Circuit Puzzle

Use the number pad keys to rotate cables. Connect all cables to the power sources to go up a level. Power wraps between the edges (leftmost cable is connected to rightmost). You're timed, so be quick.

Circuit Puzzle is written by Jakub Wasilewski revised from his MinorHack entry at July 7th, 2007 at 7:00 pm UTC.

// circuitpuzzle.cpp listing begins:

```
#include <allegro.h>
#include <cmath>

enum {UP = 1, DOWN = 2, LEFT = 4, RIGHT = 8, LIT = 16, SRC = 32};
BITMAP *tiles[64];

BITMAP *drawTile(int face) {
  int backCol = (!(face & LIT)) ? makecol(200, 200, 200)
   : makecol(210, 210, 200);
  int frontCol = (!(face & LIT)) ? makecol(0, 0, 0)
   : makecol(255, 255, 255);

  BITMAP *bmp = create_bitmap(64, 64);
  rectfill(bmp, 0, 0, 64, 64, backCol);
  if (face & UP) rectfill(bmp, 28, 0, 36, 36, frontCol);
  if (face & DOWN) rectfill(bmp, 28, 28, 36, 64, frontCol);
  if (face & LEFT) rectfill(bmp, 0, 28, 36, 36, frontCol);
  if (face & RIGHT) rectfill(bmp, 28, 28, 64, 36, frontCol);
  if (face & SRC) rectfill(bmp, 20, 20, 44, 44, makecol(255, 0, 0));
  return bmp;
}

volatile int ticks = 0;
void tick() {
  ticks++;
}

struct Block {
  int x, y;
  int rot, face;

  int rotStage;
  int rotTarget, rotDir;

  bool lit;

  Block(int x, int y, int face)
    : x(x), y(y), face(face)
  {
    rot = 0;
    rotStage = 0;
    rotTarget = 0;
    rotDir = 0;
  }

  void update() {
    if (rotDir)
      rotStage += rotDir * 4;
    if ((rotTarget - rotStage) * rotDir < 0) {
      rotStage = rotTarget;
      rot += rotDir;
      rot %= 4;
      rotDir = 0;
    }
  }

  void draw(BITMAP *bmp) {
    int sx = 320 - 3 * 32 + x * 64;
    int sy = 240 - 3 * 32 + y * 64;

    rotate_sprite(bmp, tiles[face + (lit ? LIT : 0)], sx, sy, itofix(rotStage));
```

// Listing continued on next page…

// Listing continued from previous page

```
  }

  void right() {
    if (rotDir == -1) return;
    rotTarget += 64;
    rotDir = 1;
  }

  void left() {
    if (rotDir == 1) return;
    rotTarget -= 64;
    rotDir = -1;
  }
};

bool has(Block *b, int dir) {
  const int dirs[] = {UP, RIGHT, DOWN, LEFT};

  int act;
  if (dir == UP)
    act = dirs[(104 - b->rot) % 4];
  if (dir == RIGHT)
    act = dirs[(105 - b->rot) % 4];
  if (dir == DOWN)
    act = dirs[(106 - b->rot) % 4];
  if (dir == LEFT)
    act = dirs[(107 - b->rot) % 4];

  return (b->face & act);
}

void fillFrom (Block **b, int i) {
  if (b[i]->lit) return;

  b[i]->lit = true;

  const int dirs[] = {UP, RIGHT, DOWN, LEFT};
  int above = (i + 3) % 9;
  int below = (i + 6) % 9;
  int left = (i / 3) * 3 + (i % 3 + 2) % 3;
  int right = (i / 3) * 3 + (i % 3 + 1) % 3;

  if (has(b[i], UP) && (has(b[above], DOWN)))
    fillFrom(b, above);
  if (has(b[i], DOWN) && (has(b[below], UP)))
    fillFrom(b, below);
  if (has(b[i], LEFT) && (has(b[left], RIGHT)))
    fillFrom(b, left);
  if (has(b[i], RIGHT) && (has(b[right], LEFT)))
    fillFrom(b, right);
}

void fill (Block **b) {
  int src = -1;
  for (int i = 0; i < 9; i++)
  {
    b[i]->lit = false;
    if (b[i]->face & SRC)
    {
      src = i;
    }
  }
```

// Listing continued on next page…

// Listing continued from previous page

```
  fillFrom(b, src);
}

BITMAP *buf;

void init() {
  allegro_init();

  set_color_depth(32);
  set_gfx_mode(GFX_AUTODETECT_WINDOWED, 640, 480, 0, 0);

  for (int i = 0; i < 64; i++)
    tiles[i] = drawTile(i);

  install_keyboard();
  install_int_ex(tick, BPS_TO_TIMER(100));

  buf = create_bitmap(640, 480);
}

int main() {
  init();

  Block *b[9];
  int level = 0;
  float time;
  bool win;
  do {
    level++;

    time = 25.0 - 15.0 * (std::pow(level * 0.1, 2.0));
    if (level == 1)
      time = 100.0;
    for (int y = 0; y < 3; y++)
      for (int x = 0; x < 3; x++) {
        int r;
        do {
          r = rand () % 15 + 1;
        } while (!(r & (r-1)));

        b[6 - y * 3 + x] = new Block(x, y, r);
      }
    b[4]->face += SRC;

    bool end = false;
    int k;
    win = false;

    while(!end) {
      while (ticks > 0) {
        while (keypressed()) {
          k = readkey() >> 8;
          end = (k == KEY_ESC);
          int num = k - KEY_1_PAD;
          if ((num >= 0)&&(num <= 8)) {
            b[num]->right();
          }
        }

        time -= 0.01;
        for (int i = 0; i < 9; i++)
          b[i]->update();
        ticks--;
```

// Listing continued on next page…

// Listing continued from previous page

```
        fill(b);
        win = true;
        for (int i = 0; i < 9; i++)
          win &= b[i]->lit;

        if (win)
          end = true;

        if (time < 0)
          end = true;
      }

      clear_bitmap(buf);
      for (int i = 0; i < 9; i++)
        b[i]->draw(buf);
      textprintf_ex(buf, font, 8, 8, makecol(255, 255, 255), -1,
        "LEVEL: %02d", level);
      textprintf_right_ex(buf, font, 632, 8, makecol(255, 0, 0), -1,
        "TIME: %0.2f", time);
      textprintf_centre_ex(buf, font, 320, 40, makecol(80, 80, 80), -1,
        "Press numpad keys to rotate cables.");
      textprintf_centre_ex(buf, font, 320, 50, makecol(80, 80, 80), -1,
        "Connect all cables to the power sources to go up a level.");
      textprintf_centre_ex(buf, font, 320, 60, makecol(80, 80, 80), -1,
    "Power wraps between the edges (leftmost cable is connected to rightmost)");
      blit(buf, screen, 0, 0, 0, 0, 640, 480);
    }

    if (win)
    {
      clear_bitmap(screen);
      textprintf_centre_ex(screen, font, 320, 100, makecol(0, 255, 0), -1,
        "N E X T     L E V E L");
      textprintf_centre_ex(screen, font, 320, 120, makecol(255, 255, 0), -1,
        "Press SPACE to continue.", level);

      int kk;
      do
      { kk = readkey() >> 8; }
      while (kk != KEY_SPACE);
    }
  } while (win);

  clear_bitmap(screen);
  textprintf_centre_ex(screen, font, 320, 100, makecol(255, 0, 0), -1,
    "G A M E    O V E R");
  textprintf_centre_ex(screen, font, 320, 120, makecol(255, 255, 0), -1,
    "Final level: %d", level);
  textprintf_centre_ex(screen, font, 320, 140, makecol(80, 80, 80), -1,
    "Press SPACE to exit.", level);

      int kk;
      Do { kk = readkey() >> 8; }
      while (kk != KEY_SPACE);
  return 0;
}
END_OF_MAIN();
```