Rafał Kubas

1.Task topic

Write a program that generates random labyrinths based on a square grid. The initial grid is empty. Then a random "wall" between two squares is built, so that no area is definitely closed. This process is repeated until no more walls can be built. The result should be displayed on the screen. Then the program should allow user to find a way through this labyrinth. Details to be discussed.

2.Project analysis

First I had to figure out easiest way to create at least simple maze. And the idea is as follows: we take few cells of table(every second vertically and horizontally (`for(x=1; x<ms-1; x+=2) {for(y=1; y<ms-1; y+=2)`) then by using randomized numbers we create tunnel from mentioned cell. I used floodfill algorithm to check if there is possible way from randomized start to finish. Algorithm is simple, changes all same values in neighborhood for value set in procedure.

3.External specification

To run the program there is no required additional file. There is only need to specify size of maze and frequency of tunnels.

4.Internal specification

`void tabprint(int size, int **tab)`- procedure for printing 2D array

`if(tab[i][j]==1){printf("%c", blank); }`- in mentioned procedure we wanted to make maze looks more realistic, so instead of number I used other values from ASCII

`void floodfill(int **tab, int size, int posx, int posy, int prevV, int newV)` //procedure of floodfilling algorithm
{ //requires 2D array, size of it, start position(x and y coordinates), previous value of actual position

`int **MazeTable;` //Creating dynamic two-dimensional array used to make represent maze

`int **PathTable;` //Creating dynamic two-dimensional array used for testing if it's possible to walk through

`srand(time(NULL));`//Random function for creating random start/end

`int possible=0;` //value as a condition of while loop (main loop for creating maze possible to finish)

`while(possible==0)` //till finish is not accessible from start position this was main loop of creating maze. If there was no way to reach finish from start, possible wasn't increased then loop was still working.

Creating of tunnels was described in comment section in source code.

Examplary generating start/finish

```
case 1:
start_x=rand() % (ms-1)+1;
start_y=1;
end_x=rand() % (ms-1)+1;
end_y=ms-2;
                        break;
```

Here start is on the top of array, and finish on the bottom.

`void fill_val(int **tab, int size,  int posx, int posy, int newV)` procedure which was recalling floodfill and was taking previous value of cell.

5.Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <time.h>
#include <string.h>



int wall=219; //value used to printing our array, it represents wall in
our maze
int blank=255; //value used to represent empty space in maze

void tabprint(int size, int **tab) //procedure which is printing 2
dimensional array(in our case maze constructed from 1s and 0s
{ //requires size of array
 for(int i=0; i<size; i++)
                {
                for(int j=0; j<size; j++)
                {


                if(tab[i][j]==0)
                    {
                        printf("%c", blank); //If value of 'cell' equals
0 it means that there is possible way (no wall)
                        //so to show it, we were printing value of
'blank' as char
                    }
                if(tab[i][j]==1)
                {
                    printf("%c", wall);  //same situation but this time
for walls
                }
                 if(tab[i][j]==11)
                {
                    printf("F");  //Shows on main array (MazeTable) where
is finnish
                }
                  if(tab[i][j]==2)
                {
                    printf("#"); //Used for floodfill algorithm
                }

                   if(tab[i][j]==10)
                {
                    printf("S"); //Shows where is start
                }
                }

                printf("\n");
                }
}



void floodfill(int **tab, int size, int posx, int posy, int prevV, int
newV) //procedure of floodfilling algorithm
```

```
{ //requires 2D array, size of it, start position(x and y coordinates),
previous value of actual position
    // Also new Value which was needed to replace previous one)
    if (posx < 0 || posx >= size || posy < 0 || posy >= size) //Limits,
cases which cannot happen
        return;
    if (tab[posx][posy] != prevV)
        return;

    // Replace the value
   tab[posx][posy] = newV;

    // Recurence calling for east, north, south, west (4 possible ways of
flooding)
    floodfill(tab, size, posx+1, posy, prevV, newV);
    floodfill(tab, size, posx, posy+1, prevV, newV);
    floodfill(tab, size, posx, posy-1, prevV, newV);
    floodfill(tab, size, posx-1, posy, prevV, newV);

}
// It finds the previous value on (x, y)
// calls floodfill()
void fill_val(int **tab, int size,  int posx, int posy, int newV)
{
    int prevV = tab[posx][posy];
        floodfill(tab, size, posx, posy-1, prevV, newV);
}

int main ()
{ //STEP 1
     //Getting size of maze from the user
        printf("Size of square maze:");
        int ms; scanf("%d", &ms); printf("\n");
        printf("Frequently of connections\nLess creates more interesting
mazes:");
        int fc; scanf("%d", &fc); printf("\n");


          int **MazeTable; //Creating dynamic two-dimensional array used
to make represent maze
          int **PathTable; //Creating dynamic two-dimensional array used
for testing if it's possible to walk through
            PathTable = (int **)malloc(ms * sizeof(int*));
            MazeTable = (int **)malloc(ms * sizeof(int*));
            for (int i = 0; i < ms; i++)   //allocating memory for every
array
                {
                MazeTable[i] = (int *)malloc(ms * sizeof(int));
                } //allocating memory in every "cell"

        for(int i=0; i<ms; i++)
                {
                for(int j=0; j<ms; j++)
                {
                MazeTable[i][j]=1; //fullfiling MazeTable with walls
                //because algorithm of creating is based on "digging
tunnels" not putting walls
                }
```

```
                }

                for (int i = 0; i < ms; i++)     //allocating memory for
every array
                {
                PathTable[i] = (int *)malloc(ms * sizeof(int));
                } //allocating memory in every "cell"

        for(int i=0; i<ms; i++)
                {
                for(int j=0; j<ms; j++)
                {
                PathTable[i][j]=1; //same for table of flooding
                }
                }

srand(time(NULL));//Time for creating random start/end positions - 4
cases
int x,y;

int possible=0; //value as a condition of while loop (main loop for
creating maze possible to finish)
int start_x=0, start_y=0, end_x=0, end_y=0; //start/end coordinates
int c;
int des_poss;


while(possible==0) //till finish is not accessible from start position
{


        for(int i=0; i<ms; i++)
                {
                for(int j=0; j<ms; j++)
                {
                MazeTable[i][j]=1; //overwriting is used after first
execution of loop
                }
                }




//generating

  for(x=1; x<ms-1; x+=2)
    {for(y=1; y<ms-1; y+=2)
    {
        MazeTable[x][y] = 0;        // empty cell (we start creating from
them)
        c = 0;                          // number of corridors
        if(rand() % fc == 0)
        {
            if(x > 1)
              MazeTable[x-1][y] = 0; // left-side corridor
            c++;                        // increasing number of corridors
        }
        if(rand() % fc == 0)
```

```
        {
            if(y > 1)
              MazeTable[x][y-1] = 0;  // upward corridor
            c++;                          // increasing number of
corridors
        }
        if(c == 0)                      // If corridor wasn't created
        {                               // it will be created from left
side or from up
            if(rand() % fc)
            {
                if(x > 1) MazeTable[x-1][y] = 0;
            }
            else
            {
                if(y > 1) MazeTable[x][y-1] = 0;
            }
        }
    }

}



while(MazeTable[start_x][start_y]==1&&MazeTable[end_x][end_y]==1 ||
PathTable[start_x][start_y]==0&&PathTable[end_x][end_y]==0)
{ //There will be randomized start and end coordinates, 4 case (they're
always on opposite sides)
    des_poss= rand() % 4;
switch (des_poss)   //there are saved cells for secure
{
case 0:
start_x=1;
start_y=rand() % (ms-1)+1;
end_x=ms-2;
end_y=rand() % (ms-1)+1;
                            break;

case 1:
start_x=rand() % (ms-1)+1;
start_y=1;
end_x=rand() % (ms-1)+1;
end_y=ms-2;
                            break;
case 2:
start_x=ms-2;
start_y=rand() % (ms-1)+1;
end_x=1;
end_y=rand() % (ms-1)+1;
                            break;
case 3:
start_x=rand() % (ms-1)+1;
start_y=ms-2;
end_x=rand() % (ms-1)+1;
end_y=1;
                            break;

}
```

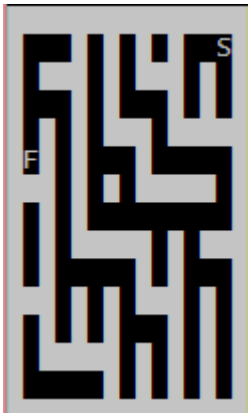
```
}


MazeTable[start_x][start_y]=10;  //For start and end there are other
values than 0s and 1s
MazeTable[end_x][end_y]=11;


        for(int i=0; i<ms; i++)
                {
                for(int j=0; j<ms; j++)
                {
                PathTable[i][j]=MazeTable[i][j];  //copying map of
MazeTable to PathTable
                }
                }


PathTable[start_x][start_y]=0; //But for start and end on pathtable it
was better to make same values as for empty spaces
PathTable[end_x][end_y]=0;


fill_val(PathTable, ms,  start_x, start_y, 2); //proceedure of
floodfilling on path table, with coordinates start_x, start_y, and
replacing value of cells with 2



if(PathTable[end_x][end_y]) //if PathTable is not equal 0, then we can
assume that finish is reachable from the start so, by increasing variable
possible, it makes end of loop
                {
                    possible++;
                }
}
/*
//If someone wants to see how the floodfill looks like on pathTable it is
enough to delete this comment section
printf("WORKS\n");
tabprint(ms, PathTable);
printf("\n\n\n");
*/
tabprint(ms, MazeTable);
getchar();

      return 0;
       }
```

6.Testing

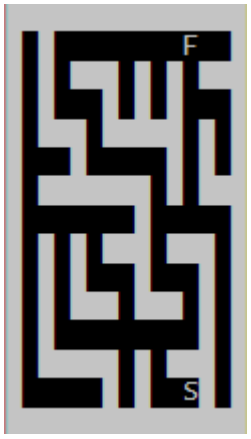
Recommended size is between 10 and 60~ it is dependent on console window. I will put here examplary test of creating maze with possible walk.
Screen how table with algorithm looks like.First in my test it

generated something like that, there is no way. Size 15, frequency 4.

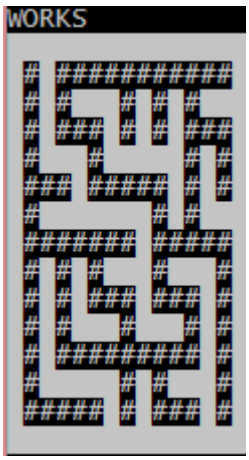
Output table on the screen



Because of loop (condition finish is not accessable)it must have created another one



Generating output maze, then using algorithm it checks if it's possible to get to finish



So as the result it was returning previous maze (without showed algorithm)

7.Conclusions

I have nothing more to say about already created code. The whole programm in my opinion is ready. The only one thing which can be changed is position of start and finish. They're not strictly hold with end of tunnel.