Numbrix Solver and Generator

If you enjoy Maryln Vos Savant's Numbrix puzzles but aren't satisfied with a measly one a day then this program is for you. Generate unique numbrix puzzles on the fly for a moment's distraction or generate a whole book of them to take with you. If you're stuck on a puzzle in the paper and can't wait for the solution to be printed the next day you can also solve puzzles with this.

If you've never played a Numbrix now's your chance to play this simple puzzle that is sure to make you feel like a genius.

What is a Numbrix? Numbrix is a new puzzle created by Maryln Vos Savant and run weekly in the Parade magazine. Numbrix are simple and satisfying and require no arithmetic skills beyond needing to count.

In a numbrix the numbers starting at 1 count upward making a winding path through the grid, touching every square until they entire grid is filled. So for a 9x9 numbrix the numbers go from 1 to 81. From a square you can go 1 step in only the coordinal directions; up, down, left, or right. Diagonals are not allowed. At the start you only see only some of the squares have numbers. It is your job to fill in the rest.

With this program you can either type in your options with the keyboard or click with the mouse on any screen.

// numbrixgen.h listing begins:

```
// Numbrix Generator
// By Joe Larson for Cymon's Games
// Based on the Numbrix puzzle created by Maryln Vos Savant
#ifndef NUMBRIXGEN_H_INCLUDED
#define NUMBRIXGEN_H_INCLUDED

class NpathGen { // Unicursal grid filling path generator for Numbrix Generation
  public:
  // constructors
  NpathGen (int sizex = 9, int sizey = 0, int randseed = time(NULL));
  ~NpathGen ();
  // accesors
  int getXSize () {return maxx;}
  int getYSize () {return maxy;}
  int getXY(int x, int y) {return board[x + y * maxx];}
  void setSeed (int s) {seed = s;}
  int getSeed () {return seed;}
  void setMask (int m) {mask = m;}
  int getMask () {return mask;}
  // Misc. Methods

  private:
  int maxx, maxy, start, end, seed, mask;
  int step[5];
  int* board;
  clock_t timer, secs;
  void generate ();
  int change();
};

class NSolver {
  public:
  NSolver (int x, int y, int** grid);
  NSolver (int dim, int** grid) {NSolver (dim, dim, grid);}
  ~NSolver ();
  int getSolutionXY (int x, int y, unsigned int n);
  int getSolutionXY (int x, int y) {return getSolutionXY (x, y, 0);}
  int getNumSolutions () {return solutions.size();}

  private:
  std::vector<int**> solutions;
  int sizex, sizey;
  int** puzzle;
  int* puzmask;
  int findrun();
  void wiggle (int x, int y, int num, signed int dir);
  void check (int x, int y, int num, signed int dir);
};

#endif // NUMBRIXGEN_H_INCLUDED
```

// numbrixgen.cpp listing begins:

```
#include <vector>
#include <cstdlib>
#include <ctime>
#include "numbrixgen.h"

#define NUMBENDS 8
```

// Listing continued on next page...

// Listing continued from previous page

```
enum {NONE, UP, LEFT, DOWN, RIGHT, END};

int bends[NUMBENDS][2][2][2] = {      /* [type][before/after][y][x]*/
{{{ NONE, DOWN},{ NONE,  END}},{{ DOWN, LEFT},{RIGHT,  END}}},
{{{ NONE, NONE},{  END, LEFT}},{{ DOWN, LEFT},{  END,   UP}}},
{{{  END, NONE},{   UP, NONE}},{{  END, LEFT},{RIGHT,   UP}}},
{{{RIGHT,  END},{ NONE, NONE}},{{ DOWN,  END},{RIGHT,   UP}}},
{{{ NONE,  END},{ NONE,   UP}},{{RIGHT,  END},{   UP, LEFT}}},
{{{ NONE, NONE},{RIGHT,  END}},{{RIGHT, DOWN},{   UP,  END}}},
{{{ DOWN, NONE},{  END, NONE}},{{RIGHT, DOWN},{  END, LEFT}}},
{{{  END, LEFT},{ NONE, NONE}},{{  END, DOWN},{   UP, LEFT}}}
};

NpathGen::NpathGen (int sizex, int sizey, int randseed) {
  maxx = sizex;
  if (sizey) maxy = sizey;
  else maxy = sizex;
  seed = randseed;
  board = new int[maxx * maxy];
  step[0] = 0; step[1] = -maxx; step[2] = -1; step[3] = maxx; step[4] = 1;
  generate();
}

NpathGen::~NpathGen () {
  delete[] board;
}

void NpathGen::generate () {
  srand (seed);
  do {
    time(&secs);
    for (int c = 0; c < maxx * maxy; c++) board[c] = NONE; // Clear the board
    start = rand() % (maxx * maxy);
    do {
      switch (rand() % 4) {
        case 0 : if (start - maxx >= 0) {
                   board[start] = UP;
                   end = start - maxx;
                 } break;
        case 1 : if (start % maxx) {
                   board[start] = LEFT;
                   end = start - 1;
                 } break;
        case 2 : if ((start + maxx) < (maxx * maxx)) {
                   board[start] = DOWN;
                   end = start + maxx;
                 } break;
        case 3 : if ((start + 1) % maxx) {
                   board[start] = RIGHT;
                   end = start + 1;
                 } break;
      }
    } while (board[start] == NONE);
    board[end] = END;
  } while (change() == 0);
                // now the puzzle is generated, but it needs to be changed from
  int c = 1;                                    // direction vectors to numbers
  int xy = start;
  int next = start + step[board[start]];
  while (xy != end) {
    board[xy] = c++;
    xy = next;
    next = xy + step[board[xy]];
```

// Listing continued on next page...

At the main menu you are given 3 options:

Play a random puzzle: With this option the computer will create a randomly generated numbrix for you to solve interactively. You will be timed so be as fast as you can. If you liked a particular puzzle you can get it by going to the directory where this program is run from and getting the last.txt file. Move it or rename it to prevent it from being overwritten the next time you play.

Solve a Numbrix: With this option you can type in a numbrix that you find in the paper or one generated in a batch with option 3 and then either solve it yourself or the computer can solve it for you.

Make a batch of puzzles: With this option you can generate a book full of numbrix puzzles at once. The initial settings will be the same for all, but keep in mind that a “random” type puzzle will be different with every one. Since time to generate varies and can take quite a while it may be a good idea to let the program run for a while. Files will be named by taking the base name you specify and adding a number after it. You can retrieve generated puzzles in the directory where this program is run.

At the time of this writing this program can make puzzles that are exactly like the numbrix puzzles in print, but allows for some significant departures from the traditional formula:

- Traditional numbrix puzzles are either 8x8 or 9x9. This program allows the creation of puzzles sized from 5x5 to 12x12 (for puzzles play in the game) or 30x30 (for puzzles generated in the batch command) and allows rectangular dimensions.
- Numbrix layout types in print have only so far been the "traditional", donut, strawberry, and strawberry donut. This program adds an option for random layout. The random layout does maintain a quad symmetry for aesthetic purposes. (This is no longer true, but these remain the most common types seen.)
- Occasionally a generated puzzle will have more than one solution. In that case a square will be moved to give it a unique solution, and quad symmetry will also be maintained. This technically breaks the layout rules and make Numbrix's that stick to their original layout somewhat rare, but usually doesn't look too bad.
- Of course puzzles generated by this program are different because they are not made by hand.
- At the moment there is no functionality for determining the challenge level of a puzzle.

Technical note:

Occasionally generation of a puzzle will happen in-

// Listing continued from previous page

```
  }
  board[xy] = c;
}

int NpathGen::change() {
  int c, d;

  using namespace std;

  time(&timer);
  if (difftime(timer, secs) > (maxx * maxy) / 25) return 0; // bandaid
        // First perform a check for completeness and for isolated squares.
  d = 0;
  for (int xy = 0; xy < (maxx * maxy); xy++) {
    if (board[xy] == NONE) {
      c = 0; d++;
      for (int s = 1; s < 5; s++) {                // check for going off edges
        if (((xy + step[s]) >= 0) && (xy + step[s] < (maxx * maxy))// up & down
          && ((xy % maxx) || ((xy + step[s] + 1) % maxx))          // left and
          && (((xy + step[s]) % maxx) || ((xy + 1) % maxx)))          // right
            if ((board[xy + step[s]] == NONE) // see if empty sqaure is alone
              || (xy + step[s] == start) || (xy + step[s] == end))
              c++;
      }
      if (c == 0) return 0;
    }
  }
  if (d == 0) return 1;

  vector<int> modsloc;
  vector<int> modstype;
  modsloc.push_back(999);
  modstype.push_back(999); // Need a value so insert() works. Will be ignored.
        // Now we make a list of all possible modifications. First the bends...
  for (int xy = 0; xy < (maxx * maxy) - maxx; xy++) {
    if ((xy % maxx) == (maxx - 1)) continue;
    for (c = 0; c < NUMBENDS; c++) {
      if ((bends[c][0][0][0] != END) && (bends[c][0][0][0] != board[xy]))
        continue;
      if ((bends[c][0][0][1] != END) && (bends[c][0][0][1] != board[xy + 1]))
        continue;
      if ((bends[c][0][1][0] != END) && (bends[c][0][1][0] != board[xy + maxx]))
        continue;
      if ((bends[c][0][1][1]!=END) && (bends[c][0][1][1]!=board[xy + maxx + 1]))
        continue;
      d = rand() % modsloc.size();
      modsloc.insert(modsloc.begin() + d,xy);
      modstype.insert(modstype.begin() + d,c);
    }
  }
  for (int s = 1; s < 5; s++) {            // ...then extending the beginning...
    if (((start + step[s]) >= 0) && (start + step[s] < (maxx * maxy))// up & down
      && ((start % maxx) || ((start + step[s] + 1) % maxx))          // left and
      && (((start + step[s]) % maxx) || ((start + 1) % maxx)))          // right
      if (board[start + step[s]] == NONE) {
        d = rand() % modsloc.size();
        modsloc.insert(modsloc.begin() + d, start + step[s]);
        modstype.insert(modstype.begin() + d, 100);
      }
  }
  for (int s = 1; s < 5; s++) {                       // ...then extending the
end.
    if (((end + step[s]) >= 0) && (end + step[s] < (maxx * maxy))  // up & down
```

// Listing continued on next page...

// Listing continued from previous page

```
      && ((end % maxx) || ((end + step[s] + 1) % maxx))            // left and
      && (((end + step[s]) % maxx) || ((end + 1) % maxx)))            // right
        if (board[end + step[s]] == NONE) {
          d = rand() % modsloc.size();
          modsloc.insert(modsloc.begin() + d, end + step[s]);
          modstype.insert(modstype.begin() + d, 101);
        }
  }
  if (modsloc.size() == 0) return 0; // no possbile modifications
  while (modsloc.size()) {                  // iterate through the list.
    int kind = modstype.back(); modstype.pop_back();
    int xy = modsloc.back(); modsloc.pop_back();
    if (kind == 999) continue;
    if(kind < 100) {                          // apply the modification
      if (bends[kind][0][0][0] != END) board[xy]          = bends[kind][1][0][0];
      if (bends[kind][0][0][1] != END) board[xy + 1]      = bends[kind][1][0][1];
      if (bends[kind][0][1][0] != END) board[xy + maxx]   = bends[kind][1][1][0];
      if (bends[kind][0][1][1] != END) board[xy + maxx+1] = bends[kind][1][1][1];
    } else if (kind == 100) {
      if ((xy - start) == -1) board[xy] = RIGHT;
      if ((xy - start) == 1) board[xy] = LEFT;
      if ((xy - start) == -maxx) board[xy] = DOWN;
      if ((xy - start) == maxx) board[xy] = UP;
      d = start; // Save the old starting point in case of undo.
      start = xy;
    } else {
      if ((xy - end) == -1) board[end] = LEFT;
      if ((xy - end) == 1) board[end] = RIGHT;
      if ((xy - end) == -maxx) board[end] = UP;
      if ((xy - end) == maxx) board[end] = DOWN;
      d = end; // save the old ending point in case of undo
      end = xy;
      board[end] = END;
    }
    if (change()) return 1;                        // recursive call to self

              // If that didn't work, undo the modification and try the next one
    if(kind < 100) {
      if (bends[kind][1][0][0] != END) board[xy]          = bends[kind][0][0][0];
      if (bends[kind][1][0][1] != END) board[xy + 1]      = bends[kind][0][0][1];
      if (bends[kind][1][1][0] != END) board[xy + maxx]   = bends[kind][0][1][0];
      if (bends[kind][1][1][1] != END) board[xy + maxx+1] = bends[kind][0][1][1];
    } else if (kind == 100) {
      board[start] = NONE;
      start = d;
    } else {
      board[end] = NONE;
      end = d;
      board[end] = END;
    }
  } // end of loop through list
  return 0;       // None of the modifications worked, so back up and try again.
}
// End NpathGen class

NSolver::NSolver (int x, int y, int** grid) {
  int go;
  signed int dir = -1;

  puzzle = grid;
  sizex = x; sizey = y;
  puzmask = new int[sizex * sizey + 2]; // Adding space for 0.
  for (int c = 0; c < sizex * sizey + 1; c++) puzmask[c] = 0;
```

// Listing continued on next page...

stantly, other times it may take considerably longer. The reason for this is that the path generator will occasionally get stuck. While technically it will eventually succeed if allowed to run this program tried to reduce generation time by timing the path generation and if it's taking too long stop it and start over. It's not the most robust solution, but generally speaking it works.

Numbrix Solver and Generator was written by Joe Larson.

// Listing continued from previous page

```
  for (int xx = 0; xx < sizex; xx++)
    for (int yy = 0; yy < sizey; yy++)
      if (puzzle[xx][yy])
        puzmask[puzzle[xx][yy]] = 1;
  puzmask[0] = 0; puzmask[sizex * sizey + 1] = 1;
  go = findrun (); // find the shortest run and start there.
  if (go > sizex * sizey) { // go is at the end. Back up and turn around.
    dir = 1;
    while (!puzmask[--go]);
  }
  for (int xx = 0; xx < sizex; xx++) // find the go number on the board.
    for (int yy = 0; yy < sizey; yy++)
      if (go == puzzle[xx][yy]) wiggle (xx, yy, go, dir);
}

NSolver::~NSolver () {
  delete [] puzmask;
  for (unsigned int c = 0; c < solutions.size(); c++) {
    int** temp = solutions.at(c);
    for (int x = 0; x < sizex; x++)
      delete [] temp[x];
    delete [] temp;
  }
}

int NSolver::getSolutionXY (int x, int y, unsigned int n) {
  if (solutions.size() < n + 1) return 0;
  int** temp = solutions.at(n);
  return temp[x][y];
}

int NSolver::findrun() { // find the shortest run
  int minrun = sizex * sizey;
  int loc = 0;
  int run = 0;

  for (int c = 1; c <= sizex * sizey + 1; c++) {
    if (puzmask[c]) {
      if ((run) && (run < minrun)) {
        minrun = run;
        loc = c;
      }
      run = 0;
    } else run++;
  }
  return loc;
}

void NSolver::wiggle(int x, int y, int num, int dir) {
  if (x - 1 >= 0)    check (x - 1, y, num + dir, dir);
  if (y - 1 >= 0)    check (x, y - 1, num + dir, dir);
  if (x + 1 < sizex) check (x + 1, y, num + dir, dir);
  if (y + 1 < sizey) check (x, y + 1, num + dir, dir);
}

void NSolver::check(int x, int y, int num, int dir) {

  if (!puzzle[x][y]) {
    if (puzmask[num]) return;
    puzzle[x][y] = num;
    puzmask[num] = 2;
  } else if (puzzle[x][y] != num) return;
  if ((puzmask[num] == 1) || (num == 1) || (num == sizex * sizey)) {
```

// Listing continued on next page...

// Listing continued from previous page

```
    int dirtemp = -1;
    int go = findrun (); // find the shortest run and start there.
    if (!go)  { // We've found a solution. Store it in solutions.
      int** temp = new int*[sizex];
      for (int xx = 0; xx < sizex; xx++) {
        temp[xx] = new int[sizey];
        for (int yy = 0; yy < sizey; yy++)
          temp[xx][yy] = puzzle[xx][yy];
      }
      solutions.push_back (temp);
    } else if (go > sizex * sizey) {// go is at the end. Back up and turn around.
      dirtemp = 1;
      while (!puzmask[--go]);
    }
    for (int xx = 0; xx < sizex; xx++) // find the go number on the board.
      for (int yy = 0; yy < sizey; yy++)
        if (go == puzzle[xx][yy]) wiggle (xx, yy, go, dirtemp);
  } else wiggle (x, y, num, dir);
  if (2 == puzmask[num]) {
    puzzle[x][y] = 0;
    puzmask[num] = 0;
  }
}
// End NSolver class
```

// utils.h listing begins:

```
#ifndef UTILS_H_INCLUDED
#define UTILS_H_INCLUDED

#ifndef NCURSES
#define getmouse(c) nc_getmouse(c)
#endif

class wScreenMask {
  public:
  wScreenMask (WINDOW* w);
  ~wScreenMask ();
  void add_element (int y, int x, std::string prompt, char key);
  int get_input ();

  private:
  WINDOW* win;
  int wx, wy;
  int** keymask;
};

void wcharfillrect(WINDOW *w, char c, int sy, int sx, int height, int width);

void wcentermsg (WINDOW *w, std::vector<std::string> msg, int y, int x, int box);
#endif // UTILS_H_INCLUDED
```

// utils.cpp listing begins:

```
#include <curses.h>
#include <vector>
#include <string>
#include "utils.h"

wScreenMask::wScreenMask (WINDOW* w) {
  win = w;
  getmaxyx (win, wy, wx);
  keymask = new int*[wy];
  for (int yy = 0; yy < 26; yy++) {
```

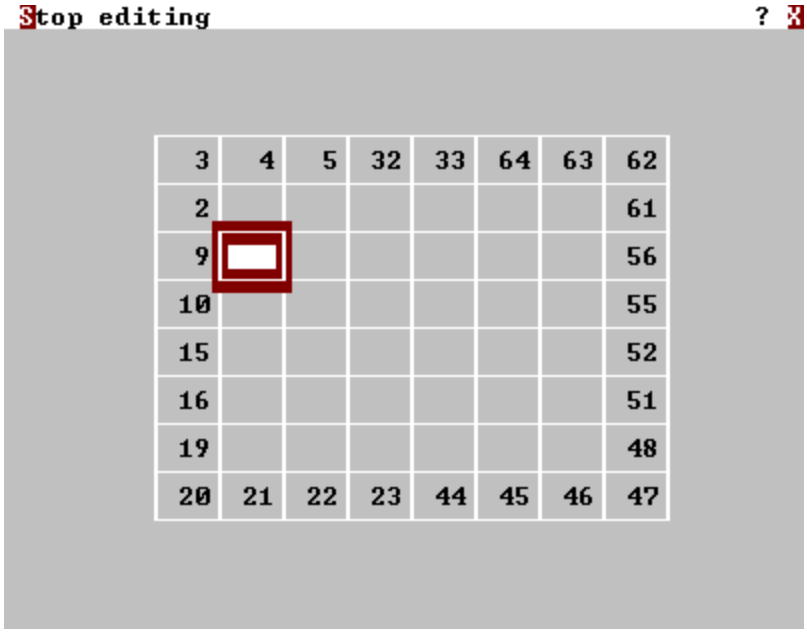
// Listing continued on next page...

// Listing continued from previous page

```
    keymask[yy] = new int[wx];
    for (int xx = 0; xx < 50; xx++)
      keymask[yy][xx] = 0;
  }
}

wScreenMask::~wScreenMask () {
  for (int yy = 0; yy < 26; yy++)
    delete keymask[yy];
  delete keymask;
}

void wScreenMask::add_element (int y, int x, std::string prompt, char key) {
  int attr = 0;

  wmove (win, y, x);
  for (unsigned int c = 0; c < prompt.size(); c++) {
    if ('#' == prompt[c]) {
      c++;
      if ((prompt[c] > '0') && (prompt[c] < '9'))
        attr = COLOR_PAIR(prompt[c++] - '0');
      if ('B' == prompt[c]) { attr |= A_BOLD; c++; }
      if ('K' == prompt[c]) { attr |= A_BLINK; c++; }
    }
    wattrset(win, attr);
    waddch (win, (char)prompt[c]);
    keymask[y][x++] = key;
  }
}

int wScreenMask::get_input () {
  MEVENT mouseinput;
  int input = getch ();

  if (input == KEY_MOUSE) {
    getmouse (&mouseinput);
    input = keymask[mouseinput.y][mouseinput.x];
  }
  return input;
}
//end class wScreenMask

void wcharfillrect(WINDOW *w, char c, 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, c);
  }
}

void wcentermsg (WINDOW *w, std::vector<std::string> msg, int y, int x, int box)
{
  if (!y) y = (LINES - msg.size()) / 2;
  unsigned int maxx = 0;
  for (unsigned int c = 0; c < msg.size(); c++)
    if (msg[c].size() > maxx) maxx = msg[c].size();
  if (!x) x = (COLS - maxx) / 2;
  if (box) wcharfillrect (w, ' ', y - 1, x - 1, msg.size() + 2, maxx + 2);
  for (unsigned int yy = 0; yy < msg.size(); yy++)
    mvwprintw (w, y + yy, x + (maxx - msg[yy].size()) / 2, msg[yy].c_str());
}
```

// main.cpp listing begins:

```
#include <curses.h>
#include <iostream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <string>
#include <sstream>
#include <cstdlib>
#include <ctime>
#include <cmath>
#include <cctype>

#include "numbrixgen.h"
#include "utils.h"

#define VER "ver15Jan2009"

#ifndef NCURSES
#define getmouse(c) nc_getmouse(c)
#endif

using namespace std;

WINDOW* puzwin;

void blankscreen () {
  wattrset (puzwin, COLOR_PAIR (1) | A_BOLD);
  wcharfillrect (puzwin, ' ', 0, 0, 26, 50);
  wattrset (puzwin, COLOR_PAIR (1) | A_BLINK);
  wcharfillrect (puzwin, ' ', 0, 0, 1, 50);
  wrefresh(puzwin);
}

void init() {
  initscr();
  curs_set (0);
  cbreak ();
  noecho();
  keypad(stdscr,1);
  mousemask (ALL_MOUSE_EVENTS, NULL);
#ifdef PDCURSES
  PDC_set_title("Cymon's Games - Numbrix Generator " VER);
#endif
  start_color();
  init_pair (1, COLOR_BLACK, COLOR_WHITE);
  init_pair (2, COLOR_WHITE, COLOR_WHITE); // To be used with | A_BOLD
  init_pair (3, COLOR_WHITE, COLOR_RED);
  init_pair (4, COLOR_WHITE, COLOR_CYAN);
  resize_term(26,50);
  puzwin = newwin(26, 50, 0, 0);
  wattrset (puzwin, COLOR_PAIR (1) | A_BLINK);
  wcharfillrect (puzwin, ' ', 0, 0, 26, 50);
  wattrset (puzwin, COLOR_PAIR (1) | A_BLINK);
  wcharfillrect (puzwin, ' ', 0, 0, 1, 50);
  mvwaddch (puzwin, 0, 49, '?');
  wrefresh(puzwin);
}

void makemask(int type, int puzx, int puzy, int** mask) {
  int xx, yy;

  for (int xx = 0; xx < puzx; xx++)
    for (int yy = 0; yy < puzy; yy++)
```



// Listing continued on next page...

// Listing continued from previous page

```
      mask[xx][yy] = 0;
  switch (type) {
    case 2: // Donught
      for (xx = 1; xx < puzx - 1; xx++)
        for (yy = 1; yy < puzy - 1; yy++)
          if ((xx == 1) || (xx == puzx - 2) || (yy == 1) || (yy == puzy - 2))
            mask[xx][yy] = 1;
      break;
    case 3: // Strawberry
      for (xx = 0; xx < puzx; xx++)
        for (yy = 0; yy < puzy; yy++)
          if (((xx <= 1) || (xx >= puzx - 2) || (yy <= 1) || (yy >= puzy - 2))
            && ((xx + yy + 1) % 2))
            mask[xx][yy] = 1;
      break;
    case 4: // Strawberry Donught
      for (xx = 1; xx < puzx - 1; xx++)
        for (yy = 1; yy < puzy - 1; yy++)
          if (((xx <= 2) || (xx >= puzx - 3) || (yy <= 2) || (yy >= puzy - 3))
            && ((yy + xx + 1) % 2))
            mask[xx][yy] = 1;
      break;
    case 5: // Random
      for (int c = 0; c < ((puzx * puzy) / 25) + 1; c++) { // change this line if
random mask generation is taking too long.
        do {
          yy = rand() % puzy; xx = rand() % puzx;
        } while (mask[xx][yy]);
        mask[xx][yy] = mask[puzx - xx - 1][puzy - yy - 1] =
        mask[xx][puzy - yy - 1] = mask[puzx - xx - 1][yy] = 1;
      }
      break;
    default: // Traditional
      for (yy = 0; yy < puzy; yy++)
        for (xx = 0; xx < puzx; xx++)
          if ((xx == 0) || (xx == puzx - 1) || (yy == 0) || (yy == puzy - 1))
            mask[xx][yy] = 1;
      break;

  }
}

void info () {
  vector<string> message;
  message.push_back ("Numbrix Solver and Generator");
  message.push_back ("----------------------------");
  message.push_back (VER); message.push_back ("");
  message.push_back ("by Joseph Larson for");
  message.push_back ("Cymon's Games");
  message.push_back ("based on the Numbrix puzzle");
  message.push_back ("created by Maryln Vos Savant"); message.push_back ("");
  message.push_back ("For programming resources");
  message.push_back ("source code and a free game");
  message.push_back ("every week please visit:");
  message.push_back ("http://WWW.CYMONSGAMES.COM");
  message.push_back (""); message.push_back ("");
  wattrset (puzwin, COLOR_PAIR (4) | A_BOLD);
  wcentermsg (puzwin, message, 0, 0, 1);
  wattrset (puzwin, COLOR_PAIR (3) | A_BOLD);
  mvwprintw (puzwin, 19, 17, " Press any key ");
  wrefresh (puzwin);
  getch();
}
```

// Listing continued on next page...

// Listing continued from previous page

```
int settings (int* puzx, int* puzy, int* masktype, int batch) {
  wScreenMask screen (puzwin);
  int input, curloc = 0;
  string code;

  *puzy = *puzx = 9; *masktype = 1; code = "";
  while (1) {
    // Visual elements
    blankscreen ();
    wattrset (puzwin, COLOR_PAIR (1) | A_BLINK);
    wcharfillrect (puzwin, ' ', 6, 12, 12 + (!batch) * 2, 26);
    screen.add_element (4, 21, "#1KOptions", 0);
    screen.add_element (7, 18, "#1KSize: X__, Y__", 0);
    screen.add_element (7, 24, "#3BX#2B__", 'X'); // two underscores
    screen.add_element (7, 29, "#3BY#2B__", 'Y'); // two underscores
    screen.add_element (9, 15, "#1KNumbrix Type:", 0);
    screen.add_element (10, 15, "#3B1)#4B Original         ", '1');
    screen.add_element (11, 15, "#3B2)#4B Donut            ", '2');
    screen.add_element (12, 15, "#3B3)#4B Strawberry       ", '3');
    screen.add_element (13, 15, "#3B4)#4B Strawberry Donut ", '4');
    screen.add_element (14, 15, "#3B5)#4B Random           ", '5');
    screen.add_element (16, 14, "#3B Press ENTER to go on ", '\n');
    if (!batch)
      screen.add_element(18, 14, "#1KPress #3BC#1K to use a code", 'C');
    screen.add_element (0, 46, "#1K ? ", '?');
    screen.add_element (0, 49, "#3BX", 'Q');
    wattrset (puzwin, COLOR_PAIR (4) | A_BLINK | A_BOLD);
    if (*masktype)  mvwprintw (puzwin, 9 + *masktype, 14, "*");
    wattrset (puzwin, COLOR_PAIR (2) | A_BOLD);
    mvwprintw (puzwin, 7, 25, "%2d", *puzx);
    mvwprintw (puzwin, 7, 30, "%2d", *puzy);
    if (curloc) {
      curs_set (1);
      if (curloc < 3) wmove (puzwin, 7, 26 + (curloc - 1) * 5);
      else {
        wattrset (puzwin, COLOR_PAIR (2) | A_BOLD);
        mvwprintw (puzwin, 18, 14, "%-22s", code.c_str());
      }
    } else curs_set (0);
    wrefresh(puzwin);
    input = screen.get_input ();
    if ((input >= '0') && (input <= '9')) {
      if (!curloc)
        if ((input >= '1') && (input <= '5')) *masktype = input - '0';
      if (curloc == 1) {
        *puzx = *puzx * 10 + input - '0';
        if (*puzx >= 10) {
          curloc = 0;
          if ((!batch) && (*puzx > 12)) *puzx = 12;
          else if (*puzx > 12) *puzx = 30;
        }
      }
      if (curloc == 2) {
        *puzy = *puzy * 10 + input - '0';
        if (*puzy >= 10) {
          curloc = 0;
          if ((!batch) && (*puzy > 12)) *puzy = 12;
          else if (*puzy > 12) *puzy = 30;
        }
      }
      if (curloc == 3) {
        code += input;
```

// Listing continued on next page...

// Listing continued from previous page

```
      }
    } else {
      if (*puzx < 5) *puzx = 5; if (*puzy < 5) *puzy = 5;
      switch (input) {
        case 'X':
        case 'x':
          curloc = 1; *puzx = 0; break;
        case 'Y':
        case 'y':
          curloc = 2; *puzy = 0; break;
        case 'C':
        case 'c':
          if (!batch) curloc = 3; break;
        case '\b':
          if ((curloc == 3) && (code.length()))
             code.erase(code.end() - 1, code.end()); break;
        case KEY_ENTER :
        case '\n':
          if ((!curloc) || (curloc == 3)) {
            int seed = 1;
            if (curloc == 3) { // parse the code for the X and Y
              *puzx = (code.at(0) - '0') * 10 + (code.at(1) - '0');
              *puzy = (code.at(2) - '0') * 10 + (code.at(3) - '0');
              if ((*puzx < 0) || (*puzx > 12) || (*puzy < 0) || (*puzy > 12)) {
                *puzy = 9; *puzx = 9; curloc = 0; break;
              }
              * masktype = (code.at(4) - '0') * 10 + (code.at(5) - '0');
              if ((*masktype < 1) || (*masktype > 5)) *masktype = 1;
              seed = 0;
              for (int c = 6; c < code.length(); c++)
                seed = seed * 10 + code.at(c) - '0';
            }
            return seed;
          } else curloc = 0;
          break;
        case '?' : info(); curloc = 0; break;
        case 'Q' :
        case 'q' :
        case KEY_EXIT :
          return 0;
        default :
          curloc = 0;
      }
    }
  }
}

int editpuzzle (int puzx, int puzy, signed int** grid, int editable) {
  wScreenMask screen(puzwin);
  MEVENT mouseinput;
  int input, edit, upper, left, boxx, boxy;

  boxx = boxy = 0; edit = 1;
  upper = 13 - puzy;
  left = 25 - (puzx * 2);
  while (1) {
    // Visual Elements
    blankscreen ();
    screen.add_element (0, 1, "#3BS#1Ktop editing " , 'S');
    screen.add_element (0, 46, "#1K ? ", '?');
    screen.add_element (0, 49, "#3BX", 'Q');
    wattrset (puzwin, COLOR_PAIR (2) | A_BOLD);
    for (int yy = 1; yy < (2 * puzy); yy++) {
```

// Listing continued on next page...

// Listing continued from previous page

```
      mvwaddch (puzwin, upper + yy, left, (yy%2) ? ACS_VLINE : ACS_LTEE);
      mvwaddch (puzwin, upper + yy, left + (4 * puzx),
        (yy%2) ? ACS_VLINE : ACS_RTEE);
      for (int xx = 1; xx < (4 * puzx); xx++) {
        mvwaddch (puzwin, upper, left + xx, (xx%4) ? ACS_HLINE : ACS_TTEE);
        mvwaddch (puzwin, upper + 2 * puzy, left + xx, (xx%4)
          ? ACS_HLINE : ACS_BTEE);
        mvwaddch (puzwin, upper + yy, left + xx, (yy%2) ?
          (xx%4) ? ' ' : ACS_VLINE
          : (xx%4)? ACS_HLINE : ACS_PLUS);
      }
    }
    mvwaddch(puzwin, upper, left, ACS_ULCORNER);
    mvwaddch(puzwin, upper + 2 * puzy, left, ACS_LLCORNER);
    mvwaddch(puzwin, upper, left + 4 * puzx, ACS_URCORNER);
    mvwaddch(puzwin, upper + 2 * puzy, left + 4 * puzx, ACS_LRCORNER);

    for (int xx = 0; xx < puzx; xx++)
      for (int  yy = 0; yy < puzy; yy++) {
        screen.add_element (upper + 2 * yy + 1, left + 4 * xx + 1,
          "#1   ", 1); // 3 spaces
        if (grid[xx][yy] < 0) wattrset (puzwin, COLOR_PAIR (1));
        else wattrset (puzwin, COLOR_PAIR (1) | A_BOLD);
        wmove(puzwin, upper + 2 * yy + 1, left + 4 * xx + 1);
        if (grid[xx][yy])
          wprintw (puzwin, "%3d", abs(grid[xx][yy]));
      }
    if (!editable) {curs_set (0); wrefresh (puzwin); return 1;}
    wattrset (puzwin, COLOR_PAIR (3) | A_BOLD);
    wmove (puzwin, upper + 2 * boxy, left + 4 * boxx);
    waddch (puzwin, ACS_ULCORNER); waddch (puzwin, ACS_HLINE);
    waddch (puzwin, ACS_HLINE); waddch (puzwin, ACS_HLINE);
    waddch (puzwin, ACS_URCORNER);
    wmove (puzwin, upper + 2 * boxy + 1, left + 4 * boxx);
    waddch (puzwin, ACS_VLINE);
    wmove (puzwin, upper + 2 * boxy + 1, left + 4 * boxx + 4);
    waddch (puzwin, ACS_VLINE);
    wmove (puzwin, upper + 2 * boxy + 2, left + 4 * boxx);
    waddch (puzwin, ACS_LLCORNER); waddch (puzwin, ACS_HLINE);
    waddch (puzwin, ACS_HLINE); waddch (puzwin, ACS_HLINE);
    waddch (puzwin, ACS_LRCORNER);
    if ((grid[boxx][boxy] >= 0) && (grid[boxx][boxy] < (puzx * puzy))) {
      wattrset (puzwin, COLOR_PAIR (1) | A_BLINK);
      wmove(puzwin, upper + 2 * boxy + 1, left + 4 * boxx + 1);
      if (grid[boxx][boxy])
        wprintw (puzwin, "%3d", abs(grid[boxx][boxy]));
      else wprintw (puzwin, "   "); // 3 spaces
      wmove(puzwin, upper + 2 * boxy + 1, left + 4 * boxx + 3);
      curs_set(1);
    } else curs_set (0);
    wrefresh (puzwin);
    input = screen.get_input ();
    if ((input >= '0') && (input <= '9')) {
      if (grid[boxx][boxy] >= 0) {
        if (edit) grid[boxx][boxy] = edit = 0;
        if (grid[boxx][boxy] * 10 + input - '0' <= (puzx * puzy))
        grid[boxx][boxy] = grid[boxx][boxy] * 10 + input - '0';
      }
    } else {
      getmouse (&mouseinput);
      switch (input) {
        case 1 :
          boxy = (mouseinput.y - upper) / 2;
```

// Listing continued on next page...

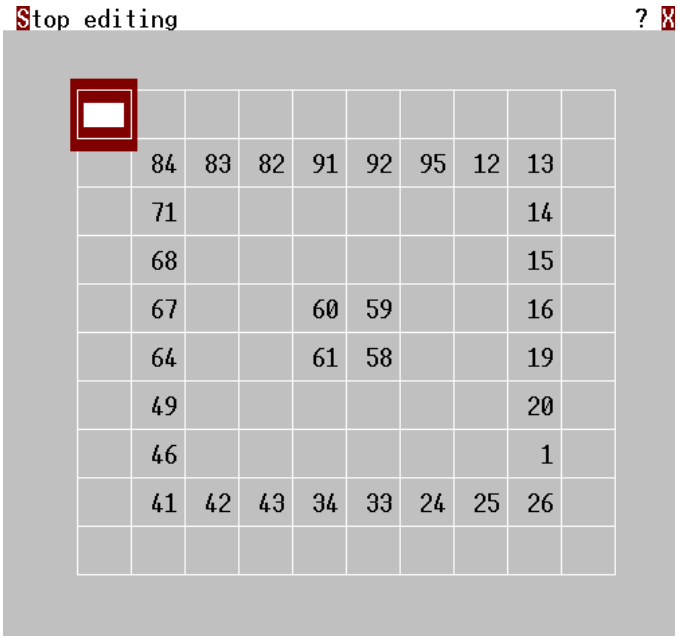
// Listing continued from previous page

```
          boxx = (mouseinput.x - left) / 4;
          edit = 1;
          break;
        case KEY_UP :
          if (boxy > 0) {boxy--; edit = 1;}
          break;
        case KEY_DOWN :
          if (boxy < puzy - 1) {boxy++; edit = 1;}
          break;
        case KEY_LEFT :
          if (boxx > 0) {boxx--; edit = 1;}
          break;
        case KEY_RIGHT :
          if (boxx < puzx - 1) {boxx++; edit = 1;}
          break;
        case KEY_ENTER :
        case '\n' :
          if ((grid[boxx][boxy] > 0) && edit)
            grid[boxx][boxy] = 0;
          break;
        case KEY_BACKSPACE :
        case '\b' :
          if ((grid[boxx][boxy] > 0))
            grid[boxx][boxy] = 0;
          break;
        case '?' :
          info ();
          break;
        case 's' :
        case 'S' :
          return 1;
        case 'q' :
        case 'Q' :
        case KEY_EXIT :
          return 0;
      }
    }
  }
}

void save2txt (int** grid, NpathGen* sol, string name) {
  ofstream puzfile;
  name += ".txt";
  puzfile.open (name.c_str(), ios::out | ios::trunc);
  puzfile << "Numbrix Puzzle\n"
    << "Created with the Numbrix Solver Generator by Joseph Larson\n"
    << "based on the Numbrix puzzle by Maryln Vos Savant.\n"
    << "Software version " << VER << "\n\n"
    << "Puzzle Code: " << setw(2) << setfill('0') << sol->getXSize()
    << setw(2) << setfill('0') << sol->getYSize()
    << setw(2) << setfill('0') << sol->getMask()
    << sol->getSeed() << "\n\n" << setfill (' ');
  for (int yy = 0; yy < sol->getYSize(); yy++)
    for (int line = 0; line < 4; line++) {
      for (int xx = 0; xx < sol->getXSize(); xx++) {
        switch (line) {
          case 0:
            puzfile << "+-----"; break;
          case 2:
            if (grid[xx][yy])
              puzfile << "| " << setw(3) << abs(grid[xx][yy]) << " ";
            else
          case 1:
```

// Listing continued on next page...

// Listing continued from previous page

```
          case 3:
            puzfile << "|     "; // 5 spaces
        }
      }
      switch (line) {
        case 0:
          puzfile << "+\n"; break;
        default :
          puzfile << "|\n"; break;
      }
    }
  for (int xx = 0; xx < sol->getXSize(); xx++)
    puzfile << "+-----";
  puzfile << "+\n\n\n";
  puzfile << "Solution\n\n";
  for (int yy = 0; yy < sol->getYSize(); yy++) {
    for (int xx = 0; xx < sol->getXSize(); xx++)
      puzfile << setw(4) << sol->getXY(xx,yy);
    puzfile << "\n\n";
  }
  puzfile.close();
}

void scrollmsg(vector<string> msg) {
  blankscreen ();
  wattrset (puzwin, COLOR_PAIR (1) | A_BLINK);
  mvwprintw (puzwin, 0,0, "Working...");
  wattrset (puzwin, COLOR_PAIR (1));
  int yy = 25;
  for (vector<string>::iterator s = msg.end(); (s != msg.begin()) && (yy > 0);) {
    s--;
    mvwprintw (puzwin, yy--, 0, s->c_str());
  }
  wrefresh (puzwin);
}

void batch () {
  int input;
  wScreenMask screen (puzwin);
  signed int** grid;
  NSolver* check;
  string name = "numbrix";
  int puzx, puzy, masktype, num, curloc, seed;
  time_t start, timer;
  double secs, totsecs;

  num = 1; curloc = 0; seed = time(NULL);
  if (!settings (&puzx, &puzy, &masktype, 1)) return;
  do {
    blankscreen();
    wattrset (puzwin, COLOR_PAIR (1) | A_BLINK);
    wcharfillrect (puzwin, ' ', 6, 5, 7, 39);
    screen.add_element (4, 17, "#1K Batch Options ", 0);
    screen.add_element (7, 6, "#3BN#1Kumber to generate : #4B   ", 'N');
    screen.add_element (9, 6,
      "#3BF#1Kile base name : #4B                    ", 'F'); // 20 spaces
    screen.add_element (11, 14, "#3B Press ENTER to start ", '\n');
    wattrset (puzwin, COLOR_PAIR (4) | A_BOLD);
    mvwprintw (puzwin, 7, 27, "%3d", num);
    mvwprintw (puzwin, 9, 23, "%-20s", name.c_str());
    vector <string> message;
    message.push_back ("Time to generate varies greatly.");
    message.push_back ("If you are generating many puzzles");
```



// Listing continued on next page...

// Listing continued from previous page

```
    message.push_back ("or very large puzzles");
    message.push_back ("please be patient."); message.push_back ("");
    message.push_back ("Generated puzzle files will appear");
    message.push_back ("in the directory this program is");
    message.push_back ("being run from. Any files with");
    message.push_back ("the same name will be overwritten.");
    wcentermsg (puzwin, message, 15, 0, 1);
    if (curloc) {
      curs_set (1);
      if (curloc == 1) wmove (puzwin, 7, 29);
      else wmove (puzwin, 9, 23 + name.length ());
    } else curs_set (0);
    wrefresh (puzwin);
    input = screen.get_input ();
     if (curloc == 2) {
      if (isalnum(input)) name += input;
      if (input == '\b') name.erase(name.end() - 1);
      if (input == '\n') curloc = 0;
      input = 0;
    } else {
      if (curloc == 1) {
        if ((input >= '0') && (input <= '9'))
          num = num * 10 + input - '0';
        if (input == '\b') num /= 10;
        if ((input == '\n') || (num >= 100)) curloc = 0;
      }
      if (num < 0) num = 1;
      switch (input) {
      case 'N' :
      case 'n' :
        if (curloc != 2) {num = 0; curloc = 1;} break;
      case 'F' :
      case 'f' :
        if (curloc != 2) {curloc = 2;} break;
      case '?' :
        info (); break;
      case 'Q' :
      case 'q' :
        return;
      }
    }
  } while (input != '\n');
  curs_set (1);
  grid = new int*[puzx];
    for (int xx = 0; xx < puzx; xx++)
      grid[xx] = new int[puzy];
  vector <string> messages;

  messages.push_back ("Beginning Generation");
  totsecs = 0;
  for (int c = 0; c < num; c ++) {
    time (&start);
    ostringstream digits;
    digits  << setw(3) << setfill ('0') << c + 1;
    string nametemp = name + digits.str();
    messages.push_back ("Generating " + nametemp + "...");
    scrollmsg(messages);
    seed += rand ();
    NpathGen puzzle(puzx, puzy, seed);
    puzzle.setMask (masktype);
    makemask (masktype, puzx, puzy, grid);
    for (int xx = 0; xx < puzx; xx++)
      for (int  yy = 0; yy < puzy; yy++)
```

// Listing continued on next page...

// Listing continued from previous page

```
        grid[xx][yy] = (grid[xx][yy]) ? puzzle.getXY(xx, yy) : 0;
    messages.push_back ("Checking uniqueness...");
    scrollmsg(messages);
    check = new NSolver(puzx, puzy, grid);
    while (check->getNumSolutions () > 1) {
      scrollmsg(messages);
      int xx; int yy;
      do {
        xx = rand() % puzx; yy = rand () % puzy;
      } while (grid[xx][yy] == 0);
      grid[xx][yy] = 0; // take one away.
      grid[xx][puzy - yy - 1] = 0;
      grid[puzx - xx - 1][yy] = 0;
      grid[puzx - xx - 1][puzy - yy - 1] = 0;
      do {
        xx = rand() % puzx; yy = rand () % puzy;
      } while (check->getSolutionXY(xx, yy, 0)==check->getSolutionXY(xx, yy, 1));
      grid[xx][yy] = puzzle.getXY(xx, yy); // add a different one in.
      grid[xx][puzy - yy - 1] = puzzle.getXY(xx, puzy - yy - 1);
      grid[puzx - xx - 1][yy] = puzzle.getXY(puzx - xx - 1, yy);
      grid[puzx - xx - 1][puzy - yy - 1]
        = puzzle.getXY(puzx - xx - 1, puzy - yy - 1);
      delete check;
      check = new NSolver(puzx, puzy, grid);
      messages.push_back ("Multiple solutions found. Fixed. Checking again.");
      scrollmsg(messages);
    }
    ostringstream tictoc;
    tictoc.flush();
    time (&timer);
    secs = difftime (timer, start);
    totsecs += secs;
    tictoc << (int)secs;
    messages.push_back (tictoc.str() + " seconds to generate.");
    messages.push_back ("Saving " + nametemp + "...");
    scrollmsg(messages);
    save2txt (grid, &puzzle, nametemp);
  }
  ostringstream minutes, seconds, average;
  minutes << (int)totsecs / 60;
  seconds << (int)totsecs % 60;
  messages.push_back (minutes.str() + ":" + seconds.str() +
    " spent generating.");
  totsecs /= num;
  average << totsecs;
  messages.push_back (seconds.str() + " seconds per on average.");
  scrollmsg(messages);
  curs_set (0);
  wattrset (puzwin, COLOR_PAIR (4) | A_BOLD);
  vector <string> message;
  message.push_back ("Batch Complete."); message.push_back ("");
  message.push_back ("Files have been created in the");
  message.push_back ("directory this program was");
  message.push_back ("run from."); message.push_back ("");
  message.push_back ("Press any key...");
  wcentermsg (puzwin, message, 0, 0, 1);
  wrefresh (puzwin);
  getch ();
}

void solve () {
  wScreenMask screen(puzwin);
  int input;
```

// Listing continued on next page...

// Listing continued from previous page

```
  int curloc;
  signed int** grid;
  NSolver* check;
  int puzx, puzy;
  time_t start, timer;
  double secs;

  puzy = puzx = 9; curloc = 0;
  while (1) {
    blankscreen ();
    wattrset (puzwin, COLOR_PAIR (1) | A_BLINK);
    wcharfillrect (puzwin, ' ', 6, 13, 5, 24);
    screen.add_element(4, 21, "#1KOptions", 0);
    screen.add_element(7, 18, "#1KSize: X__, Y__", 0);
    screen.add_element(9, 14, "#3B Press ENTER to go on ", '\n');
    screen.add_element(7, 24, "#3BX#3B__", 'X');
    screen.add_element(7, 29, "#3BY#2B__", 'Y');
    wattrset (puzwin, COLOR_PAIR (2) | A_BOLD);
    mvwprintw (puzwin, 7, 25, "%2d", puzx);
    mvwprintw (puzwin, 7, 30, "%2d", puzy);
    screen.add_element (0, 46, "#1K ? ", '?');
    screen.add_element (0, 49, "#3BX", 'Q');
    vector<string> message;
    wattrset (puzwin, COLOR_PAIR (4) | A_BOLD);
    message.push_back ("Editor Commands:          "); message.push_back ("");
    message.push_back ("Use Mouse or Keyboard"); message.push_back ("");
    message.push_back ("Arrow Keys - Navigate     ");
    message.push_back ("Numbers Keys - Enter guess");
    message.push_back ("BackSpace - Clear Square  ");
    message.push_back ("'S' - Solve               ");
    wcentermsg (puzwin, message, 13, 0, 1);
    if (curloc) {
      curs_set (1);
      wmove (puzwin, 7, 26 + (curloc - 1) * 5);
    } else curs_set (0);
    wrefresh(puzwin);
    // Process Input
    input = screen.get_input ();
    if ((input >= '0') && (input <= '9')) {
      if (curloc == 1) {
        puzx = puzx * 10 + input - '0';
        if (puzx >= 10) {
          curloc = 0;
          if (puzx > 12) puzx = 12;
        }
      }
      if (curloc == 2) {
        puzy = puzy * 10 + input - '0';
        if (puzy >= 10) {
          curloc = 0;
          if (puzy > 12) puzy = 12;
        }
      }
    } else {
      if (puzx < 5) puzx = 5; if (puzy < 5) puzy = 5;
      switch (input) {
        case 'X':
        case 'x':
          curloc = 1; puzx = 0; break;
        case 'Y':
        case 'y':
          curloc = 2; puzy = 0; break;
        case KEY_ENTER :
```

// Listing continued on next page...

// Listing continued from previous page

```
        case '\n':
          if (!curloc) {
            grid = new int*[puzx];
            for (int xx = 0; xx < puzx; xx++) {
              grid[xx] = new int[puzy];
              for (int yy = 0; yy < puzy; yy++)
                grid[xx][yy] = 0;
            }
            do {
              if (!editpuzzle(puzx, puzy, grid, 1)) return;
              wattrset (puzwin, COLOR_PAIR (4) | A_BOLD);
              mvwprintw (puzwin, 13, 19, " Solving... "); wrefresh (puzwin);
              check = new NSolver(puzx, puzy, grid);
              if (check->getNumSolutions() > 1) {
                vector<string> message;
                wattrset (puzwin, COLOR_PAIR (4) | A_BOLD);
                message.push_back("Multiple solutions found.");
                message.push_back("Press 'C' to change between");
                message.push_back("them.");
                message.push_back("Press 'Q' to Quit.");
                message.push_back("Press Any Key");
                wcentermsg (puzwin, message, 0, 0, 1);
                wrefresh(puzwin);
                getch ();
              }
              if (check->getNumSolutions() < 1) {
                vector<string> message;
                wattrset (puzwin, COLOR_PAIR (4) | A_BOLD);
                message.push_back("No solutions found."); message.push_back("");
                message.push_back("You may have accidently entered");
                message.push_back("a number wrong."); message.push_back("");
                message.push_back("Press any key to edit the puzzle.");
                wcentermsg (puzwin, message, 0, 0, 1);
                wrefresh(puzwin);
                getch ();
              } else {
                do {
                  wattrset (puzwin, COLOR_PAIR (1) | A_BLINK);
                  wcharfillrect (puzwin, ' ', 10, 5, 6, 40);
                  screen.add_element (11, 6,
                    "#1KDo you want to try solving this puzzle", 0);
                  screen.add_element (12, 8,
                    "#1KBefore you are shown the solution?", 0);
                  screen.add_element (14, 19, "#3B Yes ", 'Y');
                  screen.add_element (14, 27, "#3B No ", 'N');
                  screen.add_element (0, 46, "#1K ? ", '?');
                  screen.add_element (0, 49, "#3BX", 'Q');
                  wrefresh (puzwin);
                  input = screen.get_input();
                  switch (input) {
                    case 'y' : input = 'Y'; break;
                    case 'n' : input = 'N'; break;
                    case 'Q' :
                    case 'q' : return;
                    case '?' : info();
                  }
                } while ((input != 'N') && (input != 'Y'));
                if (input == 'Y') {
                  for (int xx = 0; xx < puzx; xx++)
                    for (int  yy = 0; yy < puzy; yy++)
                      grid[xx][yy] = -grid[xx][yy];
                  int won = 1;
                  time (&start);
```

// Listing continued on next page...

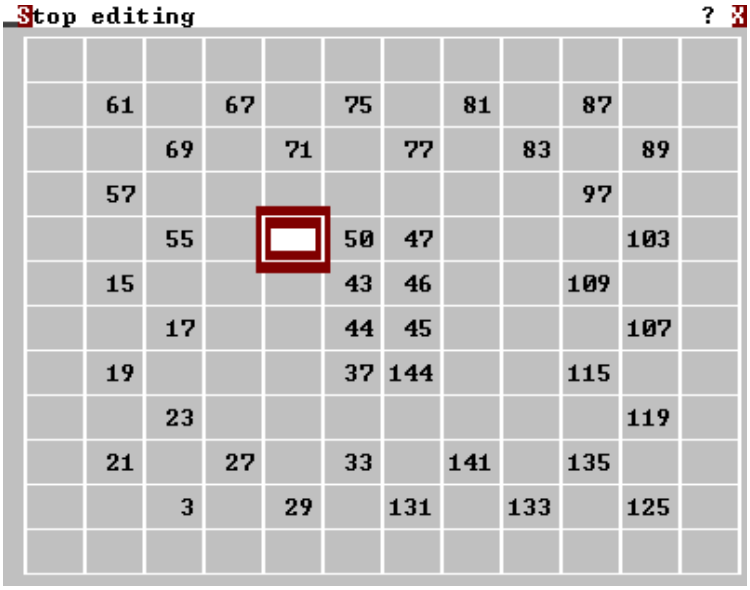
// Listing continued from previous page

```
      do {
        editpuzzle(puzx, puzy, grid, 1);
        won = 1;
        for (int xx = 0; xx < puzx; xx++)
          for (int  yy = 0; yy < puzy; yy++)
            if (abs(grid[xx][yy]) != check->getSolutionXY(xx, yy, 0))
              won = 0;
        if (!won) {
          wattrset (puzwin, COLOR_PAIR (1) | A_BLINK);
          wcharfillrect (puzwin, ' ', 11, 8, 5, 34);
          screen.add_element (12, 10,
            "#1KAre you sure you want to stop?", 0);
          screen.add_element (14, 19, "#3B Yes ", 'Y');
          screen.add_element (14, 27, "#3B No ", 'N');
          screen.add_element (0, 46, "#1K ? ", '?');
          screen.add_element (0, 49, "#3BX", 'Q');
          wrefresh (puzwin);
          do {
            input = screen.get_input ();
            switch (input) {
              case 'y' :
              case 'Y' :
              case 'Q' :
              case 'q' :
                input = 1; break;
              case 'n' :
              case 'N' :
                input = 2; break;
              case '?' :
                info ();
              default :
                input = 0;
            }
          } while (!input);
        }
      } while ((input != 1) && !won);
      time (&timer);
      secs = difftime (timer, start);
      if (won) {
        ostringstream minutes, seconds;
        minutes << (int)secs / 60;
        seconds << setw(2) << setfill ('0') << (int)secs % 60;
        wattrset (puzwin, COLOR_PAIR (4) | A_BOLD);
        vector <string> message;
        message.push_back ("Congratulations");
        message.push_back ("");
        message.push_back ("You solved that puzzle in.");
        message.push_back (minutes.str() + ":" + seconds.str());
        message.push_back ("");
        message.push_back ("Press Any key...");
        wcentermsg (puzwin, message, 0, 0, 1);
        wrefresh (puzwin);
        getch ();
        return;
      }
      for (int xx = 0; xx < puzx; xx++)
        for (int  yy = 0; yy < puzy; yy++)
          grid[xx][yy] = abs(grid[xx][yy]);
    }
    int sol = 0;
    do {
      for (int yy = 0; yy < puzy; yy++)
        for (int xx = 0; xx < puzx; xx++)
```

// Listing continued on next page...

// Listing continued from previous page

```
                    if (grid[xx][yy] == check->getSolutionXY (xx, yy, sol))
                      grid[xx][yy] = -check->getSolutionXY (xx, yy, sol);
                    else grid[xx][yy] = check->getSolutionXY (xx, yy, sol);
                  blankscreen ();
                  if (!editpuzzle (puzx, puzy, grid, 0)) return;
                  wattrset (puzwin, COLOR_PAIR (1) | A_BLINK);
                  screen.add_element (0, 0, "#3BE#1Kdit ", 'E');
                  screen.add_element (0, 5, "#3BQ#1Kuit     ", 'Q');
                  if (check->getNumSolutions() > 1 )
                    screen.add_element (0, 10, "#3BC#1Khange Solution", 'C');
                  wrefresh(puzwin);
                  input = screen.get_input();
                  switch (input) {
                    case 'e' :
                      input = 'E'; break;
                    case 'C' :
                    case 'c' :
                      sol = (++sol)%check->getNumSolutions(); break;
                    case '?' :
                      info (); break;
                    case 'Q' :
                    case 'q' :
                      return;
                  }
                  for (int yy = 0; yy < puzy; yy++)
                    for (int xx = 0; xx < puzx; xx++)
                      if (grid[xx][yy] < 0) grid[xx][yy] = -grid[xx][yy];
                      else grid[xx][yy] = 0;
                } while (input != 'E');
              }
              delete check;
            } while (1);
          } else curloc = 0;
          break;
        case '?' :
          info (); break;
        case 'Q' :
        case 'q' :
          return;
        default :
          curloc = 0;
      }
    }
  }
}

void play () {
  int input = 0;
  wScreenMask screen (puzwin);
  signed int** grid;
  NSolver* check;
  int puzx, puzy, masktype, won, seed;
  time_t start, timer;
  double secs;

  seed = settings (&puzx, &puzy, &masktype, 0);
  if (!seed) return;
  if (seed == 1) seed = time(NULL);
  blankscreen();
  wattrset (puzwin, COLOR_PAIR (3) | A_BOLD);
  curs_set (1);
  mvwprintw (puzwin, 10, 11, "This may take a few minutes");
  mvwprintw (puzwin, 12, 14, " GENERATING PUZZLE... "); wrefresh(puzwin);
```



// Listing continued on next page...

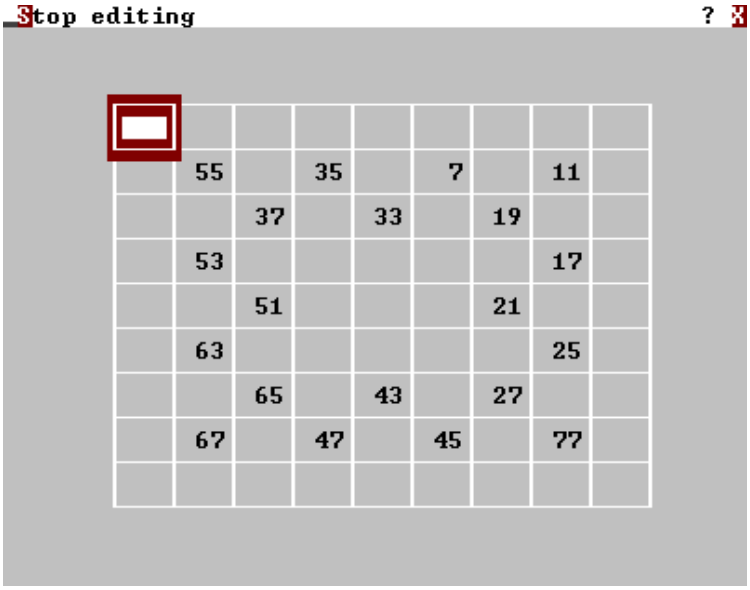
// Listing continued from previous page

```
  NpathGen puzzle(puzx, puzy, seed);
  puzzle.setMask (masktype);
  grid = new int*[puzx];
  for (int xx = 0; xx < puzx; xx++)
    grid[xx] = new int[puzy];
  makemask (masktype, puzx, puzy, grid);
  for (int xx = 0; xx < puzx; xx++)
    for (int  yy = 0; yy < puzy; yy++)
      grid[xx][yy] = (grid[xx][yy]) ? puzzle.getXY(xx, yy) : 0;
  mvwprintw (puzwin, 14, 13, " CHECKING UNIQUENESS... "); wrefresh(puzwin);
  check = new NSolver(puzx, puzy, grid);
  while (check->getNumSolutions () > 1) {
    mvwprintw (puzwin, 16, 19, " FIXING... "); wrefresh(puzwin);
    int xx; int yy;
    do {
      xx = rand() % puzx; yy = rand () % puzy;
    } while (grid[xx][yy] == 0);
    grid[xx][yy] = 0; // take one away.
    grid[xx][puzy - yy - 1] = 0;
    grid[puzx - xx - 1][yy] = 0;
    grid[puzx - xx - 1][puzy - yy - 1] = 0;
    do {
      xx = rand() % puzx; yy = rand () % puzy;
    } while (check->getSolutionXY(xx, yy, 0) == check->getSolutionXY(xx, yy, 1));
    grid[xx][yy] = puzzle.getXY(xx, yy); // add a different one in.
    grid[xx][puzy - yy - 1] = puzzle.getXY(xx, puzy - yy - 1);
    grid[puzx - xx - 1][yy] = puzzle.getXY(puzx - xx - 1, yy);
    grid[puzx - xx - 1][puzy - yy - 1]
      = puzzle.getXY(puzx - xx - 1, puzy - yy - 1);
    delete check;
    check = new NSolver(puzx, puzy, grid);
  }
  for (int  xx = 0; xx < puzx; xx++)
    for (int yy = 0; yy < puzy; yy++)
      grid[xx][yy] = -grid[xx][yy]; // "lock" masked squares
  save2txt (grid, &puzzle, "last");
  curs_set (0);
  wattrset(puzwin, COLOR_PAIR (4) | A_BOLD);
  vector<string> message;
  message.push_back ("Ready!"); message.push_back ("");
  message.push_back ("Use Mouse or Keyboard"); message.push_back ("");
  message.push_back ("Editor Keys:              "); message.push_back ("");
  message.push_back ("Arrow Keys - Navigate     ");
  message.push_back ("Numbers Keys - Enter guess");
  message.push_back ("BackSpace - Clear Square  ");
  message.push_back ("'S' - Stop                "); message.push_back ("");
  message.push_back ("If you have the correct solution");
  message.push_back ("when you hit stop your time will");
  message.push_back ("be counted and you'll win."); message.push_back ("");
  message.push_back ("Press Any Key to Start");
  wcentermsg (puzwin, message, 0, 0, 1);
  wattrset (puzwin, COLOR_PAIR (1) | A_BLINK);
  mvwprintw (puzwin, 25, 0, "Code: %02d%02d%02d%d ",
    puzzle.getXSize(), puzzle.getYSize(), puzzle.getMask(), puzzle.getSeed());
  wrefresh (puzwin); getch ();
  time (&start);
  do {
    editpuzzle(puzx, puzy, grid, 1);
    won = 1;
    for (int xx = 0; xx < puzx; xx++)
      for (int  yy = 0; yy < puzy; yy++)
        if (abs(grid[xx][yy]) != puzzle.getXY(xx, yy)) won = 0;
    if (!won) {
```

// Listing continued on next page...

// Listing continued from previous page

```
      wattrset (puzwin, COLOR_PAIR (1) | A_BLINK);
      wcharfillrect (puzwin, ' ', 11, 8, 5, 34);
      screen.add_element (12, 10, "#1KAre you sure you want to stop?", 0);
      screen.add_element (14, 19, "#3B Yes ", 'Y');
      screen.add_element (14, 27, "#3B No ", 'N');
      screen.add_element (0, 46, "#1K ? ", '?');
      screen.add_element (0, 49, "#3BX", 'Q');
      wattrset (puzwin, COLOR_PAIR (1) | A_BLINK);
      mvwprintw (puzwin, 25, 0, "Code: %02d%02d%02d%d ",
      puzzle.getXSize(), puzzle.getYSize(), puzzle.getMask(), puzzle.getSeed());
      wrefresh (puzwin);
      do {
        input = screen.get_input ();
        switch (input) {
          case 'y' :
          case 'Y' :
          case 'Q' :
          case 'q' :
            input = 1; break;
          case 'n' :
          case 'N' :
            input = 2; break;
          case '?' :
            info ();
          default :
            input = 0;
        }
      } while (!input);
    }
  } while ((input != 1) && !won);
  for (int yy = 0; yy < puzy; yy++)
    for (int xx = 0; xx < puzx; xx++)
      grid[xx][yy] = (abs(grid[xx][yy]) == puzzle.getXY(xx, yy))
        ? -puzzle.getXY(xx, yy) : puzzle.getXY(xx, yy);
  editpuzzle (puzx, puzy, grid, 0);
  wattrset (puzwin, COLOR_PAIR (1) | A_BLINK);
  time (&timer);
  secs = difftime (timer, start);
  mvwprintw (puzwin, 0, 1, "Time %d:%02d  Press Any Key"
    , (int) secs / 60, (int) secs % 60);
  wattrset (puzwin, COLOR_PAIR (1) | A_BLINK);
  mvwprintw (puzwin, 25, 0, "Code: %02d%02d%02d%d ",
  puzzle.getXSize(), puzzle.getYSize(), puzzle.getMask(), puzzle.getSeed());
  if (won) {
    ostringstream minutes, seconds;
    minutes << (int)secs / 60;
    seconds << setw(2) << setfill ('0') << (int)secs % 60;
    wattrset (puzwin, COLOR_PAIR (4) | A_BOLD);
    vector <string> message;
    message.push_back ("Congratulations");
    message.push_back ("");
    message.push_back ("You solved that puzzle in.");
    message.push_back (minutes.str() + ":" + seconds.str());
    message.push_back ("");
    message.push_back ("Press Any key...");
    wcentermsg (puzwin, message, 0, 0, 1);
  }
  wrefresh (puzwin);
  getch ();
  return;
}

void mainmenu () {
```



// Listing continued on next page...

// Listing continued from previous page

```
  int input;
  wScreenMask screen(puzwin);

  while (1) {
    // Visual elements.
    blankscreen ();
    wattrset (puzwin, COLOR_PAIR (1) | A_BLINK);
    wcharfillrect (puzwin, ' ', 8, 9, 11, 32);
    screen.add_element (6, 10, "#1K Numbrix Solver and Generator ", 0);
    screen.add_element (9, 17, "#1KDo you want to:",0);
    screen.add_element (11, 13, "#3B1)#4B Play a random puzzle? ", '1');
    screen.add_element (13, 15, "#3B2)#4B Solve a Numbrix? ", '2');
    screen.add_element (15, 11, "#3B3)#4B Make a batch of puzzles? ", '3');
    screen.add_element (17, 17, "#3BPress Q to quit", 'Q');
    screen.add_element (0, 46, "#1K ? ", '?');
    screen.add_element (0, 49, "#3BX", 'Q');
    wrefresh (puzwin);
    // get and process input.
    input = screen.get_input ();
    switch (input) {
      case '1' : play (); break;
      case '2' : solve (); break;
      case '3' : batch (); break;
      case 'Q' :
      case 'q' :
      case KEY_EXIT :
        exit (0);
      case '?' : info();
    }
  }
}

int main() {
  init();
  mainmenu();
  endwin();
  return 0;
}
```