PROJECT REPORT

FOREST FIRE SIMULATION

MOUREY Quentin
NASS Michaël

23/05/2019

OUTLINE

INTRODUCTION

About a month ago, we were given a project to do for a duration of a month, and then present it as an oral presentation supplied with a report.

In this report, we will talk about what we had to do for this second project of the second semester.

Our second project is about to simulate the spreading of a forest fire. On a given field, the fire must spread all along it regarding the time and the distance. Multiple constraints are placed, and the most important one will be the type of material. Indeed, the field is composed of materials all different from each other. We have a total of eight types of material, and each type will be consumed by fire faster or more slowly according to the time (except for water and ground). The whole field is randomly generated, and then adjustable by the user to fit the size of the field.

The final purpose of this is to know what was consumed by the fire, what wasn't and in how many steps.

During the realisation of the code, many problems occurred, and we took some decision in order to make the code work properly as expected. During this report, we will present our working code of the simulation of a forest fire with the help of pictures and explanation.

Thus, we will firstly make the presentation of our code, the features expected and added, how does it work thanks to our functions and our main. Then, we will talk secondly about the code itself, how we managed to make it work, the difficulties encountered and finally how we could have improved our code.

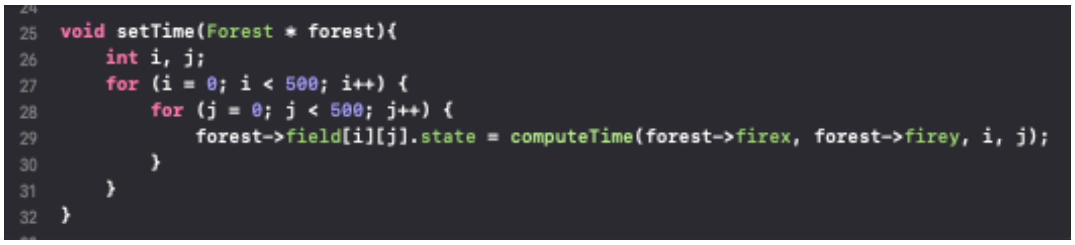
PRESENTATION OF THE CODE

a) How does it work

For a matter of visibility and clarity, we started to divide our code. Indeed, there are a couple of files that are each a sub-code of the main code which is our main.c. The main is hence very short and is used only to call our functions and run the program. Each function is in a separated file, which helps for a quick change on it, or for the reader to easily find where are they are located.

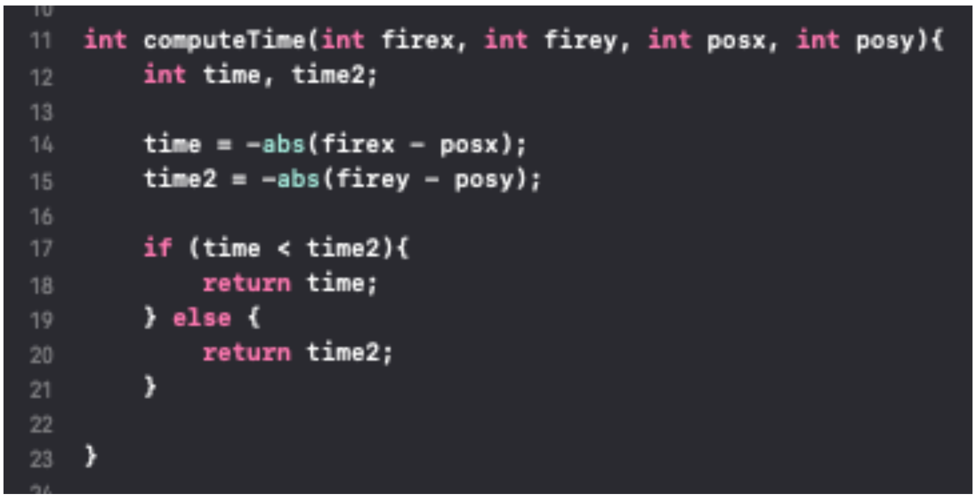
Talking about functions, let us introduce them. All our functions are clear and easy to find, due to the name given that is obvious. Each function has an important role in our whole program and are called when needed in the main.

As an example, we know that in the forest fire, the time is an important variable. For this, we created the file called “TimeManager.c” that, as its title say, manage the time during the game. This file contains five function, meant to set the time or compute it for example. We have below an example of the function called “setTime()” that sets the initial time of the forest:



```
void setTime(Forest * forest){
    int i, j;
    for (i = 0; i < 500; i++) {
        for (j = 0; j < 500; j++) {
            forest->field[i][j].state = computeTime(forest->firex, forest->firey, i, j);
        }
    }
}
```

We can see here that we make the use of structures, here called Forest. Instead of taking the time as an entity, we used it to be a single layer, linked with the distance. The function "setTime" calls the other function "computeTime" that is meant to compute the time at t_0, by calculating the negative absolute distance between the fire start coordinates and the Element of the Forest.

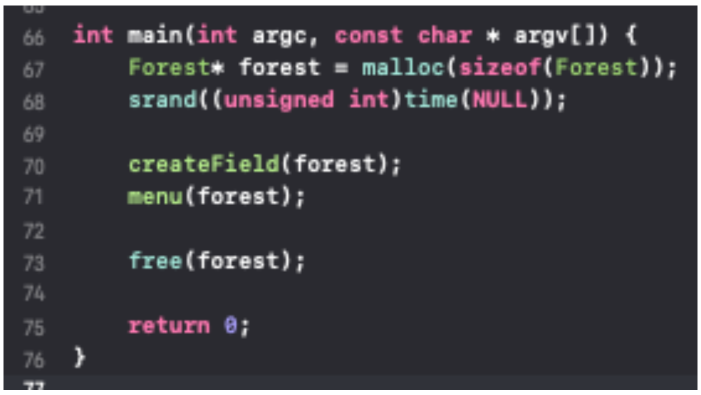


```c
int computeTime(int firex, int firey, int posx, int posy){
    int time, time2;

    time = -abs(firex - posx);
    time2 = -abs(firey - posy);

    if (time < time2){
        return time;
    } else {
        return time2;
    }

}
```

Here, we use a two dimensional "for" statement in order to check both sides of the forest which are the y-axis and the x-axis. Thus, for every element of the field of our forest, time is computed.

Now that we showed how one of our functions works and is called locally, let's see how it is called on the main algorithm that calls all the functions, and thus make the program work.

Here is our short and efficient main:



```c
int main(int argc, const char * argv[]) {
    Forest* forest = malloc(sizeof(Forest));
    srand((unsigned int)time(NULL));

    createField(forest);
    menu(forest);

    free(forest);

    return 0;
}
```

It dynamically allocates the Field, initiate the time for the random function and calls the function to create the Field. When it is done, it calls the function menu which is a recursive function and asks at every turn what the user want to do.

b) Features

We decided to use a large 2D array for the field of the forest. This allows to have a modular field size at any time easily. When generating the field, the user is asked to define the x and y size of the field. The user is also prompted to select the x;y coordinates of the fire start point. If the coordinates are matching water or ground, he is prompted to select new coordinates. The user is invited to customize the generated field. Our field has the advantage not to be dependent on time but on distance, which allows us to easily change an element, change the size of the field or even change the source of the fire at any time. We also give the user the possibility to navigate through time using a t+x function allowing to jump or reverse an indicated number of steps, but also using a t=x function allowing to know precisely at any time how the fire is propagating. The program has no user interface, however it does provide a visual experience using Emojis characters (🌊,🌴,🍂,🌿,🛤,🗿,🔥,🌫,⬛). Since we use emojis for fire, ashes and extinguished ashes, we also provide a functionality to get more information on any element of the field. You can know what the element actually is/was, in how many steps it will burn, transform into ashes and/or extinguished ashes.

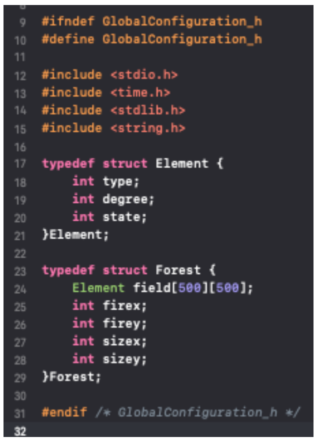
BEHIND THE CODE

a) *How the code was thought*

Firstly, we judged important the clarity of our code for the reader: we divided it into many files each of them containing all necessary functions in order to make an action, such as for example the file “TimeManager.c” which contains five functions that are related to manage the time in the game.

We began our code with an existing code already, written by us: our first project which was a Battleship. Indeed, there were a few things that are quite the same between this project and the Battleship. For example, there is a field to manage on both projects; so we have just re-used the main functions of the Battleship project and renamed them for the Forest Fire project. As well, we had the idea on the Battleship project to display the different ships on the field with emojis, and we did that again for this project as said earlier.

We started to create a Header folder, in which we have made our first basis header file, named “GlobalConfiguration.h”. This file hosts the definitions of the Element and Forest structures, and the general #include necessary for the program, and is referred in every other .h files.



```c
#ifndef GlobalConfiguration_h
#define GlobalConfiguration_h

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

typedef struct Element {
    int type;
    int degree;
    int state;
}Element;

typedef struct Forest {
    Element field[500][500];
    int firex;
    int firey;
    int sizex;
    int sizey;
}Forest;

#endif /* GlobalConfiguration_h */
```

The functions that we created for this project are mostly in the file “TimeManager.c” and “FieldManager.c”. The ground of the “TimeManager.c” file is to handle time without taking time as an actual value. So we decided to take benefits of the state value from the Element structure. Instead of just saying if the object is on fire or not, it now fully represents our time scale. The function setTime() is crucial for this, as it calculates the distance between the fire center position and each elements of the state to get a distance radius inside the field. Therefore, when the time is running, the state of every element of the field increases by 1. Using this method, we can easily navigate through time without caring about the status of an element, as the time in the state value will manage it alone, thanks to the “DisplayManager.c” file that knows how to handle our time scale when we need to display the forest.

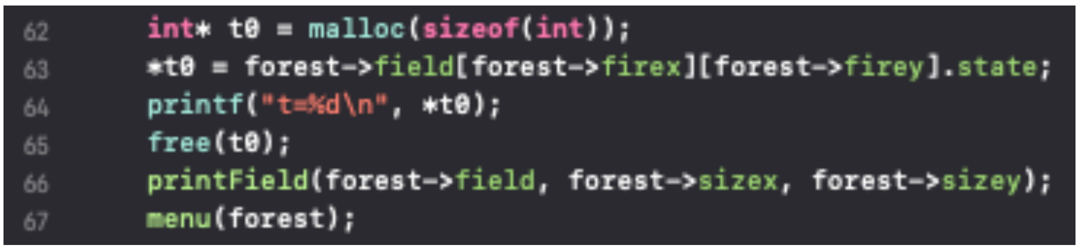
b) Difficulties encountered and possible improvements

During the writing of our code, we have of course faced many problems that we had to go through, as well as unsolved problems.

As we can obviously see it on the code, we have many functions on the program, which means that we had to manage them all firstly and find where they were every time we needed them. However, at the end, the clarity of the code was of course way better.

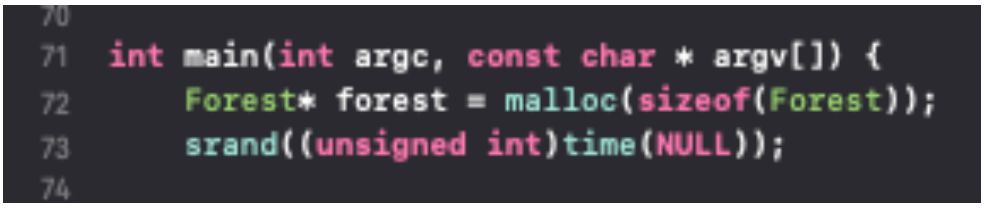
Indeed, at the end when the “main” was built, we had to make sure that it called the right functions in order to make the program work properly. This is why there is a recursive function above the main called “menu” that manages to call all the function written in the others files, and the “main” calls this function once, then the recursion makes it being called whenever needed and no need to call it again in the main.

Talking about recursion, this recursive function "menu" in the file "main.c". As we can see it on the picture below, the end of the function "menu" is this:



```
    int* t0 = malloc(sizeof(int));
    *t0 = forest->field[forest->firex][forest->firey].state;
    printf("t=%d\n", *t0);
    free(t0);
    printField(forest->field, forest->sizex, forest->sizey);
    menu(forest);
```

We can see on the last line of the picture which represents the function "menu", "menu" is called again in its own function. That is the recursion that we made but took us some time to think about on a first thought. We have also made the use of dynamic allocation with "malloc" for the size of the field of the forest, as we can see it on the first line of the same picture above, and at other locations such as in our main as shown below:



```
int main(int argc, const char * argv[]) {
    Forest* forest = malloc(sizeof(Forest));
    srand((unsigned int)time(NULL));
```

Here in the main, the malloc is still used for the size of the field of the forest.

A big issue that we had was when dealing with dereferencing the Forest when calling the printField() function from another function already taking Forest* as an argument. The compiler did not complain about any error, but it was making the program completely crazy, crashing on impossible lines (like a printf() or even a commented line) with an impossible to debug error message

(BAD_ACCESS). We tried serval solutions found on Internet to get around this, but nothing worked properly. We ended up rewriting the printField() function to not take the forest as an argument but the associated field and it's data values instead, and it solved our issue.

Finally, we had some issues with basic problems, sure basic, but very annoying still, which prevents the program from working properly, such as for example the confusion between the "greater than" and the "smaller than" signs when comparing two integers. Some problem occurred with this resulting the display of wrong results. As well, the very common problem of confusing the address and the value of a variable in our functions, which we solved in the end by always making the function calls by reference, to always use the same function prototype and avoid confusion, and for the function that do not edit anything, make the function calls by the values of the reference (the content of the structure).

Talking about the possible improvements of our code, we could have indeed made firstly a graphic interface for the Forest Fire. A graphic interface would have been more immersive for the simulation and for the user of course. Unfortunately, especially for this project, a graphic interface would have been way too difficult for us to make, and the duration of a month to finish the project was too short for this, as we are not used to use the SDL Library. Another improvement could be a backup functionality, which was not that complicated to create as we could have used the same code than the one in our Battleship project, which was reliable.

CONCLUSION

As a conclusion, the Forest Fire code taught us many new features that we gladly added to our program. We have of course improved and learnt many new things that we will not forget for our upcoming projects. Although we had already based a bit ourselves on our previous code of our previous project, the other part of the programming of the code wasn't easy to find and took us a certain time and re-thinking to write our code.