Nibbles

Nibbles is another game that requires no introduction. Navigate a snake around a space eating the blue dots avoiding the walls and yourself. The more blue dots you eat the longer your tail gets making the task of avoiding yourself more and more difficult. As you get longer the game becomes and exercise in tail management.

This version of the game is extremely basic and can act as an effective base for many variations. An ambitious programmer could use this code and change the display so the snake has a head and tail, add levels with obstacles to avoid, change the number of blue squares at a time, or even make a two player version.

Nibbles is written by Nils Magnus Englund.

// config.h listing begins:

```
#ifndef __CONFIG_H_
#define __CONFIG_H_
// Size of the window used by nibbles
#define WINDOWX         80
#define WINDOWY         25
// Number of microseconds between each tick
#define TICKSPEED       100000
// The length of a new worm, don't set this too high!
#define STARTLENGTH     5
// How many ticks the worm will continue to grow after start
#define STARTELENGTH    0
// How much the worm grows when it eats
#define WORMGROWTH      5
// Color settings - see the ncurses manpage for other colors
#define WORMCOLOR       COLOR_YELLOW
#define LINECOLOR       COLOR_WHITE
#define STATUSCOLOR     COLOR_WHITE
#define DOTCOLOR        COLOR_BLUE
#define BACKGROUND      COLOR_BLACK
// How many moves we make room for in the history array - the higher, the
// better. When HISTORYSIZE ticks have passed, the last bytes (number equal
// to the length of the worm) are copied to the start of the array.
#define HISTORYSIZE       8192
// Don't edit below this line!
#define WORMCOLORPAIR   1
#define LINECOLORPAIR   2
#define STATUSCOLORPAIR 3
#define DOTCOLORPAIR    4

#define LEFT          1
#define RIGHT         2
#define UP            3
#define DOWN          4

#ifndef true
#define true          1
#endif
#ifndef false
#define false         0
#endif

#define GRIDX         ((WINDOWX-2)>>1)
#define GRIDY         (WINDOWY-4)

#define Main          0
#define Grid          1
#define Status        2

#define VERSION       1.2

#endif /* __CONFIG_H__ */
```

// misc.h listing begins:

```
#ifndef __MISC_H_
#define __MISC_H_
// seed the RNG using microseconds from gettimeofday()
void randomize (void);
// generate a random integer, min <= X <= max
int randint (int min, int max);
#endif /* __MISC_H_ */
```

// Listings continued on next page...

// misc.c listing begins:

```
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>

#include "misc.h"

void randomize (void)
{
#ifdef UNIX
   struct timeval *tv  = (struct timeval  *) malloc(sizeof(struct timeval ));
   struct timezone *tz = (struct timezone *) malloc(sizeof(struct timezone));
   gettimeofday(tv,tz);
   srand(tv->tv_usec);
   free(tv);
   free(tz);
#else
   srand (time(NULL));
#endif
}

int randint (int min, int max)
{
   return (min + (int) ((max+1) * (rand() / (RAND_MAX+1.0))));
}
```

// screen.h listing begins:

```
#ifndef __SCREEN_H_
#define __SCREEN_H_

void endgraphics (void);
void startgraphics (WINDOW **w);
void drawblock (WINDOW **w, int x, int y);
void clearblock (WINDOW **w, int x, int y);
void newdot (WINDOW **w, bool grid[GRIDX][GRIDY], int *x, int *y);

#endif /* __SCREEN_H_ */
```

// screen.c listing begins:

```
#include <stdio.h>
#include <stdlib.h>
#include <curses.h>

#include "config.h"

#include "misc.h"
#include "screen.h"

void endgraphics (void)
{
  endwin();
}

void startgraphics (WINDOW **w)
{
  // to put the nibbles-window in the center of the screen
  int    offsetx,offsety;
  int    loop,lx,ly;


  initscr();

  if ((COLS < WINDOWX) || (LINES < WINDOWY)) {
```

// Listing continued on next page...

// Listing continued from previous page

```
    endwin();
    fprintf(stderr,"A terminal with at least %d lines and %d columns is re-
quired.\n",WINDOWY,WINDOWX);
    exit(1);
  }

  offsetx = (COLS - WINDOWX) >> 1;
  offsety = (LINES - WINDOWY) >> 1;

  w[Main] = newwin(WINDOWY,WINDOWX,offsety,offsetx);
  w[Grid] = newwin(GRIDY,GRIDX*2,offsety+1,offsetx+1);
  w[Status] = newwin(1,WINDOWX-2,offsety+WINDOWY-2,offsetx+1);

  keypad(w[Grid], TRUE);
  nonl();
  cbreak();
  noecho();
  nodelay(w[Grid], TRUE);
  curs_set(FALSE);

  if (!has_colors()) {
    endwin();
    fprintf(stderr,"A terminal with color support is required.\n");
    exit(1);
  }

  start_color();

  // configure colors
  init_pair(WORMCOLORPAIR,WORMCOLOR,BACKGROUND);
  init_pair(LINECOLORPAIR,LINECOLOR,BACKGROUND);
  init_pair(STATUSCOLORPAIR,STATUSCOLOR,BACKGROUND);
  init_pair(DOTCOLORPAIR,DOTCOLOR,BACKGROUND);

  // draw pretty lines in w[Main]
  wattrset(w[Main],COLOR_PAIR(LINECOLORPAIR)|A_BOLD);
  mvwaddch(w[Main],0,0,ACS_ULCORNER);
  mvwaddch(w[Main],0,WINDOWX-1,ACS_URCORNER);
  mvwaddch(w[Main],WINDOWY-1,0,ACS_LLCORNER);
  mvwaddch(w[Main],WINDOWY-1,WINDOWX-1,ACS_LRCORNER);
  for (loop=1;loop<WINDOWY-1;loop++) {
    mvwaddch(w[Main],loop,0,ACS_VLINE);
    mvwaddch(w[Main],loop,WINDOWX-1,ACS_VLINE);
  }
  for (loop=1;loop<WINDOWX-1;loop++) {
    mvwaddch(w[Main],0,loop,ACS_HLINE);
    mvwaddch(w[Main],WINDOWY-3,loop,ACS_HLINE);
    mvwaddch(w[Main],WINDOWY-1,loop,ACS_HLINE);
  }
  mvwaddch(w[Main],WINDOWY-3,0,ACS_LTEE);
  mvwaddch(w[Main],WINDOWY-3,WINDOWX-1,ACS_RTEE);
  wrefresh(w[Main]);

  // clear the status screen
  wattrset(w[Status],COLOR_PAIR(STATUSCOLORPAIR)|A_BOLD);
  for (loop=0;loop<=GRIDX;loop++) {
    clearblock(w,loop,0);
  }
  wrefresh(w[Status]);

  // clear the grid screen
  wattrset(w[Grid],COLOR_PAIR(WORMCOLORPAIR)|A_BOLD);
  for (lx=0;lx<=GRIDX;lx++) {
```

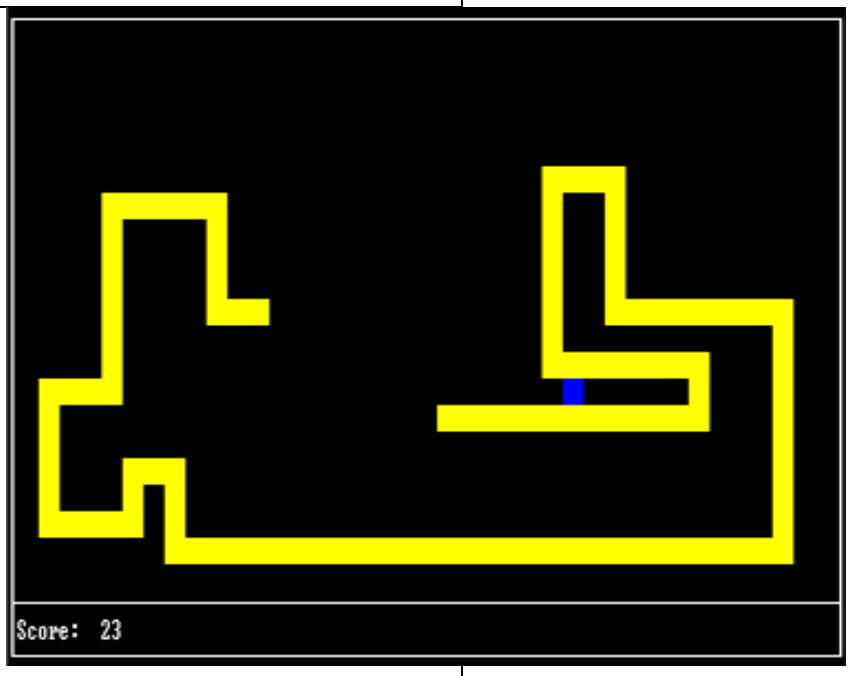
// Listing continued on next page...

// Listing continued from previous page

```
    for (ly=0;ly<=GRIDY;ly++) {
      clearblock(w,lx,ly);
    }
  }
#ifdef PDCURSES
  PDC_set_title("Nibbles - Cymon's Games");
#endif
  wrefresh(w[Grid]);
}

void drawblock (WINDOW **w, int x, int y)
{
  mvwaddch(w[Grid],y,x*2,ACS_BLOCK);
  mvwaddch(w[Grid],y,x*2+1,ACS_BLOCK);
}

void clearblock (WINDOW **w, int x, int y)
{
  mvwaddch(w[Grid],y,x*2,' ');
  mvwaddch(w[Grid],y,x*2+1,' ');
}

void newdot (WINDOW **w, bool grid[GRIDX][GRIDY], int *x, int *y)
{
  do {
    *x = randint(0,GRIDX-1);
    *y = randint(0,GRIDY-1);
  } while (grid[*x][*y]);
  wattrset(w[Grid],COLOR_PAIR(DOTCOLORPAIR)|A_BOLD);
  drawblock(w,*x,*y);
  wattrset(w[Grid],COLOR_PAIR(WORMCOLORPAIR)|A_BOLD);
}
```

// screen.c listing begins:

```
#include <stdio.h>
#include <stdlib.h>
#include <curses.h>
#include <signal.h>
#include <sys/time.h>
#include <unistd.h>
#include <string.h>

#include "config.h"
#include "misc.h"
#include "screen.h"

#ifdef PDCURSES
#define usleep(x) napms((x)/1000)
#endif

int quit (int sig)
{
  endgraphics();
  exit(0);
}

int main (int argc, char *argv[])
{
  // The tree windows (Main,Grid,Status in that order) used by ncurses
  WINDOW  *w [3];

  // The grid, to make it easy to tell where the worm is
```



// Listing continued on next page...

// Listing continued from previous page

```
  bool grid [GRIDX] [GRIDY];

  // The worm history
  short int  hx [HISTORYSIZE];
  short int  hy [HISTORYSIZE];

  // Worm variables
  int      length = STARTLENGTH; // the length of the worm
  int      extralength = STARTELENGTH; // how much the worm is going to grow
  int      curx,cury; // current position of the worm
  int      dir = RIGHT; // direction the worm is facing
  int      lastdir = RIGHT; // last direction
  int      score = 0;
  bool    paused = false;


  int      dotx,doty; // position of the dot (worm food)

  int      tick; // current tick

  // User input
  int      ch;

  // Initialize ncurses and the windows
  startgraphics(w);

  // Seed the random number generator
  randomize();

  // clear the grid
  memset(grid,0,sizeof(grid));

  // Initialize and draw the worm
  curx = (GRIDX+length) >> 1;
  cury = GRIDY >> 1;
  for (tick=0;tick<length;tick++) {
    hx[tick] = curx - (length - tick) + 1;
    hy[tick] = cury;
    grid[curx-(length-tick)+1][cury] = true;
    drawblock(w,curx-(length-tick)+1,cury);
  }

  // Initialize and draw the dot (worm food)
  newdot(w,grid,&dotx,&doty);

  // Main loop
  while (1) {
    lastdir = dir;

    if (paused) {
      ch = wgetch(w[Grid]);
      while (ch != ERR) {
        switch (ch) {
          case 'p':
            paused = false;
            mvwprintw(w[Status],0,WINDOWX - 2 /* because of the border lines */ -
7 /* strlen("PAUSED ") */, "      ");
            break;
          case 'q':
            endgraphics();
            printf("Quitter! Your score was %d.\n",score);
            exit(0);
            break;
```

// Listing continued on next page...

// Listing continued from previous page

```
        }
        ch = wgetch(w[Grid]);
      }
      mvwprintw(w[Status],0,0,"Score:  %d ",score);
      mvwprintw(w[Status],0,WINDOWX - 2 /* because of the border lines */ - 7 /*
strlen("PAUSED ") */, "Paused");
      wrefresh(w[Status]);
      usleep(TICKSPEED);
      continue;
    }

    // catch user input
    ch = wgetch(w[Grid]);
    while (ch != ERR) {
      switch (ch) {
        case KEY_LEFT:
        case 'h': // accept vi-style directions as well
          if ((lastdir == UP) || (lastdir == DOWN)) {
            dir = LEFT;
          }
          break;
        case KEY_RIGHT:
        case 'l':
          if ((lastdir == UP) || (lastdir == DOWN)) {
            dir = RIGHT;
          }
          break;
        case KEY_UP:
        case 'k':
          if ((lastdir == LEFT) || (lastdir == RIGHT)) {
            dir = UP;
          }
          break;
        case KEY_DOWN:
        case 'j':
          if ((lastdir == LEFT) || (lastdir == RIGHT)) {
            dir = DOWN;
          }
          break;
        case 'p':
          paused = true;
          break;
        case 'q':
          endgraphics();
          printf("Quitter! Your score was %d.\n",score);
          exit(0);
          break;
      }
      ch = wgetch(w[Grid]);
    }

    // modify the new position based on the direction
    switch (dir) {
      case LEFT:  curx--; break;
      case RIGHT:  curx++; break;
      case UP:  cury--; break;
      case DOWN:  cury++; break;
    }

    // check whether or not the worm has crashed in a wall
    if ((curx<0) || (curx>=GRIDX) || (cury<0) || (cury>=GRIDY)) {
      endgraphics();
      printf("You crashed in a wall. Your score was %d.\n",score);
```

// Listing continued on next page...

// Listing continued from previous page

```
      exit(0);
    }

    // check whether or not the worm has crashed with itself
    if (grid[curx][cury]) {
      endgraphics();
      printf("You crashed with yourself. Your score was %d.\n",score);
      exit(0);
    }

    // the worm is still on the grid, and it seems healty :)

    drawblock(w,curx,cury); // let's draw the new block
    grid[curx][cury] = true; // and update the grid

    // update the history array
    hx[tick] = curx;
    hy[tick] = cury;

    // has the worm eaten food?
    if ((dotx == curx) && (doty == cury)) {
      score++;
      extralength += WORMGROWTH;
      newdot(w,grid,&dotx,&doty);
    }

    // if the worm still has some growing to do..
    if (extralength > 0) {
      extralength--;
      length++;
    } else {
      // otherwise, clear the last block
      clearblock(w,hx[tick-length],hy[tick-length]);
      grid[hx[tick-length]][hy[tick-length]] = 0;
    }

    wrefresh(w[Grid]);

    // finally, update the status line
    mvwprintw(w[Status],0,0,"Score:  %d",score);
    mvwprintw(w[Status],0,GRIDX - 2 /* because of the border lines */ - 7 /*
strlen("PAUSED ") */,"       ");
    wrefresh(w[Status]);

    usleep(TICKSPEED);
    tick++;

    // if the history array is full, copy the length of the worm
    // to the beginning of the array and decrease the tick counter
    if (tick == HISTORYSIZE) {
      memcpy(hx, hx+HISTORYSIZE-length, length * sizeof(short int));
      memcpy(hy, hy+HISTORYSIZE-length, length * sizeof(short int));
      tick = length;
    }

    // ready for next tick!
  }
  endgraphics();
}
```