ESE-3025 Embedded Real Time Operating Systems

LAB 1: GAME OF LIFE

GROUP No. 2

Group Members	Student ID
Christy Rachel Philip	C0765535
Fahad Rahman	C0769871
James M Chacko	C0777192
Premil Presannan	C0777191
Vy Nguyen	C0776242

INTRODUCTION

In this, we are doing the "Game of Life" as a pthread implementation in RTOS. For this, we are cloning the following repository as reference.

https://github.com/takiszourntos/teaching/tree/master/lambton/2020/summer/ese3025/week_2/project/proj_pthreads_template

DESCRIPTION

Here we updated these three pthread functions,

1. size_t countLiveNeighbours(size_t row, size_t col)
2. void updateCell(size_t r, size_t c)
3. void* updateCommFunc(void *param)

1. **size_t countLiveNeighbours(size_t row, size_t col)**

This function is used to count the number of living cells in the neighbourhood. For that we are considering the neighbouring 9 cells together. To avoid the counting of next cells, like the 10th or 11th one, we are using the modulus function, which takes the remainder.

```
size_t countLiveNeighbours(size_t row, size_t col) {
      size_t cell_count = 0;

      for (size_t i = row - 1; i <= row + 1; i++) {
            for (size_t j = col - 1; j <= col + 1; j++) {
                  // To make sure that you don't count the cell whose
neighbours are counted
                  if (i != 0 && j != 0) {
                        // using this we can find the number of cells
alive.
                        cell_count = cell_count + (size_t)env[(size_t) (row
+ i + config_NE) % config_NE][(size_t) (col + j + config_ME) % config_ME];
                  }
            }
      }
      cell_count = cell_count - (env[row][col]==live ? 1 : 0);
      return cell_count;
}
```

2. void updateCell(size_t r, size_t c)

This function features Conway's rules:

1. if a cell is dead but surrounded by exactly three live neighbours, it sprouts to life (birth)
2. if a cell is live but has more than 3 live neighbours, it dies (overpopulation)
3. if a cell is live but has fewer than 2 live neighbours, it dies (underpopulation)
4. all other dead or live cells remain the same to the next generation (i.e., a live cell must have exactly three neighbours to survive)

```
void updateCell(size_t r, size_t c) {
cell_t state_cell = env[r][c];
size_t live_neighbours = countLiveNeighbours(r, c);

if (state_cell == 0 && (live_neighbours == 3)) {
     update_env[r][c] = state_cell = live;
} else if (state_cell == 1 && (live_neighbours < 2)) {
     update_env[r][c] = state_cell = dead;
} else if (state_cell == 1 && (live_neighbours > 3)) {
     update_env[r][c] = state_cell = dead;
} else {
     update_env[r][c] = state_cell;
}
}
```

3. void* updateCommFunc(void *param)

This function updates all the cells according to the rules. For that we are calling the "updateCell" function that we defined above.

```
void* updateCommFunc(void *param)
{
     threadID_t *threadoffsets = (threadID_t *) param;
     //size_t thread_row = threadoffsets->row;
     //size_t thread_col = threadoffsets->col;
     while(1)
     {
          if(reproduction_flag){
               threadID_t *var = param;
               size_t i_0 = var->row;
               size_t j_0 = var->col;
               size_t a = i_0 *config_NC;
               size_t b = j_0 *config_MC;
```

```
                  for(size_t i =0; i != config_NC; ++i)
                  {
                        for(size_t j=0; j!= config_MC; ++j)
                        {
                              updateCell(i+a,j+b);
                        }
                  }

            }
      }
}
```

After these, compile the code in terminal of host machine as follows,

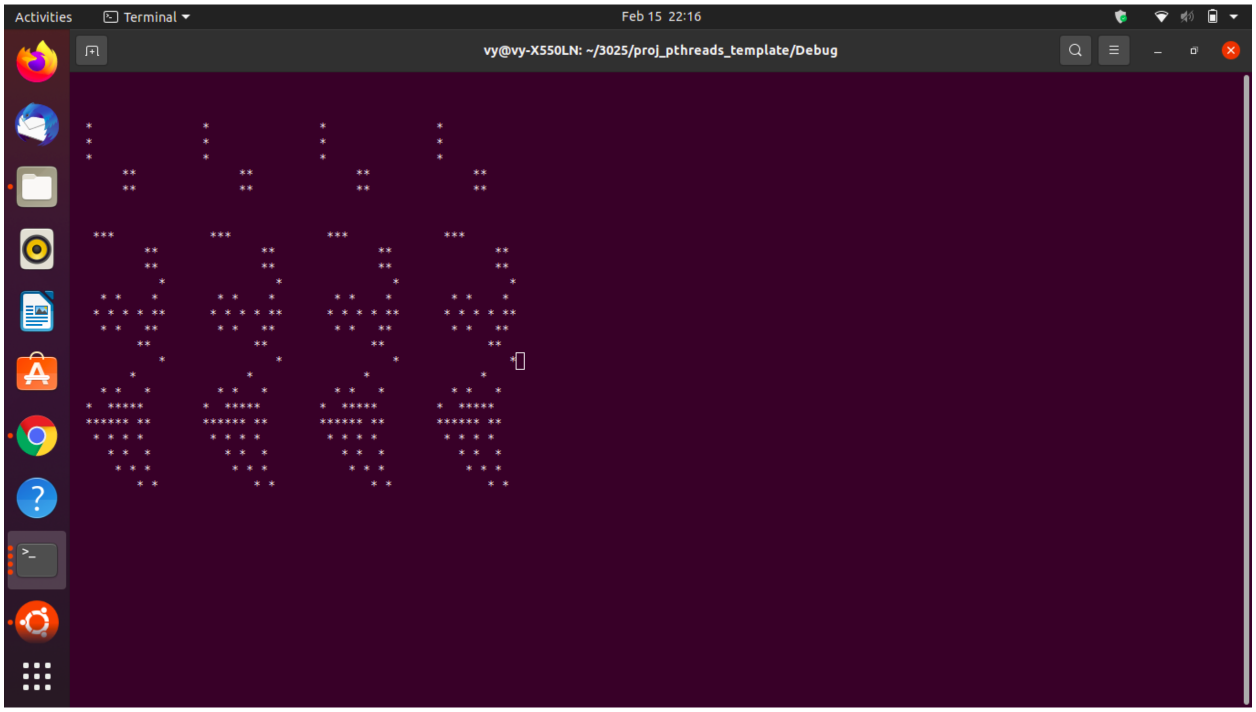
```
vy@vy-X550LN:~/3025$ cd proj_pthreads_template
vy@vy-X550LN:~/3025/proj_pthreads_template$ cd Debug
vy@vy-X550LN:~/3025/proj_pthreads_template/Debug$ make
Building file: ../source/cells.c
Invoking: GCC C Compiler
gcc -I"/home/vy/eclipse-workspace/proj_pthreads_template/includes" -O0 -g3
-Wall -c -fmessage-length=0 -MMD -MP -MF"source/cells.d"
-MT"source/cells.o" -o "source/cells.o" "../source/cells.c"
../source/cells.c: In function 'updateCommFunc':
../source/cells.c:199:14: warning: unused variable 'threadoffsets'
[-Wunused-variable]
  199 |  threadID_t *threadoffsets = (threadID_t *) param;
      |              ^~~~~~~~~~~~~
Finished building: ../source/cells.c

Building file: ../source/display.c
Invoking: GCC C Compiler
gcc -I"/home/vy/eclipse-workspace/proj_pthreads_template/includes" -O0 -g3
-Wall -c -fmessage-length=0 -MMD -MP -MF"source/display.d"
-MT"source/display.o" -o "source/display.o" "../source/display.c"
Finished building: ../source/display.c

Building file: ../source/gol.c
Invoking: GCC C Compiler
gcc -I"/home/vy/eclipse-workspace/proj_pthreads_template/includes" -O0 -g3
```

```
-Wall -c -fmessage-length=0 -MMD -MP -MF"source/gol.d" -MT"source/gol.o" -o
"source/gol.o" "../source/gol.c"
Finished building: ../source/gol.c

Building file: ../.metadata/.plugins/org.eclipse.cdt.make.core/specs.c
Invoking: GCC C Compiler
gcc -I"/home/vy/eclipse-workspace/proj_pthreads_template/includes" -O0 -g3
-Wall -c -fmessage-length=0 -MMD -MP
-MF".metadata/.plugins/org.eclipse.cdt.make.core/specs.d"
-MT".metadata/.plugins/org.eclipse.cdt.make.core/specs.o" -o
".metadata/.plugins/org.eclipse.cdt.make.core/specs.o"
"../.metadata/.plugins/org.eclipse.cdt.make.core/specs.c"
Finished building: ../.metadata/.plugins/org.eclipse.cdt.make.core/specs.c

Building target: proj_pthreads_template
Invoking: GCC C Linker
gcc  -o "proj_pthreads_template"  ./source/cells.o ./source/display.o
./source/gol.o  ./.metadata/.plugins/org.eclipse.cdt.make.core/specs.o
-lpthread -lncurses
Finished building target: proj_pthreads_template
```

Execution

```
vy@vy-X550LN:~/3025/proj_pthreads_template/Debug$ cat seed_input_32_x_16.txt |
./proj_pthreads_template

initializing environment...
     ... loading template community from stdin
     ... done.
     ... creating communities
     ... transferring block (1, 1)
     ... transferring block (1, 2)
     ... transferring block (1, 3)
     ... transferring block (1, 4)
     ... done.

creating threads...

initializing display...
```

OUTPUT SCREENSHOT



```
*               *               *               *
*               *               *               *
*               *               *               *
     **              **              **              **
     **              **              **              **


 ***             ***             ***             ***
        **              **              **              **
        **              **              **              **
          *               *               *               *
  * *    *        * *    *        * *    *        * *    *
 * * * * **      * * * * **      * * * * **      * * * * **
  * *   **        * *   **        * *   **        * *   **
       **              **              **              **
          *               *               *               *
      *               *               *               *
  * *   *         * *   *         * *   *         * *   *
*  *****        *  *****        *  *****        *  *****
****** **       ****** **       ****** **       ****** **
 * * * *         * * * *         * * * *         * * * *
   * *  *          * *  *          * *  *          * *  *
    * * *           * * *           * * *           * * *
       * *             * *             * *             * *
```

- **Analyze the display.c and gol.c for Game of Life (GOL) project through commenting line by line.**

  <u>Display.c</u>

  We are using this function to display the window to show the output.

For that we used the function create_newwin. Using that we are defining the output displaying the window's length, width, and then we divide it like the board.

Using the initDisplay function, we will print the output. And finally we update the generation using the updateDisplay function.

```c
#include "gol_config.h"
#include <unistd.h>
#include <ncurses.h>
#include "display.h"

/*
 * important variables, defined elsewhere
 */
extern cell_t env[config_NE][config_ME];
extern int STARTX;
extern int STARTY;
extern int ENDX;
extern int ENDY;
extern WINDOW *win; // to define the window as win

/*
 * PRIVATE FUNCTIONS
 */
// here we define the size of window
void create_newwin(int height, int width)
{
      win = newwin(height, width, STARTY, STARTX);
      box(win, 0, 0); /* 0, 0 gives default characters
       * for the vertical and horizontal
       * lines */
      wrefresh(win); /* show that box */

      return;
}
```

```c
/*
 * PUBLIC FUNCTIONS
 */
// using this we print the output
void initDisplay(void)
{
      printf("\ninitializing display...\n");
      usleep(2 * config_TL);
      initscr();
      cbreak();
      timeout(TIME_OUT);
      keypad(stdscr, TRUE);
      create_newwin(config_NE, config_ME);
}

// we update the generate function using this
void updateDisplay(void)
{
//    ENDX = COLS - 1;
//    ENDY = LINES - 1;

      int i, j;
      wclear(win);
      for (i = STARTX; i != config_ME; ++i)
            for (j = STARTY; j != config_NE; ++j)
                  if (env[j][i] == live)
                        mvwaddch(win, j, i, CELL_CHAR);
      wrefresh(win);
}
```

Gol.c

This function will help to work with ncurses library

```c
/*
 * gol.c
 *
 *  Created on: May 30, 2020
 *      Author: takis
 */

#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <pthread.h>
#include <unistd.h>
#include <ncurses.h>
#include "gol_config.h"
#include "cells.h"
#include "display.h"

/*
 * global variables
 */
cell_t env[config_NE][config_ME];
cell_t update_env[config_NE][config_ME];
bool reproduction_flag = false; // is high when it's mating season

int STARTX = 0;
int STARTY = 0;
int ENDX = config_ME;
int ENDY = config_NE;
WINDOW *win;
/*
 * main code
 */
int main(void)
{
	pthread_t threadptrs[config_K * config_L]; // our thread handles
	threadID_t threadID[config_K * config_L]; // thread ID
```

```
      // initialize workspace
      initEnvironment();

      // create the threads
      printf("\ncreating threads...\n");
      size_t index;
      for (size_t i = 0; i != config_K; ++i)
      {
            for (size_t j = 0; j != config_L; ++j)
            {
                  index = i * config_L + j; // map (i,j) to an 1-d index
                  threadID[index].row = i;
                  threadID[index].col = j;
                  // the following if condition returns 0 on the successful
creation of each thread:
                  if (pthread_create(&threadptrs[index], NULL,
&updateCommFunc,
                              &threadID[index]) != 0)
                  {
                        printf("failed to create the thread %d\n", (int)
index);
                        return 1;
                  }
            }
      }

      // initialize display with ncurses
      initDisplay();

      unsigned short int ctr = 0;
      while (1)
      {
            reproduction_flag = true;
            usleep(config_TL / 2); // allow new generation to check in
            reproduction_flag = false;
            usleep(config_TL / 2); // put a hold on reproduction to update
display
            if (++ctr == config_TDISP)
            {
                  ctr = 0;
                  updateDisplay();
            }
            copyEnvironment(); // write changes to the environment, env,
from update_env
      }

      // should never arrive here;
      return 1;
}
```

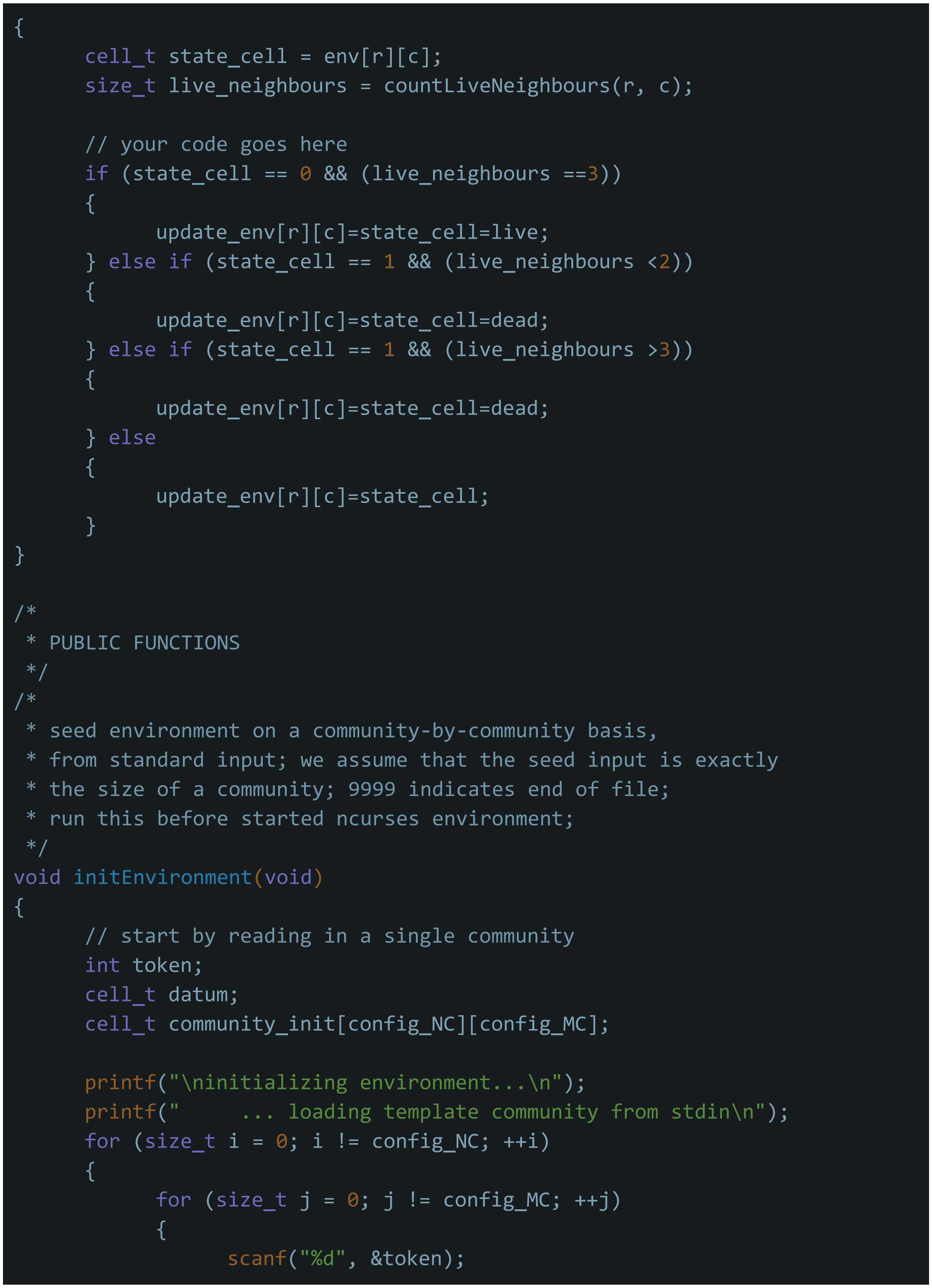
CONCLUSION

In this, we did the “Game of life” as a pthread implementation in RTOS.
Here we used a pthread and ncurses library to run it. Using these we can take output from eclipse, linux terminal and thereby in Beaglebone Black also.

APPENDIX

Cells.c

```c
/*
 * cells.c
 *
 *  Created on: May 30, 2020
 *      Author: takis
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "gol_config.h"
#include "cells.h"

/*
 * declare important variables (defined in main file as global variables)
 */
//extern cell_t **env;
//extern cell_t **update_env;
extern cell_t env[config_NE][config_ME];
extern cell_t update_env[config_NE][config_ME];
extern bool reproduction_flag;

/*
 * PRIVATE FUNCTIONS
 */

/*
 * transfer a single community identified by the block-pair (iT,jT) to env
and
 * update_env using data_init[][]
 */
void transferCommunity(size_t iT, size_t jT,
            const cell_t data_init[config_NC][config_MC])
{
      size_t i_0 = iT * config_NC;
      size_t j_0 = jT * config_MC;

      printf("     ... transferring block (%d, %d)\n", (int) (iT + 1),
                  (int) (jT + 1));
      // copy this community to each community in env to initialize it
      for (size_t i = 0; i != config_NC; ++i)
      {
            for (size_t j = 0; j != config_MC; ++j)
            {
                  env[i_0 + i][j_0 + j] = update_env[i_0 + i][j_0 + j] =
                              data_init[i][j];
            }
      }
}

/*
 * function counts the number of live neighbours of a cell located
 * at row r and column c of the env array
 *
 * for reference, neighbours are designated as follows:
 *             a b c
 *             d X e
 *             f g h
 *
 *
 */
size_t countLiveNeighbours(size_t row, size_t col)
{
      size_t cell_count = 0;

      // your code goes here -- watch those boundary conditions, e.g., col
posn of 0 or 127, or
      // a row posn of 0 or 31
      for(size_t i = row-1; i<= row +1; i++)
      {
            for(size_t j = col -1; j<= col+1; j++)
            {
                  if (i !=0 && j!=0)
                  {
                        cell_count = cell_count +(size_t) env[(i +
config_NE) % config_NE][(j + config_ME) % config_ME];
                        //cell_count=cel_count +
(size_t)env[i-1][j-1]+(size_t)env[i-1][j];
                  }
            }
      }
      /*for (neighbour_t i=a_posn; i <= h_posn; ++i)
      {

            switch (i)
                  {
                    case a_posn: ;
                    case b_posn: ;
                    case c_posn: ;
                    case d_posn: ;
                    case e_posn: ;
                    case f_posn: ;
                    case g_posn: ;
                    case h_posn: ;
            }

            // handle boundary conditions
      }
*/
      cell_count=cell_count - env[row][col];
      return cell_count;
}

/*
 * update cell located at row r and column c in env (indicated by X):
 *
 *                      a b c
 *                      d X e
 *                      f g h
 *
 * with nearest neighbours indicated as shown from a, b, ..., h.
 *
 * this function features Conway's rules:
 *          - if a cell is dead but surrounded by exactly three live
neighbours, it sprouts to life (birth)
 *          - if a cell is live but has more than 3 live neighbours, it
dies (overpopulation)
 *          - if a cell is live but has fewer than 2 live neighbours, it
dies (underpopulation)
 *          - all other dead or live cells remain the same to the next
generation (i.e., a live cell must
 *            have exactly three neighbours to survive)
 *
 */
void updateCell(size_t r, size_t c)
```



```c
{
      cell_t state_cell = env[r][c];
      size_t live_neighbours = countLiveNeighbours(r, c);

      // your code goes here
      if (state_cell == 0 && (live_neighbours ==3))
      {
            update_env[r][c]=state_cell=live;
      } else if (state_cell == 1 && (live_neighbours <2))
      {
            update_env[r][c]=state_cell=dead;
      } else if (state_cell == 1 && (live_neighbours >3))
      {
            update_env[r][c]=state_cell=dead;
      } else
      {
            update_env[r][c]=state_cell;
      }
}

/*
 * PUBLIC FUNCTIONS
 */
/*
 * seed environment on a community-by-community basis,
 * from standard input; we assume that the seed input is exactly
 * the size of a community; 9999 indicates end of file;
 * run this before started ncurses environment;
 */
void initEnvironment(void)
{
      // start by reading in a single community
      int token;
      cell_t datum;
      cell_t community_init[config_NC][config_MC];

      printf("\ninitializing environment...\n");
      printf("     ... loading template community from stdin\n");
      for (size_t i = 0; i != config_NC; ++i)
      {
            for (size_t j = 0; j != config_MC; ++j)
            {
                  scanf("%d", &token);
                  datum = (cell_t) token;
                  community_init[i][j] = datum;
            }
      }
      printf("     ... done.\n");

      printf("     ... creating communities\n");
      // copy this community to each community in env to initialize it
      for (size_t i = 0; i != config_K; ++i)
      {
            for (size_t j = 0; j != config_L; ++j)
            {
                  transferCommunity(i, j, community_init);
            }
      }
      printf("     ... done.\n");

}
/*
 * write changes to the environment, env, from update_env
 */
void copyEnvironment(void)
{
      // copy this community to each community in env to initialize it
      for (size_t i = 0; i != config_NE; ++i)
      {
            for (size_t j = 0; j != config_ME; ++j)
            {
                  env[i][j] = update_env[i][j];
            }
      }
}

/*
 * this function updates all the cells for a thread (corresponding to one
community)
 */
void* updateCommFunc(void *param)
{
      threadID_t *threadoffsets = (threadID_t *) param;
      //size_t thread_row = threadoffsets->row;
      //size_t thread_col = threadoffsets->col;
      while(1)
      {
            if(reproduction_flag){
                  threadID_t *var = param;
                  size_t i_0 = var->row;
                  size_t j_0 = var->col;
                  size_t a = i_0 *config_NC;
                  size_t b = j_0 *config_MC;
                  for(size_t i =0; i != config_NC; ++i)
                  {
                        for(size_t j=0; j!= config_MC; ++j)
                        {
                              updateCell(i+a,j+b);
                        }
                  }

            }
      }
      // you'll need to make use of updateCell(size_t r, size_t c)

      // your code goes here
}
```

Display.c

```
/*
 * display.c
 *
 *  Created on: May 30, 2020
 *      author: takis
 */

#include "gol_config.h"
#include <unistd.h>
#include <ncurses.h>
#include "display.h"

/*
 * important variables, defined elsewhere
 */
extern cell_t env[config_NE][config_ME];
extern int STARTX;
extern int STARTY;
extern int ENDX;
extern int ENDY;
extern WINDOW *win;

/*
 * PRIVATE FUNCTIONS
 */
void create_newwin(int height, int width)
{
      win = newwin(height, width, STARTY, STARTX);
      box(win, 0, 0); /* 0, 0 gives default characters
       * for the vertical and horizontal
       * lines */
      wrefresh(win); /* show that box */

      return;
}

/*
 * PUBLIC FUNCTIONS
 */
void initDisplay(void)
{
      printf("\ninitializing display...\n");
      usleep(2 * config_TL);
      initscr();
      cbreak();
      timeout(TIME_OUT);
      keypad(stdscr, TRUE);
      create_newwin(config_NE, config_ME);
}

void updateDisplay(void)
{
//    ENDX = COLS - 1;
//    ENDY = LINES - 1;

      int i, j;
      wclear(win);
      for (i = STARTX; i != config_ME; ++i)
            for (j = STARTY; j != config_NE; ++j)
                  if (env[j][i] == live)
```

```
                        mvwaddch(win, j, i, CELL_CHAR);
      wrefresh(win);
}

/*

***************************************************************************
***************** reference
 */
```

Gol.c

```
/*
 * gol.c
 *
 *  Created on: May 30, 2020
 *      Author: takis
 */

#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <pthread.h>
#include <unistd.h>
#include <ncurses.h>
#include "gol_config.h"
#include "cells.h"
#include "display.h"

/*
 * global variables
 */
cell_t env[config_NE][config_ME];
cell_t update_env[config_NE][config_ME];
bool reproduction_flag = false; // is high when it's mating season

int STARTX = 0;
```

```c
int STARTY = 0;
int ENDX = config_ME;
int ENDY = config_NE;
WINDOW *win;
/*
 * main code
 */
int main(void)
{
      pthread_t threadptrs[config_K * config_L]; // our thread handles
      threadID_t threadID[config_K * config_L]; // thread ID

      // initialize workspace
      initEnvironment();

      // create the threads
      printf("\ncreating threads...\n");
      size_t index;
      for (size_t i = 0; i != config_K; ++i)
      {
            for (size_t j = 0; j != config_L; ++j)
            {
                  index = i * config_L + j; // map (i,j) to an 1-d index
                  threadID[index].row = i;
                  threadID[index].col = j;
                  // the following if condition returns 0 on the successful
creation of each thread:
                  if (pthread_create(&threadptrs[index], NULL,
&updateCommFunc,
                              &threadID[index]) != 0)
                  {
                        printf("failed to create the thread %d\n", (int)
index);
                        return 1;
                  }
            }
      }

      // initialize display with ncurses
      initDisplay();

      unsigned short int ctr = 0;
      while (1)
      {
            reproduction_flag = true;
            usleep(config_TL / 2); // allow new generation to check in
            reproduction_flag = false;
            usleep(config_TL / 2); // put a hold on reproduction to update
display
            if (++ctr == config_TDISP)
            {
                  ctr = 0;
                  updateDisplay();
            }
            copyEnvironment(); // write changes to the environment, env,
from update_env
      }

      // should never arrive here;
      return 1;
}
```

Cells.h

```
/*
 * cells.h
 *
 *  Created on: May 30, 2020
 *      Author: takis
 */

#ifndef CELLS_H_
#define CELLS_H_

#include "gol_config.h"
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdbool.h>

/*
```

```
 * functions
 */
void initEnvironment(void);

void copyEnvironment(void);

void* updateCommFunc(void*);

#
endif /* CELLS_H_ */

Display.h
/*
 * display.h
 *
 *  Created on: May 30, 2020
 *      Author: takis
 */

#ifndef DISPLAY_H_
#define DISPLAY_H_

#include <ncurses.h>
#include <stdbool.h>

// window parameters
#define CELL_CHAR '*'
#define TIME_OUT  300

/*
 * functions
 */
void initDisplay(void);

void updateDisplay(void);

#endif /* DISPLAY_H_ */
```

Gol_config.h

```
/*
 * gol_config.h
 *
 *  Created on: May 30, 2020
 *      Author: takis
 */

#ifndef GOL_CONFIG_H_
#define GOL_CONFIG_H_

#include <stdlib.h>

/*
 * "community of cells" (handled by one thread) parameters
 */
#define config_NC       32 // # of cell rows in a community
#define config_MC       16 // # of cell columns in a community

/*
 * overall environment parameters
 */
#define config_K        1 // # of communities "down"
#define config_L        4 // # of communities "across"
#define config_NE             config_K*config_NC // number of environment
rows
#define config_ME             config_L*config_MC // number of environment
columns

/*
 * temporal parameters
 */
#define config_TL       1048576 // microseconds between generation
#define     config_TDISP      1 // number of generations between plots

/*
 * basic cell type
 */
enum cell_enum
{
      dead = 0U, live = 1U
};
typedef enum cell_enum cell_t;

/*
 * thread identifier (in units of community BLOCKS not cells!)
 */
struct threadID_struct
{
      size_t row;
      size_t col;
};
typedef struct threadID_struct threadID_t;

/*
 * a neighbour type for cells... here, X represents the cell:
 *
 *                a b c
 *                d X e
 *                f g h
 *
 */
enum neighbour_enum
{
      a_posn=0U,
      b_posn,
      c_posn,
      d_posn,
      e_posn,
      f_posn,
      g_posn,
      h_posn
};
typedef enum neighbour_enum neighbour_t;


#endif /* GOL_CONFIG_H_ */
```