Alleytris

Love it or hate it, chances are you've played it. Tetris is one of those games that needs no introduction. Since it was created Tetris has been made for just about every range of electronics, from cell phones that fit in your pocket to the side of office buildings.

This variation of Tetris is only 4 columns wide. Playing the game in such a narrow space cleared lines are frequent and failing to pay attention to the next piece can be deadly. Getting 100 lines is a serious accomplishment. It contains a lot of the features found in most modern version of the game including quick drop, "infinite spin," and shadows that help you properly place pieces.

The initial goal of this was to simply write a Tetris game, but with code that was easy to modify with new pieces and game options. Consequently the data for the blocks is laid out so that the blocks were visible in the code, despite that making the indexing a bit weird. Once that was finished the first thing to do was test out how easy it was to make variations. Pieces were added,. The board was widened.

Finally a variant was made that was so simple and yet had a profound effect on the game play. Simply changing the WIDTH to 4 instead of 10 yields a game that many just can't stop playing. It was named Al-

```c
/* Alleytris.c linsting begins: */

#include <curses.h>
#include <stdlib.h>
#include <time.h>
#include <ctype.h>
#include <string.h>

#define WIDTH 4
#define HEIGHT 23
#define PIECES 7
#define BCOL(x) (x % (COLORS / 2)) + (COLORS / 2)

int col_width;

char blocks[4][PIECES][5] = /* [line][shape][col] */
{{".X..",".X..",".XX.","..X.",".X..",".X..","...."},
 {".XX.",".X..",".X..",".XX.",".XX.",".X..",".XX."},
 {".X..",".XX.",".X..",".X..","..X.",".X..",".XX."},
 {"....","....","....","....","....",".X..","...."}};
int rots[PIECES] = {4,4,4,2,2,2,1};
int **buf; /* Buffer for game screen */
WINDOW *pwin, *nwin, *iwin;

typedef struct {
  int x, y, rot, piece;
} BRICKTYPE;

char *instruct[10] =
{"Use arrow", "keys to play", "", "UP- Rotate", "DOWN- Faster", "SPACE- Drop",
 "S- Shadow", "N- Next", "" ,"Q to Quit"};
char *pausemsg[4] = {"PAUSED", " Press"," any"," key"};
char *againmsg[4] = {"Do you want", " to play", "  again", "   (Y/N)"};

void drawmsg(WINDOW *w, int y, int x, int lines, char **msg) {
  int c;

  for (c = 0; c < lines; c++)
    mvwprintw (w, y + c, x, msg[c]);
}

void wfillrect(WINDOW *w, int sy, int sx, int height, int width){
      int x, y;

      for (y = sy; y < (sy+height); y++) {
        wmove (w, y, sx);
        for (x = 0; x < width; x++) waddch (w, ' ');
      }
}

int rx (int r, int y, int x) { /* Used for piece rotation. */
  int n;

  n = (r % 2 ? y : x);
  if (r / 2) n = 2 - n;
  if (n < 0) n = 3;
  return n;
}

int ry (int r, int y, int x) { /* Used for piece rotation. */
  return rx ((r + 1) % 4, y, x);
}
```

```c
int clip (BRICKTYPE *b) { /*collision detection */
  int c, d;

  for (d = 0; d < 4; d++) for (c = 0; c < 4; c++)
    if (blocks[rx(b->rot % rots[b->piece], d, c)][b->piece][ry(b->rot % rots[b->piece], d, c)] != '.') {
      if ((d + b->y < 0) && ((c + b->x - 1 < 1) || (c + b->x - 1 > col_width)))
        return 1; /* Edge collisions */
      else if ((d + b->y >= 0) && (buf[d + b->y][c + b->x - 1])) return 1;
    }
  return 0;
}

void init () {
  int c, x, y, i;

  srand (time(NULL));
  initscr ();
  raw (); nodelay(stdscr,1); noecho(); curs_set(0); nonl(); keypad(stdscr,1);
  resize_term(25,80);
#ifdef PDCURSES
  PDC_set_title("Cymon's Games - Alleytris");
#endif
  start_color();
  for (c = 0; c < COLORS; c++)
    if (c == COLOR_WHITE) init_pair(c, COLOR_BLACK, c);
    else init_pair(c, COLOR_WHITE, c);
  /* Allocate the memory required for the board buffer here */
  buf = (int **)malloc(sizeof(int *) * (HEIGHT + 1));
  for (i = 0; i < HEIGHT + 1; i++) {
    buf[i] = (int*)malloc(sizeof(int) * (col_width + 2));
  }
  x = (COLS - col_width - 15) / 2; y = (LINES - HEIGHT) / 2;
  clear (); refresh ();
  pwin = newwin(HEIGHT + 1, col_width * 2 + 2, y, x);
  nwin = newwin(6, 10, y + 1, x + col_width * 2 + 5);
  iwin = newwin(15, 15, y + 8, x + col_width * 2 + 3);
  wattrset (iwin, COLOR_PAIR (COLORS / 2));
  wfillrect (iwin, 1, 1, 14, 13);
  box (iwin, 0,0);
  drawmsg (iwin, 4, 1, 10, instruct);
  wrefresh (iwin);
}

void drawplay (int lvl) { /* redraws the entire screen. */
  int backcol, nextcol, x, y;

  backcol = lvl % (COLORS / 2); nextcol = (lvl + 1) % (COLORS / 2);
  wattrset (pwin, COLOR_PAIR (backcol));
  wborder(pwin, 0, 0, ' ', 0, ACS_VLINE,  ACS_VLINE, 0, 0);
  for (y = 0; y < HEIGHT; y++) for (x = 1; x <= col_width; x++) {
    wattrset (pwin, COLOR_PAIR((buf[y][x])
      ? buf[y][x] % (COLORS / 2) + (COLORS / 2) : lvl % (COLORS / 2)));
    mvwaddstr (pwin, y, x * 2 - 1, "  ");
  }
  wattrset (nwin, COLOR_PAIR (nextcol));
  box (nwin,0,0);
  wfillrect (nwin, 1, 1, 4, 8);
  mvwaddstr (nwin, 0, 3, "NEXT");
  wrefresh (pwin); wrefresh (nwin);
}

int clearedlines (int lvl) {
```

leytris and the rest was history.

In Alleytris working with the next piece is vital. Frequently you can create gaps if you know that you'll be clearing the lines above them a moment later. Games tend to be short, but with many, many lines cleared very quickly.

You can, by command line parameter, make a game with different widths. Run with a width of 10 to play plain 'ol Tetirs. Run Alleytris with a width of 5 for a game with a different challenge.

Tetris is written by Joseph Larson inspired by a game by Alex Pajitnov.

/* Listing continued from previous page */
```
  int x, y, c, d, ret;

  ret = 0;
  for (y = 0; y < HEIGHT; y++) {
    c = 0;
    for (x = 1; x <= col_width; x++) if (buf[y][x]) c++;
    if (c == col_width) {
      ret++;
      for (d = y; d > 0; d--) for (x = 1; x <= col_width; x++)
        buf[d][x] = buf[d - 1][x];
      for (x = 1; x <= col_width; x++) buf[d][x] = 0;
      drawplay (lvl); wrefresh (pwin); beep (); napms(75);
    }
  }
  return ret;
}

void drawpiece (WINDOW *w, int y, int x, int rot, int piece, int color) {
  int c, d;

  wattrset (w, COLOR_PAIR (color));
  for (d = 0; d < 4; d++) for (c = 0; c < 4; c++)
    if (blocks[rx(rot % rots[piece], d, c)][piece][ry(rot % rots[piece], d, c)] !
= '.')
      mvwaddstr (w, d + y, (c + x - 1) * 2 - 1, "  "); /* 2 spaces */
}

int play () {
  BRICKTYPE brick, shadow;
  int level, next, lines, in, c, d, shadcol, nextop, shadop;
  double delay;
  clock_t start, check;

  level = 1; nextop = 1; shadop = 0; lines = 0;
  for (c = 0; c < HEIGHT + 1; c++) for (d = 0; d < col_width + 2; d++)
    buf[c][d] = (c < HEIGHT  && d > 0 && d < col_width + 1) ? 0 : 1;
  brick.piece = rand () % PIECES;
  brick.x = col_width / 2; brick.y = 0;
  do { /* Next piece */
    delay = 1.0 - level * .05;
    if (delay < .10) delay = 0.10 - (level - 20) * .01;
    if (delay < .06) delay = 0.6;
    if (shadop) brick.y = -2;
    next = rand () % PIECES;

    if (brick.piece % (COLORS / 2) == level % (COLORS / 2))
      shadcol = (brick.piece + 1) % (COLORS / 2);
    else shadcol = brick.piece % (COLORS / 2);

    drawplay (level);
    mvwprintw (iwin, 1, 2, "Level : %d", level);
    mvwprintw (iwin, 2, 2, "Lines : %d", lines);
    wrefresh (iwin);
    start = clock ();
    shadow.piece = brick.piece;
    do {
      shadow.x = brick.x; shadow.y = brick.y; shadow.rot = brick.rot;
      while (!clip(&shadow)) shadow.y++; shadow.y--;
      if (nextop) drawpiece (nwin, 1, 2, brick.rot, next, BCOL(next));
      if (shadop) drawpiece (pwin, shadow.y, shadow.x,
        shadow.rot, shadow.piece, shadcol);
      drawpiece (pwin, brick.y, brick.x,
        brick.rot, brick.piece, BCOL(brick.piece));
```

/* Listing continued from previous page */

```c
      wrefresh (pwin); wrefresh (nwin);

      do {
        in = getch ();
        check = clock ();
      } while ((in == ERR)&&((double)(check - start) / CLOCKS_PER_SEC < delay));

      if (nextop)
        drawpiece (nwin, 1, 2, brick.rot, next, (level + 1) % (COLORS / 2));
      if (shadop)
        drawpiece (pwin, shadow.y, shadow.x, shadow.rot, shadow.piece,
          level % (COLORS / 2));
      drawpiece (pwin, brick.y, brick.x, brick.rot, brick.piece,
        level % (COLORS / 2));

      if ((double)(check - start) / CLOCKS_PER_SEC > delay) {
        brick.y++; start = clock ();
      }
      else switch (in) {
        case KEY_RIGHT : brick.x++;
                         if (clip(&brick)) brick.x--;
                         else start = clock ();
                         break;
        case KEY_LEFT  : brick.x--;
                         if (clip(&brick)) brick.x++;
                         else start = clock ();
                         break;
        case KEY_UP    : brick.rot++; brick.rot %= 4;
                         if (clip(&brick)) {
                           brick.x--;
                            if (clip (&brick)) {
                              brick.x += 2;
                              if (clip (&brick)) {
                                brick.x--; brick.rot += 3; brick.rot %= 4;
                              } else start = clock ();
                            } else start = clock ();
                         } else start = clock ();
                         break;
        case KEY_DOWN  : brick.y++;
                         start = clock ();
                         break;
        case ' '       : while (!clip(&brick)) brick.y++;
                         brick.y--;
                         start = clock ();
                         break;
        case 'n' :
        case 'N' : nextop = !nextop; break;
        case 's' :
        case 'S' : shadop = !shadop; break;
        case 'p' :
        case 'P' : wattrset (pwin, COLOR_PAIR(level % (COLORS / 2)));
                   wfillrect (pwin, 0, 1, HEIGHT, col_width * 2);
                   drawmsg (pwin, HEIGHT / 2, 2, 4, pausemsg);
                   wrefresh (pwin);
                   while (getch() == ERR) {}
                   drawplay (level);
                   wrefresh (pwin);
                   break;
        case 'q' :
        case 'Q' : return (lines);
      }
    } while (!clip (&brick)); /* end of brickfall */
    brick.y--;
```

/* Listing continued from previous page */

```c
      if (brick.y < 0) return (lines);
      for (d = 0; d < 4; d++) for (c = 0; c < 4; c++) /* commit piece to buffer */
      if (blocks[rx(brick.rot % rots[brick.piece], d, c)][brick.piece]
        [ry(brick.rot % rots[brick.piece], d, c)] != '.')
        buf[d + brick.y][c + brick.x - 1] = BCOL(brick.piece);
      drawplay (level);
      lines += clearedlines (level);
      level = lines / 10 + 1;
      brick.piece = next; brick.x = col_width / 2; brick.y = 0;
      next = rand () % PIECES;
    } while (!clip (&brick)); /* end of game */
    drawpiece (pwin, brick.y, brick.x, brick.rot, brick.piece, BCOL(brick.piece));
    wrefresh (pwin);
    return (lines);
}

int again (int hs) {
  int ch;
  WINDOW *againwin;

  againwin = newwin(11, 13, (LINES / 2) - 6, (COLS / 2) - 4);
  wattrset (againwin, COLOR_PAIR (COLORS / 2 - 1));
  wfillrect (againwin, 0, 0, 11, 13);
  mvwprintw (againwin, 1, 1, "High Score:");
  mvwprintw (againwin, 3, 3, " %d lines", hs);
  drawmsg (againwin, 6, 1, 4, againmsg);
  wrefresh (againwin);
  wclear (againwin);
  do {ch = getch();}
  while ((tolower(ch) != 'q')&&(tolower(ch) != 'n')&&(tolower(ch) != 'y'));
  wrefresh (againwin);
  wattrset (iwin, COLOR_PAIR (COLORS / 2));
  wfillrect (iwin, 1, 1, 14, 13);
  box (iwin, 0,0);
  drawmsg (iwin, 4, 1, 10, instruct);
  wrefresh (iwin);
  delwin (againwin);
  if (tolower(ch) == 'y') return 1;
  else return 0;
}

int main (int argc, char **argv) {
  int score, highscore= 0;
  int i;
  col_width = WIDTH;

  if (argc == 2) col_width = atoi(argv[1]);
  if (col_width > 20) col_width = WIDTH;
  if (col_width <= 0) col_width = WIDTH;

  init();
  do {
    score = play();
    if (score > highscore) highscore = score;
  } while (again (highscore));
  endwin();

  for (i = 0; i < col_width + 2; i++) {
      free(buf[i]);
  }
  free(buf);

  return 0;
}
```