Structure:

get input: ~~inputfile.txt~~ inputfile, toroidal, silence ncurses, # generation, outputfile

↓

scan the first line of inputfile to get rows & cols.

↓

create two same Universe (set cells dead initially) A and B

A for current generation, B for next generation

↓

check whether successfully populated using uv_populate()

to read the inputfile.

↓ True

set live cells in A using inputfile with uv_live_cell()

the rest are all dead

↓

setup ncurses screen of A

↓

use the 3 Rules to setup a new generation in B based on A.

↓

swap the pointers of A and B. ✓

↓

delete B

↓

end loop

loop within # generation times.

close the screen

↓

output A to the outputfile using uv_print().

code :

1 Universe *uv_create(int rows, int cols, bool toroidal){
    set the ~~para~~ values of parameters to members in the Universe.
}

2. int uv_rows(Universe *u){
    return the member "rows" in Universe
}

3. int uv_cols(Universe *u){
    return the member "cols" in Universe.
}

4. void uvdelete(Universe*u){
    free pointers of each row,
    then free the whole grid.
    free u.
}

5. void uv_live_cell(Univese *u, int r, int c){
    if the ~~cell~~ row-column pair is inbound,
      grid[r][c] = true
}

6. void uv_dead_cell(Universe *u, int r, int c){
    if the row-column pair is in bound,
      grid[r][c] = false
}

7. bool uv_gee_cell(Universe *u, int r, int c){
    if the row-col pair is in bound,
     return grid[r][c]
    else
      return false.
}

8. bool uv_populate (Universe *u, FILE * infile){

bool format = true

int count = 0

while ((not end of file) and (format == true)){

count = number of variables when scaning each line of the file

if (count ≠ 2 or the row-col pair is not in bound){

format = false

}

else {

uv_live_cell

}

}

return format

}

9. int uv_census (Universe *u, int r, int c){

int live = 0

if (flat universe){

loop 1: check row (r-1): if inbound and alive,

live = live +1

loop 2: check row (r) : . . . . . ε

live = live +1

loop 3: check row (r+1): . . . . .

live ++

}

else (toroidal universe){

loop 1: check row (r-1) : if inbound & alive, live ++

if not in bound, ~~neighbor_position~~ new_neighbor_position and alive

live ++

loop 2: check row (r) . . . .

live ++

loop 3: check row (r+1) . . . .

live ++

}

return live

}

flat

r-1, c-1	r-1, c	r-1, c+1 ← loop 1
r, c-1	r, c	r, c+1 ← loop 2
r+1, c-1	r+1, c	r+1, c+1 ← loop 3.

toroidal:

neighbor_position : (r_p, c_p)

new_neighbor_position : (r_n, c_n)

if r_p < 0, r_n = (uv_rows(u) + r_p) % uv_rows(u)

if r_p ≥ uv_rows(u), r_n = (uv_rows(u) − r_p) % uv_rows(u)

same as c_n

10.
```
void uv_print( Universe *u, FILE *outfile ){
    for ( int row=0 ; row < uv_rows(u) ; row ++){
        for (int col=0 ; col < uv_cols(u) ; col ++ ){
            if (grid[row, col] == true){
                print "O" in outfile
            }
            else {
                print "." in outfile
            }
        }
        print "\n" in outfile
    }
}.
```