

プログラミング及び演習

第11回 Cursesライブラリ& プログラミングプロジェクト課題

(2017/07/14)

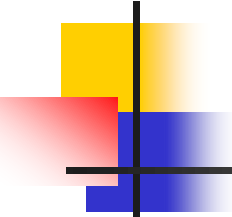
講義担当

大学院情報学研究科知能システム学専攻

教授 森 健策

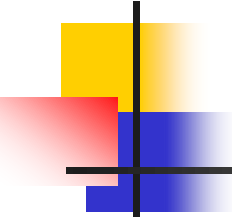
大学院情報学研究科知能システム学専攻

助教 小田 昌宏



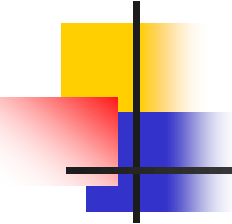
本日の講義・演習の内容

- cursesライブラリの簡単な使い方
 - 画面の任意位置に文字を表示
 - バッファリングのないキーボード入力
 - カーソルキーが押されたことを検知したい
- プログラミングプロジェクト課題の説明
 - テトリスゲーム - 統合プログラミング能力
 - 本課題の提出は単位取得の必須要件
 - 最終提出は8/10頃を予定 (演習の時間を有効活用)
- 授業アンケートを実施します。
- 演習 (長めに時間をとります)
 - 未提出の締め切り前必須課題と補足課題
 - プログラミングプロジェクト課題も開始
 - 演習の説明で本日提出分について説明があります
- 明日土曜日(7/15)は補講を実施します



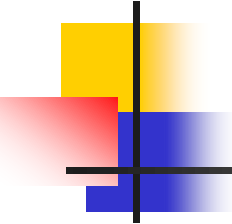
cursesライブラリとは

- 文字の画面への描画・キーボードからの入力を制御
 - 端末に依存しない方法
 - 端末の違いはライブラリがterminfoファイルを参照することで吸収
- 可能となること
 - 端末ウィンドウの操作
 - 端末ウィンドウの操作出力(文字の描画など)
 - 端末環境問い合わせ (画面の大きさ)
 - カラー処理
- cursesライブラリを使うには
 - `#include <curses.h>`
 - コンパイルオプションに`-lcurses`を追加
 - `gcc -o prog prog.c -lcurses`



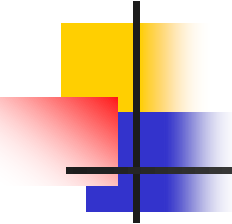
curses walk-through (1/4)

- curses ライブラリルーチンの初期化
 - 画面の初期化
 - `initscr()` を呼び出す
 - エコーなしの「一度に 1 文字」入力をON
 - `cbreak();`
 - `noecho;`
 - 矢印キーをON
 - `keypad(stdscr,TRUE);`
 - キー入力を直ちにcheckするよう設定変更
 - `timeout(0);`



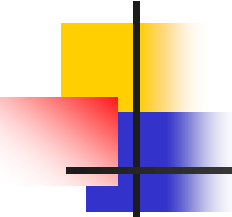
curses walk-through (2/4)

- curses ライブラリでの「ウィンドウ」
 - WINDOWS *と宣言される
 - コンソール(ターミナル)画面の全部分または一部分を表示する二次元の文字配列
 - デフォルトのウィンドウとして端末画面と同一サイズの stdscr が提供されている
 - 他のウィンドウは newwinを使って作成可
- 「ウィンドウ」使用終了時
 - `endwin();`を呼び出す



curses walk-through (3/4)

- デフォルトwindowへの文字の描画
 - `int mvaddch(int y, int x, const chtype ch);`
 - (x, y)に文字chを表示
 - y,xの順番で
 - `int mvaddstr(int y, int x, const char *str);`
 - (x, y)に文字列strを表示
- キーボードからの入力
 - `int getch();`
 - キーボードから一文字入力
 - キーを押せば直ちに入力される



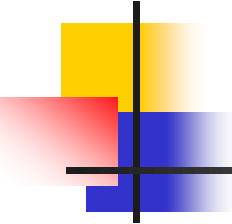
curses walk-through (4/4)

■ 特殊キーコード

- ← KEY_LEFT
- → KEY_RIGHT
- ↑ KEY_UP
- ↓ KEY_DOWN

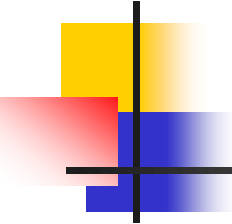
■ 画面サイズの取得

- 大域変数LINESに画面行数
- 大域変数COLSに画面横文字数
- LINES, COLSは<curses.h>の中で定義
- initscrを呼び出した時点で画面サイズの値が代入される。



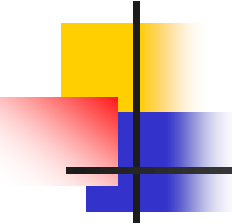
簡単なゲーム (1/8)

```
#include < curses.h>
void go();
int main(int argc, char **argv) {
    initscr();
    start_color();
    noecho();
    cbreak();
    curs_set(0);
    keypad(stdscr, TRUE);
    go();
    endwin();
    return 0;
}
```



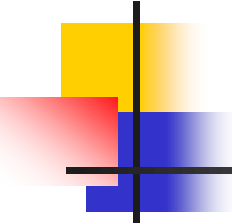
簡単なゲーム (2/8)

```
#define PATNUM 3
#define PATSIZEW 3
#define PATSIZEH 3
static char
    blockPattern[PATNUM][PATSIZEH][PATSIZEW] =
    {
        {{ '#', '#', '#'}, {' #', '#', '#'}, {' #', '#', '#'}},
        {{ ' ', '#', ' '}, {' ', '#', ' '}, {' ', '#', ' '}},
        {{ ' ', ' ', ' '}, {' #', '#', '#'}, {' ', ' ', ' '}}
    };
```



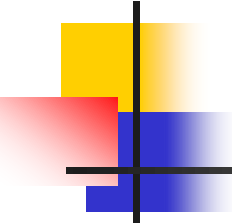
簡単なゲーム (3/8)

```
int collisionBottomWall(int blocX,  
    int blocY, int bottom, int  
    pattern)  
{  
    int flag = 0;  
    if( blocY+2 == bottom) {  
        flag = 1;  
    }  
    return flag;  
}
```



簡単なゲーム (4/8)

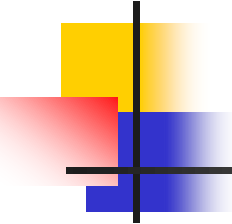
```
void go()
{
    int locX, locY;
    int locXdir = 1;
    int blocX, blocY;
    int ch;
    int delay=0;
    int waitCount = 20000;
    char msg[256];
    int pattern = 0;
    char buffer[256][256];
    int i, j;
    int ii, jj;
    blocX = COLS/4;
    blocY = 5;
    for(j=0;j<256;j++){
        for(i=0;i<256;i++){
            buffer[j][i]=' ';
        }
    }
    timeout(0);
```



簡単なゲーム (5/8)

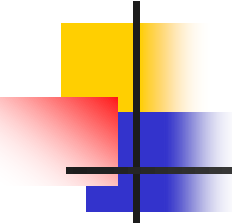
```
while ((ch=getch()) != 'Q') {
    mvaddstr(blocY,blocX,"    ");
    mvaddstr(blocY+1,blocX,"    ");
    mvaddstr(blocY+2,blocX,"    ");

    if(delay%waitCount==0) {
        blocY += 1;
    }
    if(blocX<0) {
        blocX =0;
        beep();
    }
    if(blocX>=COLS/2) {
        blocX =COLS/2-1;
        beep();
    }
    delay++;
}
```



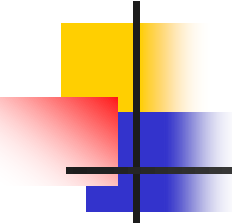
簡単なゲーム (6/8)

```
switch(ch) {
    case KEY_LEFT:
        blocX -=1;
        break;
    case KEY_RIGHT:
        blocX +=1;
        break;
    case ` `:
        if(pattern == 1 ) {
            pattern = 2;
        }else if(pattern == 2 ) {
            pattern = 1;
        }
        beep();
        break;
    default:
        break;
}
```



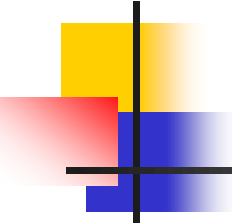
簡単なゲーム (7/8)

```
for (j=0; j<LINES; j++) {  
    for (i=0; i<COLS/2; i++) {  
        mvaddch(j, i, buffer[j][i] );  
    }  
}  
  
for (jj=0; jj<3; jj++) {  
    for (ii=0; ii<3; ii++) {  
        mvaddch(blocY+jj, blocX+ii,  
            blockPattern[pattern][jj][ii]);  
    }  
}
```



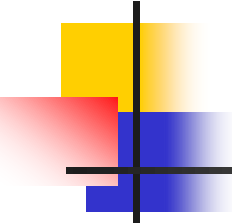
簡単なゲーム (8/8)

```
if( collisionBottomWall(blocX, blocY, LINES-3,
    pattern)){
    for(jj=0; jj<3; jj++){
        for(ii=0; ii<3; ii++){
            buffer[blocY+jj][blocX+ii] =
                blockPattern[pattern][jj][ii];
        }
    }
    beep();
    blocY = 5;
    pattern = (pattern+1)%PATNUM;
}
sprintf(msg, "X %03d Y %03d Pat %02d", blocX, blocY,
    pattern );
mvaddstr(2, COLS-20, msg );
}
}
```



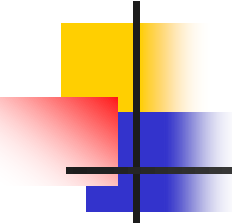
時間の計測

- `#include <time.h>`
`time_t time(time_t *t);`
- 説明
 - `time` は紀元 (1970年1月1日00:00:00 UTC) からの経過時間を秒単位で返す。もし `t` が `NULL` でなかったら返り値は `t` の指しているメモリにも格納される。
- 返り値
 - 成功した場合、紀元(the Epoch)からの経過秒数を返す。エラーの場合は `((time_t)-1)` を返し、`errno` を設定する。



使用例 - 経過時間を測る

- `#include <time.h>`
`main()`
{
 `time_t start;`
 `time_t current;`
 `time(&start);`
 `while(1){`
 `time(¤t);`
 `current -= start;`
 `printf( "Elapsed time = %d¥n", current);`
 }
}



プログラミングプロジェクト課題

- テトリスゲームを作成
- 各自のアイデアを取り込み、より充実したゲームを作成欲しい
- 構造体、線形リスト、双方向リスト、ファイルなど講義で学んだことを最大限生かしてプログラムを作成する
- 余力があればネットワーク対戦型を作成
- 本課題提出は単位取得の必須条件
- (通称: 夏休み課題)

仕様 (1/4)

■ 画面構成

- 次ページ参照
- 画面サイズに合わせて縦・横に配置できるブロックの数・各種情報の表示位置を適宜変化させる
- 画面サイズを変更しても描画エリアが変化しないプログラムは認められない
- ブロックは#記号、枠は|,-記号で表示
- 右側に経過時間、Level, 消した行数 次とその次のブロック, Score, High Score, High Scoreを出した人の名前を表示

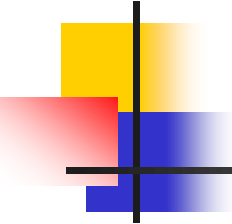
■ ブロック



ならびにこれらを90, 180, 270度回転させたもの

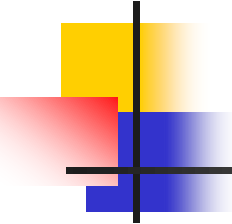
画面構成

```
0 1 2 3 4 5 6 7
01234567890123456789012345678901234567890123456789
00 *=====*
 1 | |
 2 | | Time AAAAA
 3 | | Level AAAAA
 4 | | Score AAAAA
 5 | | High Score AAAAA
 6 | | Name Def GHi
 7 | # Next Blocks
 8 | # First Second
 9 | # +---+ +---+
10 | | | | # |
 1 | | | | # |
 2 | | | | # |
 3 | | | |
 4 | | | |
 5 | | | |
 6 | | | |
 7 | | | |
 8 | | | |
 9 | ###
20 | ###
 1 | ###
 2 | ### # ###
 3 | ### # ###
 4 | #####
```



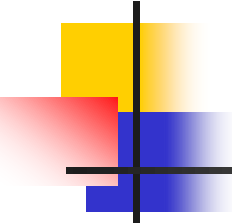
仕様 (2/4)

- ブロックの動き
 - 下方向に落ちブロックが積み上げる
 - ブロックが食い込むことのないように
 - 画面上部・中央付近から落ちる
 - スペースを押すと回転する
 - 下向き矢印を押すとブロックは強制落下
 - 一行そろうとその行は消滅。その上にあるブロックは一行さがる
 - ブロックがある高さまで積みあがるとゲームオーバー
 - 一定点数以上になればレベルアップ
 - レベルアップ毎に落下速度を上げる



仕様 (3/4)

- ゲームエリア
 - ゲームエリアはプログラム内で変更できるようにする
- Scoreの計算
 - 消された行数と落下速度から計算
- 経過時間
 - ゲーム開始からの経過時間を表示する
- 次に落下予定のブロックを右側に表示する



仕様 (4/4)

- ゲーム開始時

- 過去上位5位のHigh Score List (得点、氏名)をファイルから読み込む

- ゲーム終了時

- 上位5位以内であれば、名前を入力を促す
- 過去上位5位のHigh Score Listを表示
- 過去上位5位のHigh Score List (得点、氏名)をファイルに書き出す

