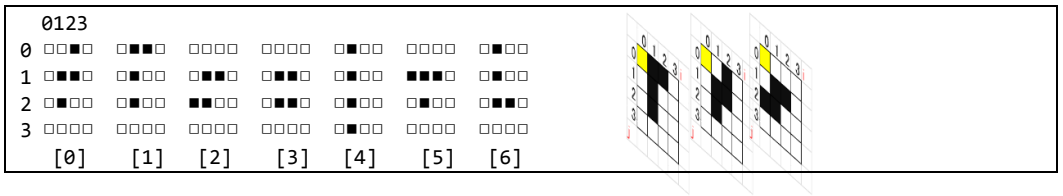


図 9 セル配置によるブロック

プログラムにおいて，何か(データ)を並べるには，配列が便利だ．STEP3 で導入した Cell 型を図 9(左)のような 4×4 の並べ方をするには

```
Cell block[4][4];
```

のように二次元配列を宣言する．ここでは縦と横のサイズ指定は次の順番としておく．

```
Cell block[縦のサイズ][横のサイズ];
```

さらに，このブロックが 7 種類あるので，全種類のブロック形状を表すには，図 9(右)のように，7×4×4 の三次元配列となる．

```
Cell block_type[7][4][4];
```

ここで，block と block_type を注意して区別しておく．block は現在落下中のブロックひとつを表しているが，block_type はブロックの全種類分用意されたテンプレートを表している．ひとつの block は，block_type のどれかからコピーされて作られる．

また，[4]や[7]のように定数をそのまま利用することは，3.3 の(1)で説明したようによくない．そこで次の定数を定義しておく．

```
#define BLOCK_SIZE 4
#define BLOCK_NUM 7
```

次に，block_type を図 9 のような 7 種類を表すように初期値を設定しよう．配列の初期化は block_type の宣言文をコード 19 のように書き換えて，tetris4.c の Cell 構造体の宣言後，main の前に挿入する．(tetris5.c)

```
#define BLOCK_SIZE 4
#define BLOCK_NUM 7
Cell block_type[BLOCK_NUM][BLOCK_SIZE][BLOCK_SIZE] = {
  '¥0', RED, BLACK, NORMAL,
  '¥0', RED, BLACK, NORMAL,
  ' ', RED, BLACK, REVERSE,
  '¥0', RED, BLACK, NORMAL,
```

```
  '¥0', RED, BLACK, NORMAL,
  ' ', RED, BLACK, REVERSE,
  ' ', RED, BLACK, REVERSE,
  '¥0', RED, BLACK, NORMAL,
  '¥0', RED, BLACK, NORMAL,
  ' ', RED, BLACK, REVERSE,
  '¥0', RED, BLACK, NORMAL,
  '¥0', RED, BLACK, NORMAL,
  '¥0', RED, BLACK, NORMAL,
  '¥0', RED, BLACK, NORMAL,
  '¥0', RED, BLACK, NORMAL,
  '¥0', RED, BLACK, NORMAL,

  '¥0', BLUE, BLACK, NORMAL,
  ' ', BLUE, BLACK, REVERSE,
  ' ', BLUE, BLACK, REVERSE,
  '¥0', BLUE, BLACK, NORMAL,
  '¥0', BLUE, BLACK, NORMAL,
  ' ', BLUE, BLACK, REVERSE,
  '¥0', BLUE, BLACK, NORMAL,
  '¥0', BLUE, BLACK, NORMAL,
  '¥0', BLUE, BLACK, NORMAL,
  ' ', BLUE, BLACK, REVERSE,
  '¥0', BLUE, BLACK, NORMAL,
  '¥0', BLUE, BLACK, NORMAL,
  '¥0', BLUE, BLACK, NORMAL,
  '¥0', BLUE, BLACK, NORMAL,
  '¥0', BLUE, BLACK, NORMAL,
  '¥0', BLUE, BLACK, NORMAL,

途中省略

  '¥0', WHITE, BLACK, NORMAL,
  ' ', WHITE, BLACK, REVERSE,
  '¥0', WHITE, BLACK, NORMAL,
  '¥0', WHITE, BLACK, NORMAL,
  '¥0', WHITE, BLACK, NORMAL,
  ' ', WHITE, BLACK, REVERSE,
  '¥0', WHITE, BLACK, NORMAL,
  '¥0', WHITE, BLACK, NORMAL,
  '¥0', WHITE, BLACK, NORMAL,
  ' ', WHITE, BLACK, REVERSE,
  ' ', WHITE, BLACK, REVERSE,
  '¥0', WHITE, BLACK, NORMAL,
  '¥0', WHITE, BLACK, NORMAL,
  '¥0', WHITE, BLACK, NORMAL,
  '¥0', WHITE, BLACK, NORMAL,
  '¥0', WHITE, BLACK, NORMAL,     //最後の「,」は有っても無くても可
};
```

コード 19 ブロックの形状データの宣言

block_type は Cell 型の配列なので，Cell の 4 つのメンバ c, charcolor，backcolor, attribute それぞれに対する値をその順番で並べて書いていく．値と値は「,」で区切る．

```
  '¥0', RED, BLACK, NORMAL,
```

これは，文字が NULL 文字，文字色が赤，背景が黒，通常の属性を示す．

```
' ', RED, BLACK, REVERSE,
```

これは，文字が空白文字，文字色が赤，背景が黒，明暗反転属性を示す．この二つでブロックの非表示部分と表示部分を表しており，図 9 における■と□に対応する．つまり

```
  '¥0', RED, BLACK, NORMAL,
  '¥0', RED, BLACK, NORMAL,
  ' ', RED, BLACK, REVERSE,
  '¥0', RED, BLACK, NORMAL,
```

これで「□□■□」を表す.

一つのブロック形状に対して 4×4 の 16 個分の Cell データが 16 行分，それが 7 種類ある．コード 19 では途中省略しているが，0，1，そして 6 種類目のブロックの初期値を示している．

(2) ブロックをコピーする関数

先ほど説明したように，block_type をテンプレートとして，block にコピーして利用する．しかし，配列と配列はそのままでは代入できない．面倒だが，配列の要素をひとつひとつ繰り返してコピーする必要がある．

これも単純な機能なので関数にしておこう(コード 20).

```
void copyBlock(Cell src[BLOCK_SIZE][BLOCK_SIZE], Cell dst[BLOCK_SIZE][BLOCK_SIZE]);

void copyBlock(Cell src[BLOCK_SIZE][BLOCK_SIZE], Cell dst[BLOCK_SIZE][BLOCK_SIZE])
{
  int i,j;
  for (j = 0; j < BLOCK_SIZE; j++)
    for (i = 0; i < BLOCK_SIZE; i++)
      dst[j][i]=src[j][i];
}
```

コード 20 copyBlock 関数

copyBlock 関数は，引数として渡されてきた src ブロックを dst ブロックに全要素コピーする．ブロックは二次元配列なので，for 文を二重に使用して縦成分と横成分両方を繰り返す．座標(i,j)でいえば(0,0)⇒(1,0)⇒(2,0)⇒(3,0)　⇒　(0,1)⇒(1,1)⇒(2,1)⇒(3,1)　⇒　(0,2)⇒(1,2)⇒(2,2)⇒(3,2)　⇒　(0,3)⇒(1,3)⇒(2,3)⇒(3,3)の順でコピーする(図 10).

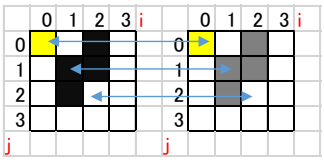
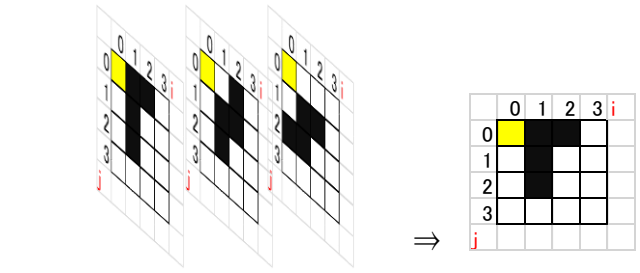


図 10　ブロックのコピー



block_type[0] block_type[1] block_type[2]

図 11　block_type[種類の番号]が二次元配列

copyBlock 関数は次のように利用できる．

```
copyBlock(block_type[0], block); //0 番目の形状を，変数 block へコピー
```

block_type は三次元配列であるが，コピーしたいのは 7 種類のうちどれか一つなので，

```
block_type[種類の番号]
```

のように指定する．図 11 に示したように，これで次元が一つ減って二次元配列となる．

(3) ブロックを表示・消去する関数

セルと同様に，画面にブロックを表示する関数と，消去する関数が必要である．すでにコード 17 で printCell 関数と clearCell 関数を作成しているので，これを利用して，printBlock と clearBlock を作成した(コード 21)．copyBlock と同様に二重の繰り返しで，ブロックに含まれる全部のセルを表示したり，消去したりしている．

```
int printBlock(Cell block[BLOCK_SIZE][BLOCK_SIZE], int x, int y);
int clearBlock(Cell block[BLOCK_SIZE][BLOCK_SIZE], int x, int y);

int printBlock(Cell block[BLOCK_SIZE][BLOCK_SIZE], int x, int y)
{
  int i, j;
  for (j = 0; j < BLOCK_SIZE; j++)
    for (i = 0; i < BLOCK_SIZE; i++)
      printCell(block[j][i], i + x, j + y);
  return 0;
}

int clearBlock(Cell block[BLOCK_SIZE][BLOCK_SIZE], int x, int y)
{
  int i, j;
  for (j = 0; j < BLOCK_SIZE; j++)
    for (i = 0; i < BLOCK_SIZE; i++)
      clearCell(block[j][i],i + x, j + y);
  return 0;
}
```

コード 21 printBlock 関数と clearBlock 関数

ここで注意するのは，各セルの表示座標である．ブロックは 4×4 のサイズなので，座

標(x,y)に全部表示できない．そこで，座標(x,y)を先頭(左上)とし，そこから 4×4 のエリアへブロックに含まれるセルを表示する．これは，4×4 の内部で(i,j)位置にあるセルを，

```
printCell(block[j][i], i + x, j + y);
```

として，最終的な画面上の座標(i+x,j+y)に変換して表示する．

例えば，セル 1 を座標(5,6)に表示する(図 12)．先頭の左上座標を(5,6)とし，ブロック内で(1,2)の位置にあるセルは，画面上では(1+5,2+4)，つまり(6,6)に表示することになる．

仮に出力する座標指定がゲームのフィールド外であっても，ブロック内のセル表示に printCell の中では checkRange によって判定しているので，問題ない．

また，ブロック内のセルを□■で表現したときに□に対応するセルは，NULL 文字を設定してあるため，printCell では何も表示せず，透明のように扱われる．もう一度コード 16 の checkRange とコード 17 の printCell と clearCell を見て確認しておくこと．

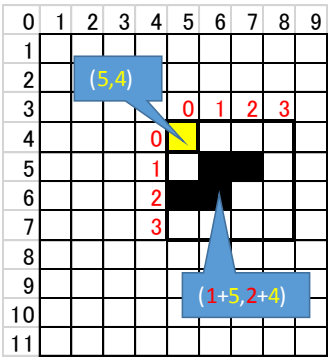


図 12 セルの表示座標の計算

(4) main の改造

tetris3.c にコード 19 からコード 21 を追加し，main を改造する(コード 22)．

```
int main(int argc, char *argv[])
{
  int i;
  Cell block[BLOCK_SIZE][BLOCK_SIZE];
  copyBlock(block_type[1],block);
  initialize();
  for(i=0;i<HEIGHT;i++)
  {
    printBlock(block,5,i);
    wait(500);
    clearBlock(block,5,i);
  }
  reset();
}
```

コード 22 ブロック落下プログラム(tetris5.c)

[課題6]

(1) コード 22 の tetris5.c をきちんと動作させよ.
(2) 落下させるブロックの種類を 3 番目のものや 5 番目にしてみよ.（データを変更せずに, main の中を一か所だけ変更すればできる）

3.6 STEP6 リアルタイムキー入力を行う

ここまでは，一切プレーヤーが操作できないプログラムであったが，ここでキーボードから入力できるようにしよう.

C 言語入門では，scanf 関数を利用してキーボードから数値や文字を入力した．しかし，scanf 関数では入力の最後にエンターキーを押さないと動作しない．テトリスのようなリアルタイムゲームで操作の度にエンターを押していては致命的である．カーソルキー(矢印キー)を押したらそのままブロックを移動させたい.

実は，Linux の端末でこのような機能を実現するには，OS のかなり深いところにあるハードウェア機能を直接たたく必要がある．さすがに Linux の高度なプログラムをここで説明するのは時間が足りない．ここでは細かい説明を省略し，使い方だけを説明する.

まず次のコード 23 を追加する.

```
#include <stdlib.h>       //追加のシステム系のヘッダファイル
#include <sys/types.h>
#include <signal.h>
#include <errno.h>
#include <termios.h>
#include <unistd.h>
extern int errno;
struct termios otty,ntty;
int kbhit(void); //キー入力があったかどうか確認
int getch(void); //キー入力１文字読み込み
int tinit(void); //端末の初期化

int kbhit(void)
{
  int ret;
  fd_set rfd;
  struct timeval timeout = {0,0};
  FD_ZERO(&rfd);
  FD_SET(0, &rfd); //0:stdin
  ret = select(1, &rfd, NULL, NULL, &timeout);
  if (ret == 1)
    return 1;
  else
    return 0;
}

int getch(void)
{
  unsigned char c;
  int n;
  while ((n = read(0, &c, 1)) < 0 && errno == EINTR) ;
  if (n == 0)
    return -1;
  else
    return (int)c;
}
```

```
static void onsignal(int sig) //内部利用のシグナルハンドラ
{
  signal(sig, SIG_IGN);
  switch(sig){
    case SIGINT:
    case SIGQUIT:
    case SIGTERM:
    case SIGHUP:
     exit(1);
     break;
  }
}

int tinit(void)
{
  if (tcgetattr(1, &otty) < 0)
    return -1;
  ntty = otty;
  ntty.c_iflag &= ~(INLCR|ICRNL|IXON|IXOFF|ISTRIP);
  ntty.c_oflag &= ~OPOST;
  ntty.c_lflag &= ~(ICANON|ECHO);
  ntty.c_cc[VMIN] = 1;
  ntty.c_cc[VTIME] = 0;
  tcsetattr(1, TCSADRAIN, &ntty);
  signal(SIGINT, onsignal);
  signal(SIGQUIT, onsignal);
  signal(SIGTERM, onsignal);
  signal(SIGHUP, onsignal);
}
```

コード 23 キー入力用の関数など

さらにコード 11 に端末初期化をコード 24 のように追加する．

```
void initialize(void)
{
  tinit(); //端末の初期化を追加
  setAttribute(NORMAL);
  setBackColor(BLACK);
  setCharColor(WHITE);
  clearScreen();
  cursolOff();
}

void reset(void)
{
  setBackColor(BLACK);
  setCharColor(WHITE);
  setAttribute(NORMAL);
  clearScreen();
  cursolOn();
  tcsetattr(1, TCSADRAIN, &otty); //端末の復元　追加
  write(1, "¥n", 1);              //後始末
}
```

コード 24 initialize と reset への端末の初期化/復元の追加修正

この修正の上で，諸君らがテトリスの実現のために使用するのは kbhit()と getch()である．kbhit()は，キーが何か押されているかどうかをチェックしてくれる関数である．getch()

は，実際に入力されているキーのキーコード番号を読み込む関数である．キーコードとはキーボードのキーに対応した番号であり，例えば「a」キーなら 0x61(16 進数表記)を示す．

これを使用したサンプルとして，コード 25 の main にいったん書き換えて，キー入力を確認してみよ．キーを押してみるとリアルタイムにキーコードが 16 進数で表示される．10 文字分入力すると終了する．

```
int main(int argc, char *argv[])
{
  int c,count;
  initialize();
  for(count=0 ; count<10 ; ) //とりあえず 10 文字読んだら終了
  {
    if(kbhit()!=0) //キーが押されていたら
    {
      c = getch(); //キーコードを読む
      printf("%x",c);
      count++;
    }
  }
  reset();
}
```

コード 25 リアルタイムキー入力(tetris6.c)

ここで，for 文は実は 10 回どころかかなりの回数(何万回も)ループしていることに注意して欲しい．for(初期化;条件;後始末)の構文のうち，後始末の部分はここでは空欄にしてあり毎回のカウントアップしていない．

for 文の中の if 文では，kbhit を判定している．kbhit は，キーが押された瞬間だけ 0 値以外を返してくるので，キーが押された瞬間だけ if 文の中が実行される．

for 文の後始末が空欄の代りに，if 文の中で count++しているので，キーが押された回数をカウントしていることになる．つまり，for 文自体はものすごい回数ループしているが，if 文の中は 10 回だけ実行される．

[課題7]

(1) コード 25 の tetris6.c を動かして，「a」を入力して「61」が表示されることを確認せよ．
(2) その他のキーを確認せよ．
(3) 矢印キーを入力してみよ．どうなったか．

3.7 STEP7 矢印キーでブロックを左右に動かす

(1) 単純キーでの操作

前の課題で確認したように，矢印キーのキーコードは，「a」などの文字キーと異なり，3 文字分 0x1b,0x5b,0x41 が一気に入力されるので，ちょっと入力処理が面倒だ．そこで，最初は矢印キーではなく，単純な「f」「j」キーを使って左右にブロックを動かしてみよう．

```
int main(int argc, char *argv[])
{
  int i,j,c,count;
  Cell block[BLOCK_SIZE][BLOCK_SIZE];
  copyBlock(block_type[1],block);
  initialize();
  i=5;
  j=10;
  printBlock(block,i,j); //初期表示
  for( count=0; count<10 ; ) //とりあえず 10 操作したら終了
  {
    if(kbhit()!=0)
    {
      clearBlock(block, i, j);
      c = getch();
      if(c=='f')
         i--;
      else if(c=='j')
         i++;
      else //それ以外の場合
      {
           reset();
           exit(1);
      }
      count++;
      printBlock(block,i,j);
    }
  }
  reset();
}
```

コード 26 f と j で左右に移動

このコード 26 では座標(5,10)からスタートして初期表示し,そこから先ほどと同様に for 文でキー入力を 10 回だけ繰り返す．キー入力があったときは，if 文の中で，最初にブロックを移動前に clearBlock で消去し，if 文を抜ける直前に新しい座標に printBlock で表示する．読み込んだキーコードが f か j かで横座標の値を++するか，--して一つずらす．それ以外のキーコードだった場合は，即座に終了する．

(2) 矢印キー(カーソルキー)での操作

実際には f と j では操作がしづらいので，矢印キーを使う．矢印キーは一回押されると先に説明した 3 文字が入力されてくるので，getch を 3 回呼び出して取得する(コード 27).

```
int main(int argc, char *argv[])
{
  int x,y,c,count;
  Cell block[BLOCK_SIZE][BLOCK_SIZE];
  copyBlock(block_type[1],block);
  initialize();
  x=5;
  y=10;
  printBlock(block,x,y); //初期表示
  count=0;
  for( count=0; count<10 ; ) //とりあえず 10 操作したら終了
  {
    if(kbhit())
```

```
    {
      clearBlock(block, x, y);
      c = getch();    //一回目
      if(c==0x1b)
      {
        c=getch();    //二回目
        if(c==0x5b)
        {
           c=getch(); //三回目
           switch(c)
           {
              case 0x41: //UP
                break;
              case 0x42: //DOWN
                break;
              case 0x43: //RIGHT
                x++;
                break;
              case 0x44: //LEFT
                x--;
                break;
           }
        }
      }
      else //矢印キー以外の場合
      {
          reset();
          exit(1);
      }
      count++;
      printBlock(block,x,y);
    }
  }
  reset();
}
```

コード 27 矢印キーで左右に移動(tetris7.c)

[課題8]

(1) コード 27 の tetris7.c を実行して，矢印キーで動作を確認せよ．(コード 26 は，理解できれば実行しなくてもよい．)
(2) 矢印キー以外の適当なキーを押したときに，ブロックを初期位置に戻すようにしてみよ．

3.8 STEP8 左右に動かしながら落下させる

(1) 単純な失敗例

コード 22 の落下プログラムと，今のコード 27 の左右に動かすプログラムを合わせて左右に動かしながら落下するプログラムを作ってみよう．単純に両者を結合してみるとコード 28 のようになるはずだ．

```
int main(int argc, char *argv[])
{
  int x,y,c;
  Cell block[BLOCK_SIZE][BLOCK_SIZE];
  copyBlock(block_type[1],block);
  initialize();
  x=5;
```

```
  y=0;
  printBlock(block,x,y); //初期表示
  for(y=0; y<HEIGHT ; y++) //自動的に落下
  {
    //落下プログラム
    printBlock(block,x,y);
    wait(500);
    clearBlock(block,x,y);

    //左右に動かすプログラム
    if(kbhit())
    {
      clearBlock(block, x, y);
      c = getch();    //一回目
      if(c==0x1b)
      {
        c=getch();    //二回目
        if(c==0x5b)
        {
           c=getch(); //三回目
           switch(c)
           {
              case 0x41: //UP
                break;
              case 0x42: //DOWN
                break;
              case 0x43: //RIGHT
                x++;
                break;
              case 0x44: //LEFT
                x--;
                break;
           }
        }
      }
      else //矢印キー以外の場合
      {
          reset();
          exit(1);
      }
      printBlock(block,x,y);
    }
  }
  reset();
}
```

コード 28 左右に移動しながら落下する(間違い)

しかし，これは思ったとおりに動かない．まずキー入力の反応が悪く，操作した後しばらくしてから動作する．しかも左右に動かすと，画面にごみが残る(実際に試してみよ).

前者の原因は，wait を使っているため 0.5 秒の間プログラムが一時停止しており，キー入力に即座に反応しないことである．また後者の原因は，落下によって縦位置が一つずれるため，printBlock と clearBlock の座標がずれて 0.5 秒前に表示したブロックを消せないことによる.

(2) 正しい方法

そこで，どう考えればよいか．ここでは，wait によってプログラムを一時停止せずに，時計を使って時間を測り，一定時間になったらブロックを落下させればよいと考える．時間計測中もプログラムは動作し続け，キー入力があれば反応させることにする．

Ubuntu の時計機能は，現在時刻を取得する gettimeofday があり，例えばコード 29 のように使用できる．

```
#include <time.h> //すでに導入済み

struct timeval start, end;
double duration;

gettimeofday(&start,NULL);
何か処理
gettimeofday(&end,NULL);
duration = end.tv_sec - start.tv_sec + (end.tv_usec - start.tv_usec)/1000000.0;
```

コード 29 gettimeofday の使用例

プログラムの最初の時刻を start に記憶しておき，処理後の時刻を end に記憶させる．end から start の時刻を引けば，処理時間を計測できる．

これを使って処理時間を計測し，あらかじめ定めた時間(thold)を超えたらブロックを一段落下させるようにする．

また，前回の表示位置を prex と prey に記憶しておく．キー操作で左右がずれたり，時間で一つ落下したりして，今回表示する座標の x と y が prex と prey と異なった時，前回表示位置でブロックを消去し，現在位置でブロックを表示するようにして，座標のずれに対処できるようにする．

正しいプログラムはコード 30 の通りとなる．

```
int main(int argc, char *argv[])
{
  int x,y,c,prex,prey;
  Cell block[BLOCK_SIZE][BLOCK_SIZE];
  struct timeval start_time, now_time, pre_time;
  double duration,thold;

  copyBlock(block_type[1],block);
  initialize();
  x=5;
  y=0;
  thold=0.5;  //落下の時間間隔
  printBlock(block,x,y); //初期表示
  gettimeofday(&start_time,NULL); //開始時刻
  pre_time = start_time;          //前回落下時刻  最初は開始時刻
  for(y=0; y<HEIGHT ; )
  {
    prex = x;
    prey = y;

    if(kbhit())
```

```
    {
      c = getch();    //一回目
      if(c==0x1b)
      {
        c=getch();    //二回目
        if(c==0x5b)
        {
           c=getch(); //三回目
           switch(c)
           {
              case 0x41: //UP
                break;
              case 0x42: //DOWN
                break;
              case 0x43: //RIGHT
                x++;
                break;
              case 0x44: //LEFT
                x--;
                break;
           }
        }
      }
      else //矢印キー以外の場合
      {
          reset();
          exit(1);
      }
    }
    gettimeofday(&now_time,NULL);
    duration = now_time.tv_sec-pre_time.tv_sec
           + (now_time.tv_usec-pre_time.tv_usec )/1000000.0; //前回からの経過時間
    if(duration > thold) //もしも落下時間間隔以上に時間経過していたら
    {
      pre_time = now_time; //前回落下時間を現在時刻に
      y++;                 //一つ落下
    }
    if(prex!=x||prey!=y)           //もしもブロックが左右移動/落下していたら
    {
      clearBlock(block,prex,prey); //前回位置のブロックを消して
      printBlock(block,x,y);       //新しい場所に表示
    }
  }
  reset();
}
```

コード 30 左右に移動しながら落下する(正解) (tetris8.c)

[課題9]

(1) tetris8.c を作成して動作を確認せよ.
(2) 落下間隔を調節してみよ.

3.9 STEP9 ブロックの回転

ここまでくれば，あとはもう一息である．後はブロックの回転，ブロックの積み重ね，ラインが揃った時に一行消滅，ができればとりあえずプレイ可能になる．

さて，次はブロックを回転させよう．ブロックの回転といっても，テトリスの場合 90° 単位で傾けるだけである．sin や cos を使って図形的に回転させなくても，行と列を入れ替

える等の操作だけで実現できる．

まず，行 j と列 i を入れ替える(図 13①⇒②)．次にそれを左右反転する(②⇒③)．これでブロックが時計回りに 90° 回転したことになる(①⇒③)．

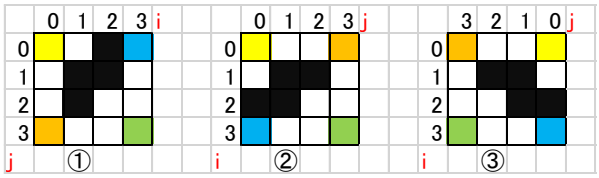


図 13 ブロックの回転

これをプログラムで表現するためには，まず①に相当するセルの配列 src と②に相当するセル配列 tmp を考える．src の各要素を，tmp の適切な場所にコピーさせる．ある(i,j)位置のセルは，(j,i)位置へ，

```
tmp[i][j] = src[j][i];
```

としてコピーすればよい．例えば①の右上(青)は(3,0)から②の(0,3)へコピーされる．

次に②の tmp から③の dst にコピーする操作は

```
dst[j][3-i] = src[j][i];
```

と書ける．ここでの 3 は，ブロックのサイズから計算できる最大の座標値で，正確に書けば

```
dst[j][BLOCK_SIZE-1 -i] = src[j][i];
```

となる．この二つの操作を合成して，src から直接 dst へコピーするためには

```
dst[i][BLOCK_SIZE-1 - j]=src[j][i];
```

となる．

これによりブロックを回転する関数 rotateBlock(コード 31)が作成できる．

```
void rotateBlock(Cell src[BLOCK_SIZE][BLOCK_SIZE], Cell dst[BLOCK_SIZE][BLOCK_SIZE]);

void rotateBlock(Cell src[BLOCK_SIZE][BLOCK_SIZE], Cell dst[BLOCK_SIZE][BLOCK_SIZE])
{
  int i,j;
  for (j = 0; j < BLOCK_SIZE; j++)
    for (i = 0; i < BLOCK_SIZE; i++)
      dst[i][BLOCK_SIZE-1 - j]=src[j][i];
}
```

コード 31 ブロックを回転する rotateBlock 関数

この rotateBlock を使って，↑キーが押されたときに回転するように main を書き換える．そのために，回転元の block だけでなく，回転後のブロックを一時格納する block_tmp を用意した上で，↑キーに対応する case 文のところに rotateBlock でブロックの回転，回転前の表示を消去，回転後のブロックを表示，回転したブロックでブロックを上書きするようにさせる(コード 32)．

```
int main(int argc, char *argv[])
{
  int x,y,c,prex,prey;
  Cell block[BLOCK_SIZE][BLOCK_SIZE], block_tmp[BLOCK_SIZE][BLOCK_SIZE];
  struct timeval start_time, now_time, pre_time;
  double duration,thold;

  copyBlock(block_type[1],block);
  initialize();
  x=5;
  y=0;
  thold=0.5;  //落下の時間間隔
  printBlock(block,x,y); //初期表示
  gettimeofday(&start_time,NULL); //開始時刻
  pre_time = start_time;          //前回落下時刻  最初は開始時刻
  for(y=0; y<HEIGHT ; )
  {
    prex = x;
    prey = y;

    if(kbhit())
    {
      c = getch();    //一回目
      if(c==0x1b)
      {
        c=getch();    //二回目
        if(c==0x5b)
        {
           c=getch(); //三回目
           switch(c)
           {
              case 0x41: //UP
                rotateBlock(block,block_tmp); //ブロック回転
                clearBlock(block,x,y);        //元のブロックを画面から消去
                printBlock(block_tmp,x,y);    //回転したブロックを表示
                copyBlock(block_tmp,block);   //回転したブロックを元のブロックに上書き
                break;
              case 0x42: //DOWN
                break;
              case 0x43: //RIGHT
                x++;
                break;
              case 0x44: //LEFT
                x--;
                break;
           }
        }
      }
      else //矢印キー以外の場合
      {
           reset();
           exit(1);
      }
    }
    gettimeofday(&now_time,NULL);
    duration = now_time.tv_sec-pre_time.tv_sec
           + (now_time.tv_usec-pre_time.tv_usec )/1000000.0; //前回からの経過時間
    if(duration > thold) //もしも落下時間間隔以上に時間経過していたら
    {
      pre_time = now_time; //前回落下時間を現在時刻に
      y++;                 //一つ落下
    }
    if(prex!=x||prey!=y)           //もしもブロックが左右移動/落下していたら
```

```
    {
      clearBlock(block,prex,prey); //前回位置のブロックを消して
      printBlock(block,x,y);       //新しい場所に表示
    }
  }
  reset();
}
```

コード 32 ブロックを回転する(tetris9.c)

[課題10]

(1) コード 31 とコード 32 の修正を加えた tetris9.c を作成して，動作を確かめよ．
(2) 反時計回りに回転させるとしたらどうしたらよいか考えよ．

3.10 STEP10 ランダムなブロック落下の繰り返し処理

ここまでで，ブロック単体の操作に関する部分が完成した．ただし，現在はブロック一個が落下した段階で終了していたが，続けて複数のブロックを落下させるような無限ループに変更しよう．また，ゲームにすることを考えて，落下するブロックは毎回ランダムな種類にしたい．

(1) ブロックの連続落下

まず，ブロックを連続落下させるには，for 文を無限ループに変更し，その代り今まで繰り返しの終了条件としていた「y<HEIGHT」の条件式を，ループの最後の方の y++をしていた部分でチェックするようにした．

これで，ブロックが落下していし終わっていない時だけ y++，落下し終わっていたら y を 0 に戻してやり直しさせている(コード 33)．

```
int main(int argc, char *argv[])
{
  int x,y,c,prex,prey;
  Cell block[BLOCK_SIZE][BLOCK_SIZE], block_tmp[BLOCK_SIZE][BLOCK_SIZE];
  struct timeval start_time, now_time, pre_time;
  double duration,thold;

  copyBlock(block_type[1],block);
  initialize();
  x=5;
  y=0;
  thold=0.5;
  printBlock(block,x,y);
  gettimeofday(&start_time,NULL);
  pre_time = start_time;
  for(; ; ) ////////////////////////無限ループに変更
  {
    prex = x;
    prey = y;

    if(kbhit())
    {
//////////////略(tetris9.c と同じ)////////////////
    }
```

```
    gettimeofday(&now_time,NULL);
    duration = now_time.tv_sec-pre_time.tv_sec
           + (now_time.tv_usec-pre_time.tv_usec )/1000000.0; //前回からの経過時間
    if(duration > thold)
    {
      pre_time = now_time;
      if(y<HEIGHT)          //ブロックが落下し終わっていなければ
        y++;                //一つ落下
      else                  //そうでない(ブロックが地上に着いた)とき
      {
        y=0;                //y を 0 に戻す(また最初から落下し直し)
        x=5;                //x も元に戻す
      }
    }
    if(prex!=x||prey!=y)            //もしもブロックが左右移動/落下していたら
    {
      clearBlock(block,prex,prey); //前回位置のブロックを消して
      printBlock(block,x,y);       //新しい場所に表示
    }
  }
  reset();
}
```

コード 33 ブロックを連続して落とす

(2) ブロックのランダム化

次に，ブロックの種類をランダム化する．C 言語でランダムな数を生成するには，rand 関数を使用する．rand 関数は，使用される度に正の整数を 0 から機械最大値(整数として通常扱える最大値)までの範囲で生成する関数である．

これを使って，

```
t = rand()%BLOCK_NUM;
```

のように rand を BLOCK_NUM(ブロックの種類数)で割った余りを計算すると，0 から BLOCK_NUM 未満のランダムな数を生成できる．

これを，最初にブロックを用意する部分で種類を決定する時に利用し，またブロックが落下し終わって最初の位置に戻る時に，次のブロックをランダムに決定する(コード 34).

```
int main(int argc, char *argv[])
{
  int x,y,c,prex,prey,t;
  Cell block[BLOCK_SIZE][BLOCK_SIZE], block_tmp[BLOCK_SIZE][BLOCK_SIZE];
  struct timeval start_time, now_time, pre_time;
  double duration,thold;

  t = rand()%BLOCK_NUM;           //最初のブロックの種類を決定
  copyBlock(block_type[t],block); //最初のブロックをテンプレからコピーして準備
  initialize();
  x=5;
  y=0;
  thold=0.5;
  printBlock(block,x,y);
  gettimeofday(&start_time,NULL);
  pre_time = start_time;
  for(; ; )
  {
    prex = x;
```

```
    prey = y;

    if(kbhit())
    {
//////////////略(tetris9.c と同じ)////////////////
    }
    gettimeofday(&now_time,NULL);
    duration = now_time.tv_sec-pre_time.tv_sec
           + (now_time.tv_usec-pre_time.tv_usec )/1000000.0; //前回からの経過時間
    if(duration > thold)
    {
      pre_time = now_time;
      if(y<HEIGHT)          //ブロックが落下し終わっていなければ
        y++;                //一つ落下
      else                  //そうでない(ブロックが地上に着いた)とき
      {
        y=0;                //y を 0 に戻す(また最初から落下し直し)
        X=5;
        t=rand()%BLOCK_NUM;             //次のブロック種類を決定
        copyBlock(block_type[t],block);  //テンプレからコピーし直し
      }
    }
    if(prex!=x||prey!=y)            //もしもブロックが左右移動/落下していたら
    {
      clearBlock(block,prex,prey); //前回位置のブロックを消して
      printBlock(block,x,y);       //新しい場所に表示
    }
  }
  reset();
}
```

コード 34 ブロックをランダムに変更(tetrisA.c)

[課題11]

(1) コード 34 の修正を加えた tetrisA.c を作成して，動作を確かめよ．たまたま同じ種類が続けて出ることもあるのであわてない．また，このプログラムは無限ループなので，ずっと待っていても終了しない．プリログラムをよく見るとわかるが，矢印キー以外の普通のキーをどれか押すことで終了する．

(2) フロックの種類を増やすにはどうしたらよいか．

3.11 STEP11 ブロックの積み重ね

いよいよブロックを積み重ねる．ブロックを積み上げるためには，座標だけを見ても判断がつかない．既に積み上がっているブロックと，落下してきたブロックの衝突を判定する必要がある．

このためには，ゲームフィールド全体を表現するマップを用意し，積み上がったブロックの各セルを記録しておく．そして，フィールドに新しく落下してきたブロックの各セルと，積み上がっているブロック，つまりマップ中に存在しているセルとの衝突判定を行う(図14)．

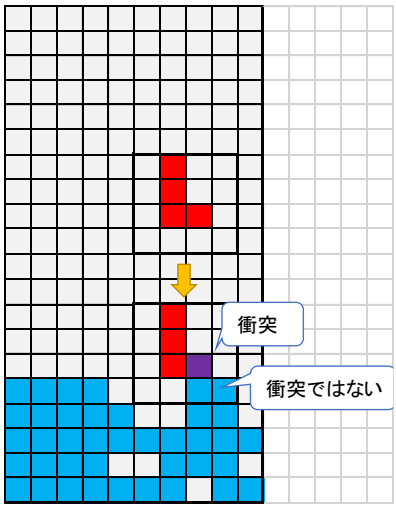


図 14　衝突判定

衝突判定で注意する点は二つ．一つはブロックの 4×4 のエリア内でも，表示する文字がない(メンバ c が NULL 文字)部分は，衝突ではないこと．もう一つは，マップ内に存在する積み上がったセルとはぶつからなくても，ゲームフィールドの外に出てしまう(境界線にぶつかる)ことも，衝突とみなすことである．

(1) マップの準備

まずマップを用意する．単純に WIDTH×HEIGHT の大きさのセルの配列があればよい．

```
Cell map[HEIGHT][WIDTH];
```

このマップは，様々な関数で参照されることが予想される．例えばマップ内にブロックを積み上げる，マップとブロックの衝突判定，マップから揃った一ラインを消滅させる，などがありそうだ．

こうした色々な関数からマップにアクセスできるように，map は大域変数(グローバル変数)としておこう．つまり，map の宣言は main 関数よりも上，Cell 構造体の宣言よりは後の位置で行う．

このマップデータは，ゲームスタート直後(プログラムの起動直後)の段階では，初期状態として空にしておく．そこで，初期化関数 initialize にマップのクリア機能を追加する(コード 35)．

```
void initialize(void)
{
  int x,y;
  Cell a = {'¥0',BLACK,BLACK,NORMAL};  //空のセルデータ
  tinit();
  setAttribute(NORMAL);
  setBackColor(BLACK);
  setCharColor(WHITE);
  clearScreen();
  cursolOff();
```

```
  for(y=0;y<HEIGHT;y++)
    for(x=0;x<WIDTH;x++)
      map[y][x] = a;        //マップを空に
}
```

コード 35 initialize へマップクリア追加

(2) 衝突判定関数，積み上げ関数

次に，ブロックの衝突判定とマップへブロックを登録して積み上げる関数を用意しよう．しかし，ブロックの衝突は，ブロック全体が積み上がったセルと重複するかどうかではなく，結局ブロック内の一つ以上のセルと，マップ内のセルとの衝突判定である．

そこで，3 つの関数を新規に作成する．セルの衝突判定，ブロックの衝突判定，マップへブロックを積み上げを行う関数である．

```
int checkCell(Cell a, int x, int y);                                    //セルの衝突検知
int checkMap(Cell block[BLOCK_SIZE][BLOCK_SIZE], int x, int y);         //ブロックの衝突検知
void putMap(Cell block[BLOCK_SIZE][BLOCK_SIZE], int x, int y);          //マップにセルを登録
```

まず指定セル a が座標 x,y でマップ上のセルと衝突しているかチェックを行う checkCell を作成する(コード 36)．境界条件をチェックする checkRange(コード 16)で座標 x,y がフィールド外かどうか，a の表示文字が NULL 文字でないか，map[y][x]にセルが存在するかどうか，全部チェックして，失敗(その位置にセルを表示しない)か成功(その位置にセルを表示する)かを決める．

```
int checkCell(Cell a, int x, int y)
{
  if(checkRange(a,x,y) || map[y][x].c != '¥0')
    return -1; //失敗
  return 0;    //成功
}
```

コード 36 checkCell 関数(マップとセルの衝突判定)

これを使って，ブロックとの衝突判定をするための checkMap を用意する(コード 37)．ブロック内 4×4 の全セルを checkCell でチェックし，どれか表示できないセルがあれば失敗，全部表示できる場合成功を返す．

```
int checkMap(Cell block[BLOCK_SIZE][BLOCK_SIZE], int x, int y)
{
  int i, j;
  for (j = 0; j < BLOCK_SIZE; j++)
    for (i = 0; i < BLOCK_SIZE; i++) {
      if (block[j][i].c != '¥0')
      {
        if(checkCell(block[j][i],x+i,y+j))
            return -1; //どれか一つでも失敗なら，ブロック全体として失敗
      }
    }
  return 0; //全部表示できそうならば成功
}
```

コード 37 checkMap 関数(マップとブロックの衝突判定)

最後に putMap として block を座標 x,y でマップに登録して「積み上げる」を表現しよう(コード 38).

```
void putMap(Cell block[BLOCK_SIZE][BLOCK_SIZE], int x, int y)
{
  int i, j;
  for (j = 0; j < BLOCK_SIZE; j++)
    for (i = 0; i < BLOCK_SIZE; i++) {
      if(checkCell(block[j][i],x+i,y+j)==0)
        map[y + j][x + i] = block[j][i];
    }
}
```

コード 38 putMap 関数(マップにブロックを積み上げる)

ブロック内のセル位置 i,j とフィールドの座標 x,y の間の関係は，3.5 の(3)ブロックを表示・消去する関数の図 12 セルの表示座標の計算ですでに説明してある.

(3) キー入力時の横方向衝突判定

マップと衝突判定を実現するために必要な関数は以上である．tetrisB.c に追加しておこう．ただし，これだけでは動作は変更しない．これらの衝突判定をブロックの移動時のタイミングで行うように動作するように組み込む必要がある.

まず, main の中のキー入力の部分で左右に動かす部分に衝突判定を組み込む(コード 39).

```
            case 0x42: //DOWN
              while(checkMap(block,x,y+1)==0)
                y++;
              break;
            case 0x43: //RIGHT
              if(checkMap(block,x+1,y)==0)
                x++;
              break;
            case 0x44: //LEFT
              if(checkMap(block,x-1,y)==0)
                x--;
              break;
```

コード 39 キー入力時の衝突判定

右移動できるかは, 試しに x+1 の座標で衝突判定してみて, もしも成功できたら x++する．左も同様に考える．今までは左右キーで無制限にブロックが移動できてしまい，ブロックが見えなくなることもあったが，これによりブロックがフィールドの外に飛び出ることがなくなった．ついでだが，↓キーでぶつかるまで while 文で繰り返し一気に下まで落とす機能を追加した.

(4) 縦方向の衝突判定

次に縦方向の判定で，main の後半部分で今まで if(y<=HEIGHT)でフィールドの高さと座標 y の比較で下まで落ち切っていたかどうか判定していた部分を，コード 40 のように衝突判定するように変更する．そしてブロックが衝突していたら，マップに登録する．

```
if(duration > thold)
{
  pre_time = now_time;
  if(checkMap(block,x,y+1)==0)          //ブロックが落下し終わっていなければ
    y++;                  //一つ落下
  else                    //そうでない(ブロックが地上に着いた)とき
  {
    putMap(block,x,y);  //ブロックを積み上げる(x,y 変更前に行う)
    X=5;
    y=0;                  //y を 0 に戻す(また最初から落下し直し)
    prex=5;
    prey=0;
    t=rand()%BLOCK_NUM;               //次のブロック種類を決定
    copyBlock(block_type[t],block);  //テンプレからコピーし直し
    printBlock(block,x,y);            //表示
  }
}
if(prex!=x||prey!=y)            //もしもブロックが左右移動/落下していたら
{
  clearBlock(block,prex,prey); //前回位置のブロックを消して
  printBlock(block,x,y);        //新しい場所に表示
}
```

コード 40 マップへの積み上げ

また，prex,prey(移動前のブロック位置)も修正する．これがないと積み上がるはずのブロックがその下の clearBlock で画面上から消去されてしまう．

ここでの最後に，天井まで積み上がった時に終了するようにしておこう(コード 41).

```
if(duration > thold)
{
  pre_time = now_time;
  if(checkMap(block,x,y+1)==0)          //ブロックが落下し終わっていなければ
    y++;                  //一つ落下
  else                    //そうでない(ブロックが地上に着いた)とき
  {
    if (y == 0) {         //y が 0(天井)だったら，
      reset();            //終了処理
      exit(1);
    }
    putMap(block,x,y);  //ブロックを積み上げる(x,y 変更前に行う)
    X=5;
    y=0;                  //y を 0 に戻す(また最初から落下し直し)
    prex=5;
    prey=0;
    t=rand()%BLOCK_NUM;               //次のブロック種類を決定
    copyBlock(block_type[t],block);  //テンプレからコピーし直し
    printBlock(block,x,y);            //表示
  }
}
```

コード 41 天井まで積み上がった終了判定

最終的にコード 42(tetrisB.c)としてブロックの積み上げに対応した main を示す.

```
int main(int argc, char *argv[])
{
  int x,y,c,prex,prey;
  Cell block[BLOCK_SIZE][BLOCK_SIZE], block_tmp[BLOCK_SIZE][BLOCK_SIZE];
  struct timeval start_time, now_time, pre_time;
  double duration,thold;
  copyBlock(block_type[1],block);
  initialize();
  x=5;
  y=0;
  thold=0.5;
  printBlock(block,x,y);
  gettimeofday(&start_time,NULL);
  pre_time = start_time;
  for(; ; ) ////////////////////////無限ループに変更
  {
    prex = x;
    prey = y;

    if(kbhit())
    {
      c = getch();    //一回目
      if(c==0x1b)
      {
        c=getch();    //二回目
        if(c==0x5b)
        {
           c=getch(); //三回目
           switch(c)
           {
              case 0x41: //UP
                rotateBlock(block,block_tmp);
                clearBlock(block,x,y);
                printBlock(block_tmp,x,y);
                copyBlock(block_tmp,block);
                break;
              case 0x42: //DOWN
                while(checkMap(block,x,y+1)==0)
                  y++;
                break;
              case 0x43: //RIGHT
                if(checkMap(block,x+1,y)==0)
                  x++;
                break;
              case 0x44: //LEFT
                if(checkMap(block,x-1,y)==0)
                  x--;
                break;
    }
    gettimeofday(&now_time,NULL);
    duration = now_time.tv_sec-pre_time.tv_sec
           + (now_time.tv_usec-pre_time.tv_usec )/1000000.0; //前回からの経過時間
    if(duration > thold)
    {
      pre_time = now_time;
      if(checkMap(block,x,y+1)==0)          //ブロックが落下し終わっていなければ
        y++;                 //一つ落下
      else                   //そうでない(ブロックが地上に着いた)とき
      {
```

```
        if (y == 0) {        //y が 0(天井)だったら，
          reset();           //終了処理
          exit(1);
        }
        putMap(block,x,y);  //ブロックを積み上げる(x,y 変更前に行う)
        X=5;
        y=0;                //y を 0 に戻す(また最初から落下し直し)
        prex=5;
        prey=0;
        t=rand()%BLOCK_NUM;              //次のブロック種類を決定
        copyBlock(block_type[t],block);  //テンプレからコピーし直し
        printBlock(block,x,y);           //表示
      }
    }
    if(prex!=x||prey!=y)            //もしもブロックが左右移動/落下していたら
    {
      clearBlock(block,prex,prey); //前回位置のブロックを消して
      printBlock(block,x,y);       //新しい場所に表示
    }
  }
  reset();
}
```

コード 42 ブロックの積み重ね(tetrisB.c)

[課題12]

(1) tetrisB.c の動作を確かめよ.
(2) カーソルキーの case 文で，↓キーの while 文と，→キーと←キーの if 文で，どのように動作が違うか．互いに変更したらどうなるか.

3.12 STEP12 ライン消滅

後は揃ったラインを消滅させれば，一応テトリスとしての動作が完成する．一行揃っているかどうかは，ブロックが積み重なるタイミングで，マップの情報を調べればできそうだ.

ここでは次のように考える.

まずマップ全体から揃ったラインを削除する関数を用意することにした．すると，その関数ではラインが揃っているか結局一行一行調べることになる．そこで一行揃っているかどうか判定する関数，揃っていたら一行消滅させる関数を用意した．また，一行消滅させたら，画面表示を更新する必要もある.

最終的に次の 4 つの関数を用意することにした.

```
void printMap(void);       //画面にマップに積み重なったセルを全部表示する
int checkLine(int y);      //一行揃ったかどうか
void deleteLine(int ys);   //揃った行を一行マップから消滅させる
void deleteMap(void);      //揃った行を全部マップ上から消去する
```

それぞれの中身は，単純なプログラムで実現できる(コード 43).

```
void printMap(void)
{
  int x,y;
  for(y=0;y<HEIGHT;y++)
    for(x=0;x<WIDTH;x++)
```

```
      printCell(map[y][x],x,y);
}

int checkLine(int y)
{
  int x;
  for(x=0;x<WIDTH;x++)
    if(map[y][x].c=='¥0') //どこか一か所でも空セルがあれば
      return -1;          //失敗
  return 0;               //全部揃っている
}

void deleteLine(int ys)
{
  int x,y;
  for(y=ys;y>0;y--)          //対象ラインより上のラインを繰り返し
    for(x=0;x<WIDTH;x++)     //一行全部繰り返し
      map[y][x]=map[y-1][x]; //一つ上のセルを下に落とす
  setBackColor(BLACK);
  clearScreen();             //ゲーム画面をクリアして
  printMap();                //マップに従って全部再描画
}

void deleteMap(void)
{
  int y;
  for(y=0;y<HEIGHT;y++)     //ゲーム画面の全部の行を繰り返し
    if(checkLine(y)==0)     //揃ったラインがあれば
      deleteLine(y);        //一行消滅
}
```

コード 43 printMap 関数，checkLine 関数，deleteLine 関数，deleteMap 関数

main で使用するのはこのうち deleteMap だけでよく，そのタイミングは先ほど述べたように, 衝突が起こった(落下し終わった)時で, ブロックを積み上げた後である(コード 44).

```
    if(duration > thold)
    {
      pre_time = now_time;
      if(checkMap(block,x,y+1)==0)          //ブロックが落下し終わっていなければ
        y++;                 //一つ落下
      else                   //そうでない(ブロックが地上に着いた)とき
      {
        if (y == 0) {        //y が 0(天井)だったら，
          reset();           //終了処理
          exit(1);
        }
        putMap(block,x,y);  //ブロックを積み上げる(x,y 変更前に行う)
        deleteMap();        //ライン消滅確認
        X=5;
        y=0;                 //y を 0 に戻す(また最初から落下し直し)
        prex=5;
        prey=0;
        t=rand()%BLOCK_NUM;              //次のブロック種類を決定
        copyBlock(block_type[t],block);  //テンプレからコピーし直し
        printBlock(block,x,y);           //表示
      }
    }
```

コード 44 ライン消滅プログラムの main 変更部分(tetrisC.c)

[課題13]

(1) tetrisC.c の動作を確かめよ.
(2) deleteMap()の挿入位置を前後にずらしたら，どんな不都合が生じるか.

3.13 STEP13 Next 表示

ようやくゲームができるようになった．後はゲームしやすさのために次のブロックを表示したり，得点を表示したりして完成である.

Next 表示をどこに表示するのか問題もあるが，とりあえず最上段に Next ブロックを表示することにしよう．そのため，ゲームフィールド全体を 4 行下にずらす．まず高さを 4 つ増やす.

```
#define HEIGHT 24
```

そして，ブロックが落下し始める初期位置の高さを変更する．Next ブロック表示分の 4 行分，つまり BLOCK_SIZE からスタートさせる(コード 45).

```
int main(int argc, char *argv[])
{
  int x,y,c,prex,prey,t;
  Cell block[BLOCK_SIZE][BLOCK_SIZE], block_tmp[BLOCK_SIZE][BLOCK_SIZE];
  struct timeval start_time, now_time, pre_time;
  double duration,thold;

  t = rand()%BLOCK_NUM;            //最初のブロックの種類を決定
  copyBlock(block_type[t],block); //最初のブロックをテンプレからコピーして準備
  initialize();
  x=5;
  y=BLOCK_SIZE;//初期高さ
  thold=0.5;
```

コード 45 初期高さの変更

また，落下しきった後，最初の位置に戻す部分も修正しておく(コード 46).

```
    if(duration > thold)
    {
      pre_time = now_time;
      if(checkMap(block,x,y+1)==0)          //ブロックが落下し終わっていなければ
        y++;                 //一つ落下
      else                   //そうでない(ブロックが地上に着いた)とき
      {
        if (y == BLOCK_SIZE) {        //y が天井だったら，
          reset();           //終了処理
          exit(1);
        }
        putMap(block,x,y);  //ブロックを積み上げる(x,y 変更前に行う)
        deleteMap();   //ライン消滅機能
        X=5;
        y=BLOCK_SIZE;                 //y を BLOCK_SIZE に戻す(また最初から落下し直し)
        prex=5;
        prey=0;
        t=rand()%BLOCK_NUM;              //次のブロック種類を決定
        copyBlock(block_type[t],block);  //テンプレからコピーし直し
        printBlock(block,x,y);           //表示
```

```
    }
  }
```

コード 46 ブロックを初期位置に戻す部分の修正

この上で，Next ブロックを作成する．最初にランダムにブロックを決定した時に，次に落とすブロックもあらかじめ決定し，表示しておく必要がある．

とりあえず，ブロックの種類番号を受け取って固定された場所にそのブロックを表示する関数として printNext を用意することにしておこう．コード 47 の printNext は単にブロックを表示と前回の表示を消去する．消去しないと画面にごみが残るためである．

```
void printNext(int type)
{
  int i, j;
  Cell a = {' ',BLACK,BLACK,NORMAL};
  setPosition(0,0);       //座標 0,0 に文字表示
  setAttribute(NORMAL);
  setBackColor(WHITE);
  setCharColor(BLACK);
  printf("NEXT");
  for (j = 0; j < BLOCK_SIZE; j++)
    for (i = 0; i < BLOCK_SIZE; i++)
      printCell(a,5+i,0+j);          //空のセルを表示して，画面消去
  printBlock(block_type[type],5,0); //座標 5,0 にブロック表示
}
```

コード 47 printNext 関数(次のブロックを固定位置に表示)

この printNext は，最初に次のブロックを決定した後と，ブロックが落下しきって次のブロックに切り替える時に使用する(コード 48).

```
int main(int argc, char *argv[])
{
  int x,y,c,prex,prey,t,next;  //next 追加
  Cell block[BLOCK_SIZE][BLOCK_SIZE], block_tmp[BLOCK_SIZE][BLOCK_SIZE];
  struct timeval start_time, now_time, pre_time;
  double duration,thold;

  t = rand()%BLOCK_NUM;            //最初のブロックの種類を決定
  next = rand()%BLOCK_NUM;         //次のブロックの種類も決定
  copyBlock(block_type[t],block); //最初のブロックをテンプレからコピーして準備
  initialize();
  printNext(next);                //次のブロックを表示しておく
  x=5;
  y=BLOCK_SIZE;//初期高さ
  thold=0.5;

////////////////////////////////途中省略

    if(duration > thold)
    {
      pre_time = now_time;
      if(checkMap(block,x,y+1)==0)          //ブロックが落下し終わっていなければ
        y++;                 //一つ落下
      else                   //そうでない(ブロックが地上に着いた)とき
      {
```

```
        if (y == BLOCK_SIZE) {        //y が天井だったら，
          reset();           //終了処理
          exit(1);
        }
        putMap(block,x,y);  //ブロックを積み上げる(x,y 変更前に行う)
        deleteMap();   //ライン消滅機能
        X=5;
        y=BLOCK_SIZE;                 //y を BLOCK_SIZE に戻す(また最初から落下し直し)
        prex=5;
        prey=0;
        t=next;                          //next を新しい t にして今度落下させる
        next=rand()%BLOCK_NUM;           //次の次のブロック種類を決定
        printNext(next);                 //今決めた次の次のブロックを表示
        copyBlock(block_type[t],block);  //テンプレからコピーし直し
        printBlock(block,x,y);           //表示
      }
    }
```

コード 48 Next ブロックの表示(tetrisD.c)

最初に決めて表示する際には，initialize の後であることに注意．initialize でいったん画面クリアするため，その後に printNext しないと意味がない．

[課題14]

(1) tetrisD.c を実行して確認せよ．
(2) 初めの printNext を initialize の前に移すとどうなるか．

3.14 STEP14 得点表示と時間調整

いよいよ最後の機能である．得点を計算して表示させよう．テトリスの得点はドロップとライン得点がある．

まず，main の中で得点を計算する変数を用意しておく(コード 49).

```
int main(int argc, char *argv[])
{
  int x,y,c,prex,prey,t,next,score;  //score 追加
  Cell block[BLOCK_SIZE][BLOCK_SIZE], block_tmp[BLOCK_SIZE][BLOCK_SIZE];
  struct timeval start_time, now_time, pre_time;
  double duration,thold;
  score = 0;   //当然 score は 0 点からスタート
```

コード 49 score の宣言と初期化

(1) ドロップ得点

ドロップ得点は，落下距離をそのまま点数にするものである(図 15).

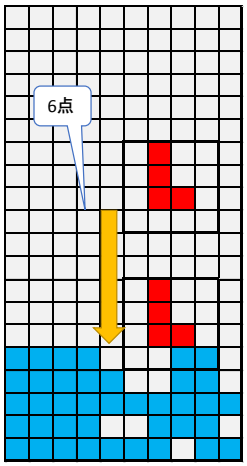


図 15　ドロップ得点

この計算は，↓キーを押してドロップした距離を，y の増分として算出しよう．移動前の y 座標値は prey に保存されているので，ドロップ後の y と引き算して求める．それを score に加えこむ．main のキー入力部分で↓キーのところに次の一行を追加する(コード 50).

```
            case 0x42: //DOWN
              while(checkMap(block,x,y+1)==0)
                y++;
              score += y-prey;
```

コード 50　ドロップ得点の加算

(2) ライン得点

消滅ラインの数に応じて得点させよう．任意に決めてよいが，昔のゲーム機のテトリスでは表 2 のようになっていた．

表 2 ライン得点

ライン数	得点
1 行(シングル)	40
2 行(ダブル)	100
3 行(トリプル)	300
4 行(テトリス)	1200

これを計算するためには，まずはラインを消滅させる deleteMap でライン数をリターンしてもらう必要がある(コード 51).

```
int deleteMap(void) //int を返すように変更
{
  int y,count;
  count=0; //ライン数を数える count を追加
  for(y=0;y<HEIGHT;y++)     //ゲーム画面の全部の行を繰り返し
```

```
    if(checkLine(y)==0)     //揃ったラインがあれば
    {
      deleteLine(y);        //一行消滅
      count++;              //その時カウントアップ
    }
  return count; //ライン数を報告
}
```

コード 51 deleteMap の修正(ライン数のカウント)

これを使い，衝突後の処理の中で次の部分を追加する(コード 52).

```
      if(checkMap(block,x,y+1)==0)
        y++;
      else
      {
         int line;
         if (y == BLOCK_SIZE) {
           reset();
           exit(1);
         }
         putMap(block,x,y);
         line = deleteMap();
         switch(line)
         {
           case 1: score+=40;  break;
           case 2: score+=100; break;
           case 3: score+=300; break;
           case 4: score+=1200;break;
         }
         x=5;
         y=BLOCK_SIZE;
         prex=5;
         prey=BLOCK_SIZE;
         t=next;
         next=rand()%BLOCK_NUM;
         printNext(next);
         copyBlock(block_type[t],block);
         printBlock(block,x,y);
      }
```

コード 52 ライン得点の計算

(3) 得点表示

せっかく計算した得点を画面に表示しないといけない．表示関数を用意する(コード 53).

```
void printScore(int score)
{
  setPosition(WIDTH,10);
  setAttribute(NORMAL);
  setBackColor(WHITE);
  setCharColor(BLACK);
  printf("Score: %d",score);
}
```

コード 53 printScore 関数(得点表示)

これは，最初の画面と，得点が加算されるタイミングで使用して，その場で得点表示させよう．得点加算のタイミングとは，コード 50 とコード 52 の得点計算直後に

```
printScore(score);
```

を挿入すればよい．

(4) 落下速度の調整

最後の最後だが，テトリスは経過時間や消滅させたライン数に応じて，ブロックの落下時間が早められていく．このあたりはゲームバランスの調整次第で，いろいろな工夫が考えられる．

ここでは，単純にブロックがひとつ落下するたびに徐々にスピードを上げてみよう(コード 54)．

```
    if(duration > thold)
    {
      pre_time = now_time;
      thold -= 0.001;
```

コード 54 落下速度の調整

ブロックの落下時間は main の変数 thold で指定してあった．これを，ブロックが落下するごとに減らしていけば，スピードをアップすることができる．

最終的に完成した main はコード 55 のようになる．

```
int main(int argc, char *argv[])
{
  int x,y,c,prex,prey,t,next,score;
  Cell block[BLOCK_SIZE][BLOCK_SIZE], block_tmp[BLOCK_SIZE][BLOCK_SIZE];
  struct timeval start_time, now_time, pre_time;
  double duration,thold;
  score=0;
  t=rand()%BLOCK_NUM;
  next=rand()%BLOCK_NUM;
  copyBlock(block_type[t],block);
  initialize();
  printNext(next);
  printScore(score);
  x=5;
  y=BLOCK_SIZE;
  thold=0.5;
  printBlock(block,x,y);
  gettimeofday(&start_time,NULL);
  pre_time = start_time;
  for(;  ; )
  {
    prex = x;
    prey = y;
    if(kbhit())
    {
      c = getch();
      if(c==0x1b)
      {
        c=getch();
```

```
      if(c==0x5b)
      {
         c=getch();
         switch(c)
         {
            case 0x41:
              rotateBlock(block,block_tmp);
              clearBlock(block,x,y);
              printBlock(block_tmp,x,y);
              copyBlock(block_tmp,block);
              break;
            case 0x42:
              while(checkMap(block,x,y+1)==0)
                y++;
              score+=y-prey;
              printScore(score);
              break;
            case 0x43:
              if(checkMap(block,x+1,y)==0)
                x++;
              break;
            case 0x44:
              if(checkMap(block,x-1,y)==0)
                x--;
              break;
         }
      }
    }
    else
    {
        reset();
        exit(1);
    }
  }
  gettimeofday(&now_time,NULL);
  duration = now_time.tv_sec-pre_time.tv_sec
         + (now_time.tv_usec-pre_time.tv_usec )/1000000.0;
  if(duration > thold)
  {
    pre_time = now_time;
    thold -= 0.001;
    if(checkMap(block,x,y+1)==0)
      y++;
    else
    {
       int line;
       if (y == BLOCK_SIZE) {
         reset();
         exit(1);
       }
       putMap(block,x,y);
       line=deleteMap();
       switch(line)
       {
         case 1: score+=40;  break;
         case 2: score+=100; break;
         case 3: score+=300; break;
         case 4: score+=1200;break;
       }
       printScore(score);
       x=5;
       y=BLOCK_SIZE;
       prex=5;
       prey=BLOCK_SIZE;
       t=next;
```

```
        next=rand()%BLOCK_NUM;
        printNext(next);
        copyBlock(block_type[t],block);
        printBlock(block,x,y);
      }
    }
    if(prex!=x||prey!=y)
    {
      clearBlock(block,prex,prey);
      printBlock(block,x,y);
    }
  }
  reset();
}
```

コード 55 得点表示・速度の調整(tetrisE.c)

[課題15]

(1) tetrisE.c を動かしてみよ.
(2) 得点の配転，時間調整などゲームバランスを調整せよ.
(3) 特殊なブロック，例えば爆弾機能を持ったブロックなどを実現するためには，どうしたらよいか.

4. おわりに

ここまでがんばった人はお疲れのことと思う．大変だったと思うがこれでかなりプログラムの実力が付いたはずである．

プログラムは，さほど難しくない単純機能(ブロックを落下させる，回転させるなど)を，ただ根気よく少しずつ付け加えていく作業である．実用的なソフトウェアは，かなり多数の機能から構成されている．それを根気よく少しずつ構築していく作業になる．これは，確かに人によって向き不向きがある．

この作業は楽しかっただろうか，また，機会があればまた別のプログラムを作ってみようと思っただろうか．あるいは，この作業に没頭してしまい，通学途中や風呂，トイレの中で無意識にプログラムのバグを探していたり，改良点を探していたりした人もいたかもしれない．(えーと思う人もいるかもしれないが，例えばある RPG をやっていて先に進めない時に，風呂の中で「あ，まだマップのここでこのアイテムを試してない」等と気づくのに似ている．年がら年中プログラムのことを考えてしまうのはプログラマの性分だろう．)

いずれかの体験をした人なら，間違いなくプログラマに向いていると断言できる．そうは思わない人も，プログラミングの実際を体験できたはずだ．

このテトリスはまだ完成度が低い．画面をもう少しきれいにしたり，ゲームバランスを調整したり，ステージ制を導入して面クリアをさせてみたり，まだまだ工夫の余地が残されている．ぜひ次のプログラミング体験を自分から挑戦してみてほしい．

ちなみに，ここで作成したテトリスプログラムは 500 行規模のもので，プログラムとしては単純な方である(表 3 ソフトウェアの規模による特徴)．まだまだ先は長い．色々改良してみて 1,000 行を超えるぐらいを目指してみよう．

実際に卒研や仕事で作成しなければならないプログラムはもっと長いだろう．しかし，行う作業は今回と変わらない．途中であきらめさえしなければ，完成に間違いなくたどり着ける．最後に，あなたの今後にこの経験が生きることを，切に願うばかりである．

表 3 ソフトウェアの規模による特徴

行数	説明	難しさ	期間	例
～100 行	おもちゃ	鼻歌交じりの思い付きでも動くレベル	1 日	C 言語入門の課題
～1,000 行	作業用，簡単なフリーソフト	適当にやっても，試行錯誤で完成できるレベル	1 週間	本稿のテトリス
～10,000 行	実用	まじめにエラー処理や構造化を考えないと完成できないレベル	数か月	役に立つフリーソフト 卒論のプログラム
～100,000 行	値段がつく売り物	ソフトウェアテストツール，バージョン管理システムなどの開発支援がないと無理	1 年	シェアウェア
～1,000,000 行	主力製品	複数人が業務として作成しないといけないレベル	1 年～数年	パッケージソフト
100 万行以上	社運をかけた巨大プロジェクト	100 人超の体制で，しかも優秀なプロジェクトマネージャがいないと破たんするレベル	数年	OS， Office，PhotoShop など

※この表は，きわめて感覚的なものである．あまり鵜呑みにしないように．

付録　完成したプログラム全体(554 行)

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <sys/types.h>
#include <signal.h>
#include <errno.h>
#include <termios.h>
#include <unistd.h>
extern int errno;
struct termios otty,ntty;
int kbhit(void);
int getch(void);
int tinit(void);

#define clearScreen()  printf("¥033[2J")
#define setPosition(x,y) printf("¥033[%d;%dH",(y)+1,(x)*2+1)
#define cursolOn()  printf("¥033[?25h")
#define cursolOff()  printf("¥033[?25l")
#define WIDTH 10
#define HEIGHT 24

#define setCharColor(n) printf("¥033[3%dm",(n))
#define setBackColor(n) printf("¥033[4%dm",(n))
#define BLACK 0
#define RED 1
#define GREEN 2
#define YELLOW 3
#define BLUE 4
#define MAGENTA 5
#define CYAN 6
#define WHITE 7
#define DEFAULT 9

#define setAttribute(n) printf("¥033[%dm",(n))
#define NORMAL 0
#define BLIGHT 1
#define DIM 2
#define UNDERBAR 4
#define BLINK 5
#define REVERSE 7
#define HIDE 8
#define STRIKE 9
#define BLOCK_SIZE 4
#define BLOCK_NUM 7

typedef struct cell {
  char c;
  int charcolor;
  int backcolor;
  int attribute;
}   Cell;

int wait(int msec);
void initialize(void);
void reset(void);
int checkRenge(Cell a, int x, int y);
int printCell(Cell c, int x, int y);
int clearCell(Cell c,int x, int y);
int printBlock(Cell block[BLOCK_SIZE][BLOCK_SIZE], int x, int y);
int clearBlock(Cell block[BLOCK_SIZE][BLOCK_SIZE], int x, int y);
void copyBlock(Cell src[BLOCK_SIZE][BLOCK_SIZE], Cell dst[BLOCK_SIZE][BLOCK_SIZE]);
```

```
void rotateBlock(Cell src[BLOCK_SIZE][BLOCK_SIZE], Cell dst[BLOCK_SIZE][BLOCK_SIZE]);
int checkCell(Cell a, int x, int y);
int checkMap(Cell block[BLOCK_SIZE][BLOCK_SIZE], int x, int y);
void putMap(Cell block[BLOCK_SIZE][BLOCK_SIZE], int x, int y);
void printMap(void);
int checkLine(int y);
void deleteLine(int ys);
int deleteMap(void);
void printNext(int type);
void printScore(int score);

Cell block_type[BLOCK_NUM][BLOCK_SIZE][BLOCK_SIZE] = {
  '¥0', RED, BLACK, NORMAL,
  '¥0', RED, BLACK, NORMAL,
  ' ', RED, BLACK, REVERSE,
  '¥0', RED, BLACK, NORMAL,
  '¥0', RED, BLACK, NORMAL,
  ' ', RED, BLACK, REVERSE,
  ' ', RED, BLACK, REVERSE,
  '¥0', RED, BLACK, NORMAL,
  '¥0', RED, BLACK, NORMAL,
  ' ', RED, BLACK, REVERSE,
  '¥0', RED, BLACK, NORMAL,
  '¥0', RED, BLACK, NORMAL,
  '¥0', RED, BLACK, NORMAL,
  '¥0', RED, BLACK, NORMAL,
  '¥0', RED, BLACK, NORMAL,
  '¥0', RED, BLACK, NORMAL,

  '¥0', BLUE, BLACK, NORMAL,
  ' ', BLUE, BLACK, REVERSE,
  ' ', BLUE, BLACK, REVERSE,
  '¥0', BLUE, BLACK, NORMAL,
  '¥0', BLUE, BLACK, NORMAL,
  ' ', BLUE, BLACK, REVERSE,
  '¥0', BLUE, BLACK, NORMAL,
  '¥0', BLUE, BLACK, NORMAL,
  '¥0', BLUE, BLACK, NORMAL,
  ' ', BLUE, BLACK, REVERSE,
  '¥0', BLUE, BLACK, NORMAL,
  '¥0', BLUE, BLACK, NORMAL,
  '¥0', BLUE, BLACK, NORMAL,
  '¥0', BLUE, BLACK, NORMAL,
  '¥0', BLUE, BLACK, NORMAL,
  '¥0', BLUE, BLACK, NORMAL,

  '¥0', GREEN, BLACK, NORMAL,
  '¥0', GREEN, BLACK, NORMAL,
  '¥0', GREEN, BLACK, NORMAL,
  '¥0', GREEN, BLACK, NORMAL,
  '¥0', GREEN, BLACK, NORMAL,
  ' ', GREEN, BLACK, REVERSE,
  ' ', GREEN, BLACK, REVERSE,
  '¥0', GREEN, BLACK, NORMAL,
  ' ', GREEN, BLACK, REVERSE,
  ' ', GREEN, BLACK, REVERSE,
  '¥0', GREEN, BLACK, NORMAL,
  '¥0', GREEN, BLACK, NORMAL,
  '¥0', GREEN, BLACK, NORMAL,
  '¥0', GREEN, BLACK, NORMAL,
  '¥0', GREEN, BLACK, NORMAL,
  '¥0', GREEN, BLACK, NORMAL,

  '¥0', YELLOW, BLACK, NORMAL,
  '¥0', YELLOW, BLACK, NORMAL,
```

```
  '¥0', YELLOW, BLACK, NORMAL,
  '¥0', YELLOW, BLACK, NORMAL,
  '¥0', YELLOW, BLACK, NORMAL,
  ' ', YELLOW, BLACK, REVERSE,
  ' ', YELLOW, BLACK, REVERSE,
  '¥0', YELLOW, BLACK, NORMAL,
  '¥0', YELLOW, BLACK, NORMAL,
  ' ', YELLOW, BLACK, REVERSE,
  ' ', YELLOW, BLACK, REVERSE,
  '¥0', YELLOW, BLACK, NORMAL,
  '¥0', YELLOW, BLACK, NORMAL,
  '¥0', YELLOW, BLACK, NORMAL,
  '¥0', YELLOW, BLACK, NORMAL,
  '¥0', YELLOW, BLACK, NORMAL,

  '¥0', CYAN, BLACK, NORMAL,
  '¥0', CYAN, BLACK, NORMAL,
  '¥0', CYAN, BLACK, NORMAL,
  '¥0', CYAN, BLACK, NORMAL,
  ' ', CYAN, BLACK, REVERSE,
  ' ', CYAN, BLACK, REVERSE,
  ' ', CYAN, BLACK, REVERSE,
  ' ', CYAN, BLACK, REVERSE,
  '¥0', CYAN, BLACK, NORMAL,
  '¥0', CYAN, BLACK, NORMAL,
  '¥0', CYAN, BLACK, NORMAL,
  '¥0', CYAN, BLACK, NORMAL,
  '¥0', CYAN, BLACK, NORMAL,
  '¥0', CYAN, BLACK, NORMAL,
  '¥0', CYAN, BLACK, NORMAL,
  '¥0', CYAN, BLACK, NORMAL,

  '¥0', MAGENTA, BLACK, NORMAL,
  '¥0', MAGENTA, BLACK, NORMAL,
  '¥0', MAGENTA, BLACK, NORMAL,
  '¥0', MAGENTA, BLACK, NORMAL,
  ' ', MAGENTA, BLACK, REVERSE,
  ' ', MAGENTA, BLACK, REVERSE,
  ' ', MAGENTA, BLACK, REVERSE,
  '¥0', MAGENTA, BLACK, NORMAL,
  '¥0', MAGENTA, BLACK, NORMAL,
  ' ', MAGENTA, BLACK, REVERSE,
  '¥0', MAGENTA, BLACK, NORMAL,
  '¥0', MAGENTA, BLACK, NORMAL,
  '¥0', MAGENTA, BLACK, NORMAL,
  '¥0', MAGENTA, BLACK, NORMAL,
  '¥0', MAGENTA, BLACK, NORMAL,
  '¥0', MAGENTA, BLACK, NORMAL,

  '¥0', WHITE, BLACK, NORMAL,
  ' ', WHITE, BLACK, REVERSE,
  '¥0', WHITE, BLACK, NORMAL,
  '¥0', WHITE, BLACK, NORMAL,
  '¥0', WHITE, BLACK, NORMAL,
  ' ', WHITE, BLACK, REVERSE,
  '¥0', WHITE, BLACK, NORMAL,
  '¥0', WHITE, BLACK, NORMAL,
  '¥0', WHITE, BLACK, NORMAL,
  ' ', WHITE, BLACK, REVERSE,
  ' ', WHITE, BLACK, REVERSE,
  '¥0', WHITE, BLACK, NORMAL,
  '¥0', WHITE, BLACK, NORMAL,
  '¥0', WHITE, BLACK, NORMAL,
  '¥0', WHITE, BLACK, NORMAL,
  '¥0', WHITE, BLACK, NORMAL,
```

```
};
Cell map[HEIGHT][WIDTH];

int main(int argc, char *argv[])
{
  int x,y,c,prex,prey,t,next,score;
  Cell block[BLOCK_SIZE][BLOCK_SIZE], block_tmp[BLOCK_SIZE][BLOCK_SIZE];
  struct timeval start_time, now_time, pre_time;
  double duration,thold;

  score=0;
  t=rand()%BLOCK_NUM;
  next=rand()%BLOCK_NUM;
  copyBlock(block_type[t],block);
  initialize();
  printNext(next);
  printScore(score);
  x=5;
  y=BLOCK_SIZE;
  thold=0.5;
  printBlock(block,x,y);
  gettimeofday(&start_time,NULL);
  pre_time = start_time;
  for(;  ; )
  {
    prex = x;
    prey = y;

    if(kbhit())
    {
      c = getch();
      if(c==0x1b)
      {
        c=getch();
        if(c==0x5b)
        {
           c=getch();
           switch(c)
           {
              case 0x41:
                rotateBlock(block,block_tmp);
                clearBlock(block,x,y);
                printBlock(block_tmp,x,y);
                copyBlock(block_tmp,block);
                break;
              case 0x42:
                while(checkMap(block,x,y+1)==0)
                  y++;
                score+=y-prey;
                printScore(score);
                break;
              case 0x43:
                if(checkMap(block,x+1,y)==0)
                  x++;
                break;
              case 0x44:
                if(checkMap(block,x-1,y)==0)
                  x--;
                break;
           }
        }
      }
      else
      {
           reset();
```

```
          exit(1);
      }
    }
    gettimeofday(&now_time,NULL);
    duration = now_time.tv_sec-pre_time.tv_sec
           + (now_time.tv_usec-pre_time.tv_usec )/1000000.0;
    if(duration > thold)
    {
      pre_time = now_time;
      thold -= 0.001;
      if(checkMap(block,x,y+1)==0)
        y++;
      else
      {
         int line;
         if (y == BLOCK_SIZE) {
           reset();
           exit(1);
         }
         putMap(block,x,y);
         line=deleteMap();
         switch(line)
         {
           case 1: score+=40;  break;
           case 2: score+=100; break;
           case 3: score+=300; break;
           case 4: score+=1200;break;
         }
         printScore(score);
         x=5;
         y=BLOCK_SIZE;
         prex=5;
         prey=BLOCK_SIZE;
         t=next;
         next=rand()%BLOCK_NUM;
         printNext(next);
         copyBlock(block_type[t],block);
         printBlock(block,x,y);
      }
    }
    if(prex!=x||prey!=y)
    {
      clearBlock(block,prex,prey);
      printBlock(block,x,y);
    }
  }
  reset();
}

int wait(int msec)
{
  struct timespec r = {0, msec * 1000L * 1000L};
  return nanosleep( &r, NULL );
}

void initialize(void)
{
  int x,y;
  Cell a = {'¥0',BLACK,BLACK,NORMAL};
  tinit();
  setAttribute(NORMAL);
  setBackColor(BLACK);
  setCharColor(WHITE);
  cursolOff();
  clearScreen();
```

```
  fflush(stdout);
  for(y=0;y<HEIGHT;y++)
    for(x=0;x<WIDTH;x++)
      map[y][x] = a;
}

void reset(void)
{
  setBackColor(BLACK);
  setCharColor(WHITE);
  setAttribute(NORMAL);
  clearScreen();
  cursolOn();
  tcsetattr(1, TCSADRAIN, &otty);
  write(1, "¥n", 1);
}

int checkRange(Cell a, int x, int y)
{
  if(a.c=='¥0' || x<0 || y<0 || x>=WIDTH || y>=HEIGHT )
    return -1;
  else
    return 0;
}

int printCell(Cell c, int x, int y)
{
  if(checkRange(c,x,y)==-1)
    return -1;
  setPosition(x, y);
  setAttribute(c.attribute);
  setBackColor(c.backcolor);
  setCharColor(c.charcolor);
  printf("%c%c", c.c, c.c);
  fflush(stdout);
  return 0;
}

int clearCell(Cell c, int x,int y)
{
  if(checkRange(c,x,y)==-1)
    return -1;
  setPosition(x, y);
  setAttribute(NORMAL);
  setBackColor(BLACK);
  setCharColor(BLACK);
  printf("  ");
  fflush(stdout);
  return 0;
}

void copyBlock(Cell src[BLOCK_SIZE][BLOCK_SIZE], Cell dst[BLOCK_SIZE][BLOCK_SIZE])
{
  int i,j;
  for (j = 0; j < BLOCK_SIZE; j++)
    for (i = 0; i < BLOCK_SIZE; i++)
      dst[j][i]=src[j][i];
}
int printBlock(Cell block[BLOCK_SIZE][BLOCK_SIZE], int x, int y)
{
  int i, j;
  for (j = 0; j < BLOCK_SIZE; j++)
    for (i = 0; i < BLOCK_SIZE; i++)
      printCell(block[j][i], i + x, j + y);
  return 0;
```

```
}

int clearBlock(Cell block[BLOCK_SIZE][BLOCK_SIZE], int x, int y)
{
  int i, j;
  for (j = 0; j < BLOCK_SIZE; j++)
    for (i = 0; i < BLOCK_SIZE; i++)
      clearCell(block[j][i],i + x, j + y);
  return 0;
}
int kbhit(void)
{
  int ret;
  fd_set rfd;
  struct timeval timeout = {0,0};
  FD_ZERO(&rfd);
  FD_SET(0, &rfd);
  ret = select(1, &rfd, NULL, NULL, &timeout);
  if (ret == 1)
    return 1;
  else
    return 0;
}

int getch(void)
{
  unsigned char c;
  int n;
  while ((n = read(0, &c, 1)) < 0 && errno == EINTR) ;
  if (n == 0)
    return -1;
  else
    return (int)c;
}

static void onsignal(int sig)
{
  signal(sig, SIG_IGN);
  switch(sig){
    case SIGINT:
    case SIGQUIT:
    case SIGTERM:
    case SIGHUP:
     exit(1);
     break;
  }
}

int tinit(void)
{
  if (tcgetattr(1, &otty) < 0)
    return -1;
  ntty = otty;
  ntty.c_iflag &= ~(INLCR|ICRNL|IXON|IXOFF|ISTRIP);
  ntty.c_oflag &= ~OPOST;
  ntty.c_lflag &= ~(ICANON|ECHO);
  ntty.c_cc[VMIN] = 1;
  ntty.c_cc[VTIME] = 0;
  tcsetattr(1, TCSADRAIN, &ntty);
  signal(SIGINT, onsignal);
  signal(SIGQUIT, onsignal);
  signal(SIGTERM, onsignal);
  signal(SIGHUP, onsignal);
}
```

```
void rotateBlock(Cell src[BLOCK_SIZE][BLOCK_SIZE], Cell dst[BLOCK_SIZE][BLOCK_SIZE])
{
  int i,j;
  for (j = 0; j < BLOCK_SIZE; j++)
    for (i = 0; i < BLOCK_SIZE; i++)
      dst[i][BLOCK_SIZE-1 - j]=src[j][i];
}

int checkCell(Cell a, int x, int y)
{
  if(checkRange(a,x,y) || map[y][x].c != '¥0')
    return -1;
  return 0;
}
int checkMap(Cell block[BLOCK_SIZE][BLOCK_SIZE], int x, int y)
{
  int i, j;
  for (j = 0; j < BLOCK_SIZE; j++)
    for (i = 0; i < BLOCK_SIZE; i++) {
      if (block[j][i].c != '¥0')
      {
        if(checkCell(block[j][i],x+i,y+j))
            return -1;
      }
    }
  return 0;
}
void putMap(Cell block[BLOCK_SIZE][BLOCK_SIZE], int x, int y)
{
  int i, j;
  for (j = 0; j < BLOCK_SIZE; j++)
    for (i = 0; i < BLOCK_SIZE; i++) {
      if(checkCell(block[j][i],x+i,y+j)==0)
        map[y + j][x + i] = block[j][i];
    }
}

void printMap(void)
{
  int x,y;
  for(y=0;y<HEIGHT;y++)
    for(x=0;x<WIDTH;x++)
      printCell(map[y][x],x,y);
}

int checkLine(int y)
{
  int x;
  for(x=0;x<WIDTH;x++)
    if(map[y][x].c=='¥0')
      return -1;
  return 0;
}

void deleteLine(int ys)
{
  int x,y;
  for(y=ys;y>0;y--)
    for(x=0;x<WIDTH;x++)
      map[y][x]=map[y-1][x];
  setBackColor(BLACK);
  clearScreen();
  printMap();
}
```

```
int deleteMap(void)
{
  int y,count;
  count=0;
  for(y=0;y<HEIGHT;y++)
    if(checkLine(y)==0)
    {
      deleteLine(y);
      count++;
    }
  return count;
}

void printNext(int type)
{

  int i, j;
  Cell a = {' ',BLACK,BLACK,NORMAL};
  setPosition(0,0);
  setAttribute(NORMAL);
  setBackColor(WHITE);
  setCharColor(BLACK);
  printf("NEXT");
  for (j = 0; j < BLOCK_SIZE; j++)
    for (i = 0; i < BLOCK_SIZE; i++)
      printCell(a,5+i,0+j);
  printBlock(block_type[type],5,0);
}

void printScore(int score)
{
  setPosition(WIDTH,10);
  setAttribute(NORMAL);
  setBackColor(WHITE);
  setCharColor(BLACK);
  printf("Score: %d",score);
}
```