1. 問題

迷路を生成するプログラム、迷路の脱出経路を探索するプログラムをそれぞれ作成する。

迷路は以下の条件で生成する。

・100×100のマス目

・X,Yの値が0,99のところはすべて壁

・立体交差はしない

・スタートとゴールのマスは床

・1つのスタートと1つのゴールが必ず存在する

・スタートからゴールまでの経路が必ず存在する

迷路は、まず初めにすべてのマスを壁として生成し、左上端の部分を床にしてその地点から床を掘り進めていく形で生成した。

掘り進める条件は、現在地点が床であり、なおかつランダムで定めた上下左右の4方向いずれかの2マス先が外壁でない壁の場合に、その方向に2マスずつ掘り進める、とした。

これを掘ることができるマスがなくなるまで繰り返すことで床と壁が1マスの幅で続く迷路を生成した。

ただし迷路生成の条件が100×100のマスなので偶数であり、上記の床を掘る条件だと右端の一列と下端の1行が壁のままとなってしまうので50列50行以上を次の列、行にずらし、50列目と50行目のマスそれぞれが床ならその場所を壁にして隣を床に、壁ならその隣を壁にして拡張した。

通路探索はスタート位置から開始して上、下、左、右の順に、進めるかどうか及びゴールが存在するかを検索し、ゴールがある場合はその場で終了、ゴールが無く、進める床がある場合に1マス進む、という挙動を繰り返し、ゴールに到達するまで繰り返すという処理で迷路を探索した。

通路探索プログラムは結果ファイルを出力するのみのモードと経路探索の状態をコンソールで表示するモードがある。コンソール表示は高速で動作するモードと動きが確認しやすいモードがある。マップ全てを表示させるのは難しいので、現在の座標を中心に20×20マスを表示している。

1. プログラム構成図

　　プログラムファイルは図1に示すものを作成した。迷路生成プログラムと脱出プログラムはそれぞれ別のプロジェクトとして作成し、1つのソリューションにまとめた。

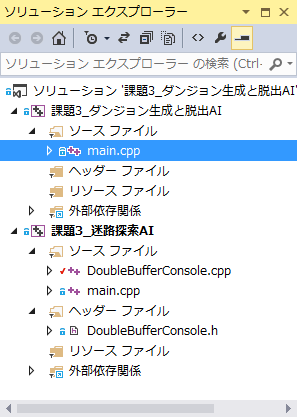


図1：ソリューションエクスプローラー

迷路生成プログラムは１つのファイルに記述した。経路探索プログラムはコンソール表示のためにダブルバッファリングを行うためのクラスを別のファイルで作成した。

1. ソースプログラム

迷路生成のプログラムを以下に示す。

// 課題3\_ダンジョン生成と脱出AI

// 作成日：2016/6/16

//

// main.cpp

// ダンジョン生成プログラム

//

// 参考ウェブ:

// c言語で迷路：http://studio.s1.xrea.com/prog/maze.html

#include <fstream>

#include <iostream>

#include <stdlib.h>

#include <time.h>

#include <Windows.h>

using namespace std;

enum BLOCK{

FLOOR, // 床

WALL, // 壁

START, // スタート

GOAL, // ゴール

ROOT, // ダミー

PLAYER // 探索位置

};

const int sizeX = 100;

const int sizeY = 100;

int X[sizeX\*sizeY];

int Y[sizeX\*sizeY];

BLOCK map[sizeY][sizeX]; // ダンジョンマップ

// 掘り進める関数

void dig( int x, int y )

{

int dx, dy, r; // 方向とランダムの変数

int count = 0; // カウンタ

r = rand();

while( count < 4 ){

switch( ( r + 4 + count ) % 4 ){

case 0:

dx = 0; dy = -1;

break;

case 1:

dx = -1; dy = 0;

break;

case 2:

dx = 0; dy = 1;

break;

case 3:

dx = 1; dy = 0;

break;

}

// 掘る方向が壁か、2マス先が壁でない場合、カウンタ増加

if( x + dx\*2 <= 0 || x + dx\*2 >= sizeX-1 ||

y + dy\*2 <= 0 || y + dy\*2 >= sizeY-1 ||

map[y+dy\*2][x+dx\*2] != WALL ) count++;

else if( map[y+dy\*2][x+dx\*2] == WALL ){// 掘れる条件に合致

map[y+dy][x+dx] = map[y+dy\*2][x+dx\*2] = FLOOR;

x = x + dx\*2;

y = y + dy\*2;

count = 0;

r = rand();

}

}

}

// メイン関数

int main(){

// マップ初期化

// まず全てを壁で埋める

for( int y = 0; y < sizeY; y++ ){

for( int x = 0; x < sizeX; x++ ){

map[y][x] = WALL;

}

}

// 左上端から掘り始める

map[1][1] = FLOOR;

while( 1 ){

int i = 0;

for( int y = 1; y < sizeY - 1; y += 2 ){

for( int x = 1; x < sizeX - 1; x += 2 ){

if( map[y][x] == FLOOR ){ // 床から掘り始める

// 2マス先が外壁以外の壁の場合、

if( ( x-2 >= 0 && map[y][x-2] == WALL ) || ( y-2 >= 0 && map[y-2][x] == WALL ) ||

( x+2 < sizeX-1 && map[y][x+2] == WALL ) || ( y+2 < sizeY-1 && map[y+2][x] == WALL ) ){

X[i] = x; // 掘り進める候補とする

Y[i] = y;

i++;

}else if( ( y == sizeY-2 ) && ( x == sizeX - 2) ) break;

}

}

}

if( i == 0 ) break; // 候補なし

else{

srand((unsigned)time(NULL));

int d = rand() % i;// 候補の中からランダムで掘る

int x = X[d];

int y = Y[d];

dig( x, y );

}

}

// 縦1列右に拡張

for( int x = sizeX - 2; x >= sizeX / 2; x-- ){

for( int y = sizeY - 2; y > 0; y-- ){

map[x][y] = map[x-1][y];

// 床が2列にならないようにする

if( x == sizeX/2 && map[x-2][y] == WALL && map[x-1][y] == FLOOR && map[x][y] == FLOOR )

map[x-1][y] = WALL;

}

}

// 横1列下に拡張

for( int y = sizeY - 2; y >= sizeY / 2; y-- ){

for( int x = sizeX - 2; x > 0; x-- ){

map[x][y] = map[x][y-1];

// 床が2列にならないようにする

if( y == sizeY/2 && map[x][y-2] == WALL && map[x][y-1] == FLOOR && map[x][y] == FLOOR )

map[x][y-1] = WALL;

}

}

// スタート位置設定

int sx, sy;

do{

sx = rand() % (sizeX-2) + 1;

sy = rand() % (sizeY-2) + 1;

}while( map[sx][sy] != FLOOR ); // 通路上に設定されない限りループ

map[sy][sx] = START;

// ゴール位置設定

int gx, gy;

do{

gx = rand() % (sizeX-2) + 1;

gy = rand() % (sizeY-2) + 1;

}while( map[gx][gy] != FLOOR ); // 通路上に設定されない限りループ

map[gy][gx] = GOAL;

// ファイル出力

ofstream ofs;

ofs.open( "maze.txt" );

for( int y = 0; y < sizeY; y++ ){

for( int x = 0; x < sizeX; x++ ){

switch( map[y][x] ){

case FLOOR:

ofs << "0";

break;

case WALL:

ofs << "1";

break;

case START:

ofs << "2";

break;

case GOAL:

ofs << "3";

break;

}

ofs << " ";

}

ofs << endl;

}

ofs.close();

return 0;

}

プログラム１：迷路生成プログラム main.cpp

経路探索プログラムを以下に示す。

// 課題3\_ダンジョン生成と脱出AI

// 作成日：2016/6/29

//

// main.cpp

// 脱出経路探索プログラム

//

// 参考ウェブ:

// c言語で迷路：http://studio.s1.xrea.com/prog/maze.html

// ダブルバッファリング：http://tama012.blogspot.jp/2012/11/blog-post.html

// http://vivi.dyndns.org/tech/cpp/DblBuffer.html

#include <fstream>

#include <iostream>

using namespace std;

#include "DoubleBufferConsole.h"

enum BLOCK{

FLOOR, // 床

WALL, // 壁

START, // スタート

GOAL, // ゴール

ROOT, // ダミー

PLAYER // 探索位置

};

const int sizeX = 100;

const int sizeY = 100;

BLOCK map[sizeY][sizeX]; // ダンジョンマップ

DoubleBufferConsole db; // ダブルバッファコンソール

bool clear = false; // ゴールしたかどうか

// 更新速度モード(描画なし、高速、低速)

enum UpdateMode{

NODRAW, FAST, NORMAL

}mode;

// プロトタイプ宣言

void Draw( int, int );

bool Check( int, int );

void Root( int, int );

// マップ描画

void Draw( int posY, int posX ){

if( mode == NODRAW ) return;

db.SetCursorPos( 0, 0 );

int row = 0; // 描画する行数

for( int y = posY-10; y < posY+10; y++ ){ // 引数で指定したy座標の上下10行分を描画

if( y < 0 || y >= sizeY ) for( int i = 0; i < 20;i++ ) db.Write( "×" ); // 外側は×を表示

else{

for( int x = posX-10; x < posX+10; x++ ){ // 引数で指定したx座標の左右10列分を描画

if( x < 0 || x >= sizeX ) db.Write( "×" ); // 外側は×を表示

else {

switch( map[y][x] ){

case FLOOR: db.Write( "　" ); break; // 床

case WALL: db.Write( "■" ); break; // 壁

case START: db.Write( "Ｓ" ); break; // スタート

case GOAL: db.Write( "Ｇ" ); break; // ゴール

case PLAYER: db.Write( "・" ); break; // 探索位置

default: db.Write( "？" ); break; // 念のため

}

}

}

}

// 改行

row++;

db.SetCursorPos( 0, row );

}

if( mode == FAST )Sleep( 10 ); // 1/1000秒待つ

else Sleep( 500 ); // 1/2秒待つ

db.Swap(); // バッファをスワップして画面に表示

}

// 上下左右どれか隣にゴールがあるか

bool CheckAround( int posY, int posX ){

if( map[posY][posX-1] == GOAL || map[posY][posX+1] == GOAL ||

map[posY-1][posX] == GOAL || map[posY+1][posX] == GOAL ){

if( mode != NODRAW )Sleep( 2000 ); // 2秒待つ

return true;

}

else return false;

}

// 探索

void Root( int y, int x ){

if( clear ) return;

if( map[y][x] == FLOOR ) {

map[y][x] = PLAYER;

Draw( y, x );

}

// 現在位置の周りにゴールがない場合

if( CheckAround( y, x ) ){

clear = true;

return;

}else{

// 上下左右それぞれ進める場所があれば進む

if( map[y-1][x] == FLOOR ){

Root( y-1, x );

}

if( map[y+1][x] == FLOOR ){

Root( y+1, x );

}

if( map[y][x-1] == FLOOR ){

Root( y, x-1 );

}

if( map[y][x+1] == FLOOR ){

Root( y, x+1 );

}

}

}

// メイン関数

int main(){

db.Write( "モードを選択\n" );

db.Write( "0:描画なし（ファイル出力のみ）\n1:高速描画モード\n2:通常モード\n " );

db.Swap();

int num;

cin >> num;

switch( num ){

case 0:

mode = NODRAW;

db.Write( "描画なしで実行します。\n" );

break;

case 1:

mode = FAST;

db.Write( "高速描画モードで実行します。\n" );

break;

case 2:

mode = NORMAL;

db.Write( "通常モードで実行します。\n" );

break;

default:

mode = NORMAL;

db.Write( "入力が不正です。通常モードで実行します。\n" );

break;

}

db.Swap();

Sleep( 1000 ); // 1秒待つ

// ファイル読み込み

ifstream ifs;

ifs.open( "maze.txt" );

if( ifs.fail() ){

cout << "ファイルを開けませんでした。" << endl;

return -1;

}

string str; // 読み込んだ文字列を格納するバッファ

int x = 0, y = 0; // マップを生成するマス目の座標

int sx, sy, gx, gy; // スタート、ゴール位置の座標

while( getline( ifs, str ) ){ // 1行ずつ読み込み、ファイルの最後までループ

for( int i = 0; i < str.size(); i++ ){ // 1文字ずつ解析

switch( str[i] ){

case '0': // 床

map[y][x] = FLOOR;

break;

case '1': // 壁

map[y][x] = WALL;

break;

case '2': // スタート

map[y][x] = START;

sx = x; sy = y;

break;

case '3': // ゴール

map[y][x] = GOAL;

gx = x; gy = y;

break;

default:

x--; // 空白を読み込んだ場合一度左へ戻る

}

x++; // 右のマスへ進む

}

x = 0; // 改行

y++;

}

// スタート位置から探索開始

Root( sy, sx );

// 結果テキスト出力

ofstream ofs;

ofs.open( "ans.txt" );

if( ofs.fail() ){

cout << "ファイルが開けません" << endl;

return -1;

}

// マップを片っ端から書き出す

for( int y = 0; y < sizeY; y++ ){

for( int x = 0; x < sizeX; x++ ){

ofs << map[y][x];

ofs << " ";

}

ofs << endl; // 1行終わったら改行

}

ofs.close();

return 0;

}

プログラム２：経路探索プログラム main.cpp

// DoubleBufferConsole.h

// コンソールをダブルバッファリングで描画するクラス

#ifndef \_DOUBLEBUFFERCONSOLE\_H

#define \_DOUBLEBUFFERCONSOLR\_H

#include <Windows.h>

#include <string>

class DoubleBufferConsole

{

private:

bool m\_bSwapped; // スワップしたかどうか

HANDLE m\_handle1; // コンソールハンドル1

HANDLE m\_handle2; // コンソールハンドル2

public:

// コンストラクタ

DoubleBufferConsole();

// デストラクタ

~DoubleBufferConsole();

// カーソル位置設定

void SetCursorPos( int, int );

// char文字列描画

void Write( const char\* );

// ワイドchar文字列描画

void Write( const wchar\_t\* );

// string文字列描画

void Write( const std::string& );

// ワイドstring文字列描画

void Write( const std::wstring& );

// バッファ入れ替え

void Swap();

};

#endif

プログラム３：経路探索プログラム　DoubleBufferConsole.h

// DoubleBufferConsole.cpp

// コンソールをダブルバッファリングで描画するクラス

#include "DoubleBufferConsole.h"

// コンストラクタ

DoubleBufferConsole::DoubleBufferConsole() : m\_bSwapped( false )

{

// バッファ1作成

m\_handle1 = CreateConsoleScreenBuffer(

GENERIC\_READ | GENERIC\_WRITE,

0, NULL, CONSOLE\_TEXTMODE\_BUFFER, NULL );

// バッファ2作成

m\_handle2 = CreateConsoleScreenBuffer(

GENERIC\_READ | GENERIC\_WRITE,

0, NULL, CONSOLE\_TEXTMODE\_BUFFER, NULL );

// バッファ1を使用

SetConsoleActiveScreenBuffer( m\_handle1 );

CONSOLE\_CURSOR\_INFO info;

info.dwSize = 1;

info.bVisible = FALSE;

SetConsoleCursorInfo( m\_handle1, &info );

SetConsoleCursorInfo( m\_handle2, &info );

}

// デストラクタ

DoubleBufferConsole::~DoubleBufferConsole()

{

CloseHandle( m\_handle1 );

CloseHandle( m\_handle2 );

}

// カーソル位置設定

void DoubleBufferConsole::SetCursorPos( int x, int y )

{

COORD pos;

pos.X = x;

pos.Y = y;

SetConsoleCursorPosition( m\_bSwapped ? m\_handle1 : m\_handle2, pos );

}

// char文字列描画

void DoubleBufferConsole::Write( const char\* p )

{

DWORD len = strlen( p );

WriteConsoleA( m\_bSwapped ? m\_handle1 : m\_handle2, p, len, &len, 0 );

}

// ワイドchar文字列描画

void DoubleBufferConsole::Write( const wchar\_t\* p )

{

DWORD len = wcslen( p );

WriteConsoleW( m\_bSwapped ? m\_handle1 : m\_handle2, p, len, &len, NULL );

}

// string文字列描画

void DoubleBufferConsole::Write( const std::string& str )

{

DWORD len = str.size();

WriteConsoleA( m\_bSwapped ? m\_handle1 : m\_handle2, str.c\_str(), len, &len, NULL );

}

// ワイドstring文字列描画

void DoubleBufferConsole::Write( const std::wstring& str )

{

DWORD len = str.size();

WriteConsoleA( m\_bSwapped ? m\_handle1 : m\_handle2, str.c\_str(), len, &len, NULL );

}

// バッファ入れ替え

void DoubleBufferConsole::Swap()

{

if( ( m\_bSwapped = !m\_bSwapped ) )

SetConsoleActiveScreenBuffer( m\_handle2 );

else

SetConsoleActiveScreenBuffer( m\_handle1 );

}

プログラム４：経路探索プログラム　DoubleBufferConsole.cpp

1. 実行結果

プログラムを実行した結果として出力されたテキストファイルmaze.txtとans.txtのスクリーンショットを以下に示す。

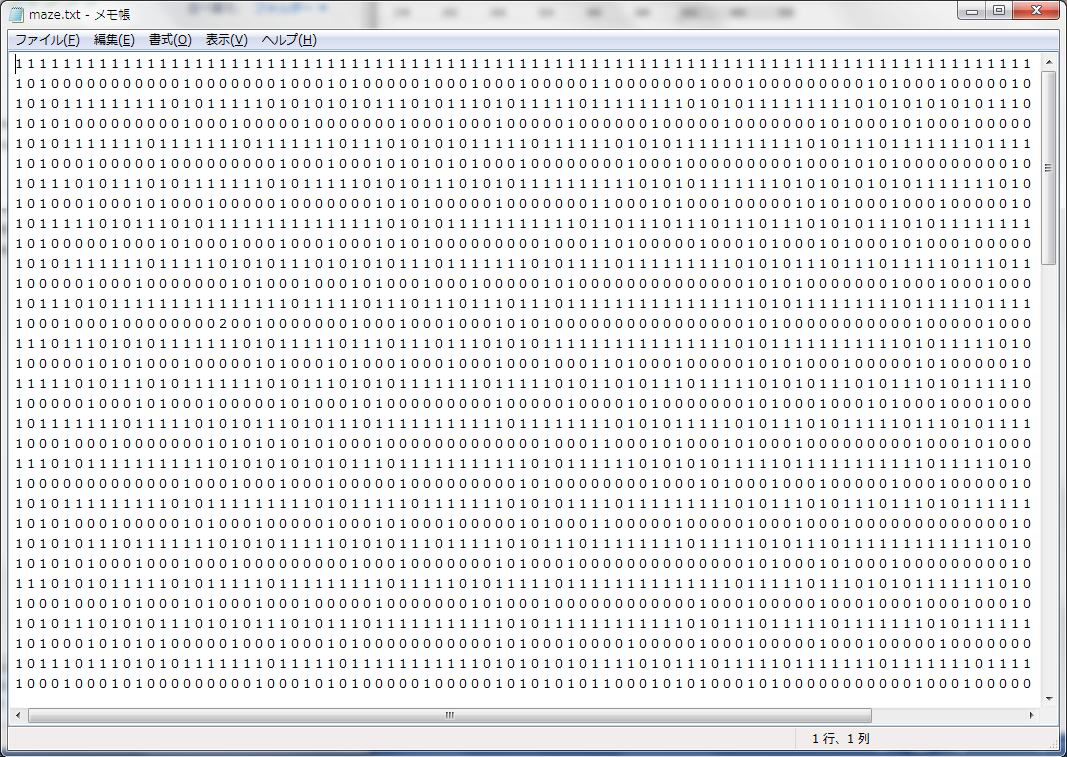


図２：maze.txtのスクリーンショット

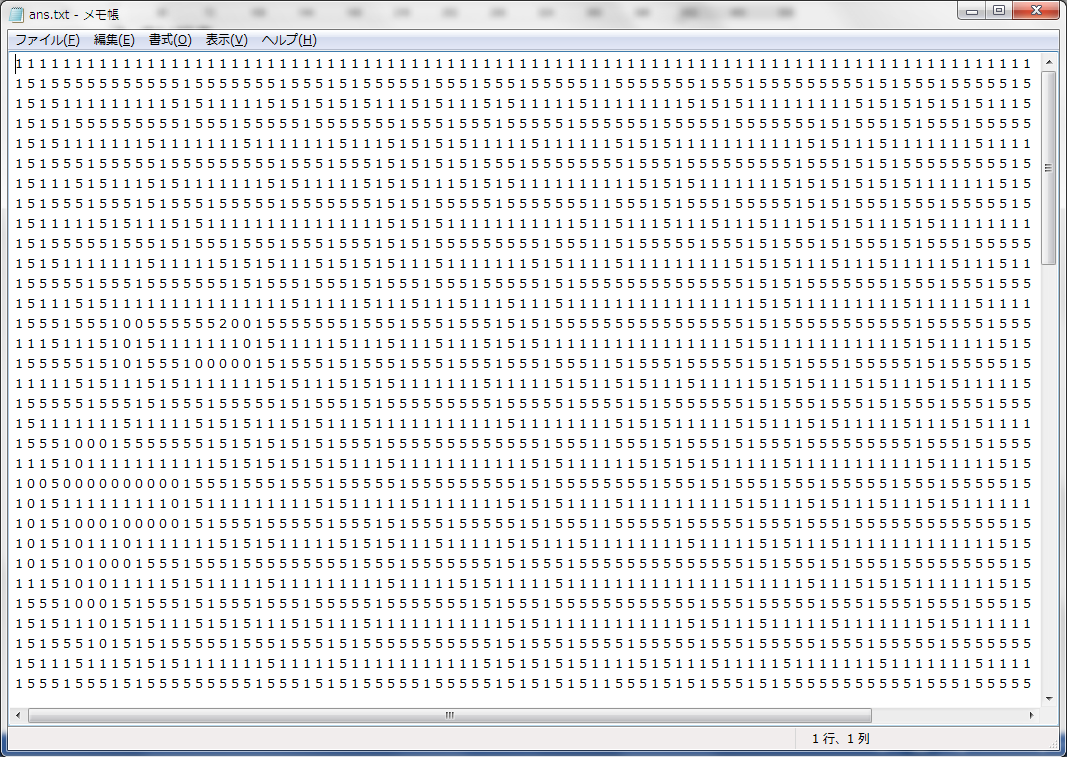


図３：ans.txtのスクリーンショット

迷路探索中のコンソール画面を以下に示す。

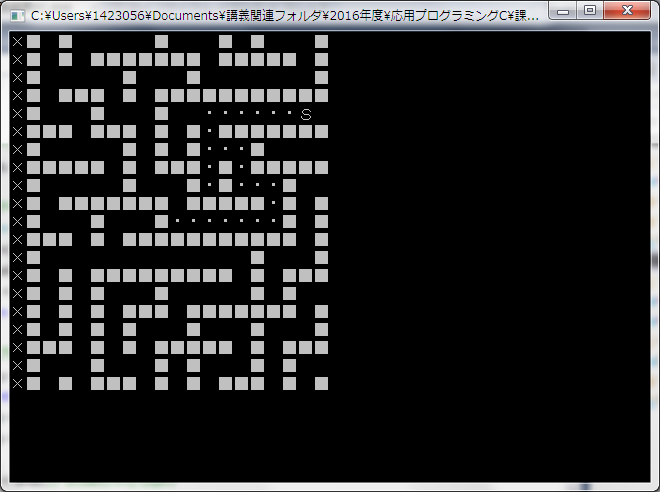


図４：迷路探索プログラム実行画面

迷路外の範囲は×を表示するようにした。

　高速描画モードでは1ミリ秒間隔で、通常モードでは500ミリ秒間隔で移動の様子を描画している。ちなみに、通常モードの場合ゴールにたどり着くまで約30分ほどかかった。経過時間は迷路の形状にもよるが、眺める分には飽きる程の時間である。

1. 考察

迷路探索では、上が床なら上に、上が壁で下が床なら下に、上下がだめなら左、それらが駄目なら右の方向に進むという処理を再帰関数でひたすら繰り返すだけであまり工夫ができなかった。

探索の条件でスタート位置とゴールの位置の座標を利用できるとあったが、どのように利用すべきか良いアイデアが浮かばなかった。

１つ浮かんだアイデアとして、上下左右どの方向に進むか決定するとき、現在の探索位置からゴールの存在する方向によって上下左右どの方向に進むかの確率を変更すればより速くゴールにたどり着けると考えたが、複雑な迷路の場合ゴールの方向に道がつながっていない場合もあると考えられるので高速化には繋がらなかった。

また、道幅が１マスの迷路であれば道をまっすぐ進むだけで済むが、道幅が２マス以上ある場合では行き止まりに当たっても道の間で曲がった部分から行き止まりまで繰り返し探索してしまうので、さらに時間がかかるものとなっている。

迷路の生成と探索は以下のWebサイトを参考にした。

c言語で迷路

<http://studio.s1.xrea.com/prog/maze.html>

参照日：2016/6/29