/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* lifegame.c -- utility functions for Convoy's Game of Life. \*

\* \*

\* Author. TODO.\*

\* \*

\* Purpose. function definitions for the game. \*

\* \*

\* Usage. \*

\* Call the functions in main() to evolve the world. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include "lifegame.h"

/\* hard-coded world size \*/

#define WORLDHEIGHT 10

#define WORLDWIDTH 10

/\* character representations of cell states \*/

#define CHAR\_ALIVE '\*'

#define CHAR\_DEAD ' '

/\* current cell states of the world \*/

static int world[WORLDHEIGHT][WORLDWIDTH];

/\* next generation cell states \*/

static int nextstates[WORLDHEIGHT][WORLDWIDTH];

/\*

\* initialize the world to a hard-coded pattern, and

\* reset all the cells in the next generation to DEAD

\*/

void initialize\_world(void) {

int i, j;

for (i = 0; i < WORLDHEIGHT; i++)

for (j = 0; j < WORLDWIDTH; j++)

world[i][j] = nextstates[i][j] = DEAD;

/\* pattern "glider" \*/

world[1][2] = ALIVE;

world[3][1] = ALIVE;

world[3][2] = ALIVE;

world[3][3] = ALIVE;

world[2][3] = ALIVE;

}

int get\_world\_width(void) {

return WORLDWIDTH;

}

int get\_world\_height(void) {

return WORLDHEIGHT;

}

int get\_cell\_state(int x, int y) {

if (x < 0 || x >= WORLDWIDTH || y < 0 || y >= WORLDHEIGHT)

return DEAD;

return world[x][y];

}

void set\_cell\_state(int x, int y, int state) {

if (x < 0 || x >= WORLDWIDTH || y < 0 || y >= WORLDHEIGHT) {

fprintf(stderr,"Error: coordinates (%d,%d) are invalid.\n", x, y);

abort();

}

nextstates[x][y] = state;

}

int num\_neighbors(int x, int y) {

int count = 0;

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

for (int j = y-1; j <= y+1; j++)

{

if (!(x < 0 || x >= WORLDWIDTH || y < 0 || y >= WORLDHEIGHT)){

if (!(i == x && j == y)){

if (get\_cell\_state(i, j) == ALIVE){

count++;

}

}

}

}

}

return count;

/\* Hint. Use get\_cell\_state(x,y) \*/

}

int get\_next\_state(int x, int y) {

int neighbours = num\_neighbors(x, y);

if (get\_cell\_state(x,y)==ALIVE)

{

if (neighbours < 2){

return DEAD;

}

else if (neighbours>3){

return DEAD;

}

else if (neighbours == 2 || neighbours == 3){

return ALIVE;

}

}

else

{

if (neighbours == 3){

return ALIVE;

}

else

{

return DEAD;

}

}

}

void next\_generation(void) {

int x, y;

for (x = 0; x < WORLDHEIGHT; x++) {

for (y = 0; y < WORLDWIDTH; y++) {

nextstates[x][y]=get\_next\_state(x,y);

}

}

finalize\_evolution(); /\* called at end to finalize \*/

}

void finalize\_evolution(void) {

int x, y;

for (x = 0; x < WORLDHEIGHT; x++) {

for (y = 0; y < WORLDWIDTH; y++) {

world[x][y] = nextstates[x][y];

nextstates[x][y] = DEAD;

}

}

}

void output\_world(void) {

char worldstr[2\*WORLDWIDTH+2];

int i, j;

worldstr[2\*WORLDWIDTH+1] = '\0';

worldstr[0] = '+';

for (i = 1; i < 2\*WORLDWIDTH; i++)

worldstr[i] = '-';

worldstr[2\*WORLDWIDTH] = '+';

puts(worldstr);

for (i = 0; i <= 2\*WORLDWIDTH; i+=2)

worldstr[i] = '|';

for (i = 0; i < WORLDHEIGHT; i++) {

for (j = 0; j < WORLDWIDTH; j++)

worldstr[2\*j+1] = world[i][j] == ALIVE ? CHAR\_ALIVE : CHAR\_DEAD;

puts(worldstr);

}

worldstr[0] = '+';

for (i = 1; i < 2\*WORLDWIDTH; i++)

worldstr[i] = '-';

worldstr[2\*WORLDWIDTH] = '+';

puts(worldstr);

}



