ENSIAS

Ecole Nationale Supérieure d'Informatique

et d'Analyse des Systèmes

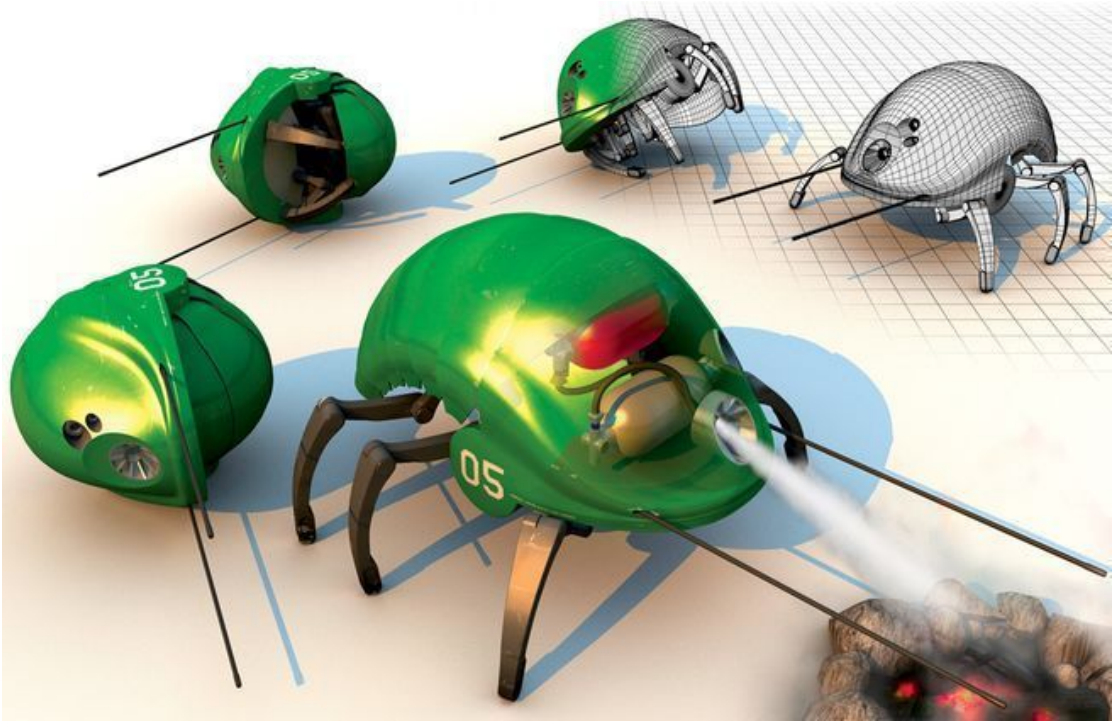
جامعة محمد الخامس بالرباط

Université Mohammed V de Rabat

Rapport de Projet C

Sujet :

Simulateur de robots



Réalisé par :

Mohammed SLIMANI

Jamal KHALIS

Encadré par :

Mr Mahmoud EL HAMLAOUI

Année Universitaire : 2017/2018

Dédicace

A nos chers parents pour leurs amours, leurs sacrifices et leurs encouragements ;

A nos familles qui nous ont toujours soutenu;

A nos amis de la promotion et chacun par son nom;

A tous nos amis;

A tous ceux et celles qui nous sont chers;

Nous dédions ce modeste travail.

Remerciements

Au terme de ce travail, nous exprimons nos plus vifs remerciements à notre encadrant **Mr. Mahmoud EL HAMLAOUI**, pour ses précieux conseils et ses encouragements au cours de la préparation de ce projet.

Aussi nous adressons nos remerciements à tous nos professeurs d'avoir partagé avec nous leur passion pour l'enseignement. Nous avons grandement apprécié votre soutien, votre implication et votre expérience.

Aussi nous tenons à remercier toutes les personnes qui ont contribué au succès de la réalisation de ce projet et qui nous ont aidé lors de la rédaction de ce rapport.

Enfin, nous remercions les membres du jury devant lesquels nous avons l'honneur de présenter ce travail, ainsi que nos familles, nos collègues de l'ENSIAS, nos amis et tous ceux qui nous ont supporté de près ou de loin.

Résumé

Dans le cadre de ce projet, on imagine un système logiciel complexe permettant d'aider les secours à combattre un ou plusieurs départs de feu sur une zone à risque délimitée et comportant :

un certains nombres d'OLE qui doit se déplacer vers l'endroit où se trouve le feu et vider leur réservoirs d'eau sur ce feu.

un drone qui fournit une carte de la zone de risque.

Abstract

When a fire occurs, It is really bad to let it stay for a while because it will grow. Firemen thought about how to reduce the time when the fire is up and they came up with OLE robots.

It is a system where there are many robots that know the place of the fire thanks to a drone that flies in the giving area.

In this project we are going to simulate the process of putting down fire with this system using the C language and the SDL library.

Liste des abréviations

Abréviations	La signification
SDL	**Simple DirectMedia Layer** est une bibliothèque logicielle libre
OLE	Off-road *Loescheinheit*

Table des figures

Table des matières

Introduction

L'organisation de secours lors d'une catastrophe majeure est un problème important. La gestion des différentes équipes travaillant sur un incendie est très complexe. On cherche donc actuellement à développer des moyens informatiques permettant de faciliter et de guider le travail des sauveteurs sur le terrain. Un domaine de recherche est l'utilisation de robots pompiers collaborant avec des drones fournissant une carte comportant des informations sur le terrain et les incendies.

C'est pour cela après avoir étudié plusieurs matières du langage C durant le premier semestre, et afin de permettre aux étudiants de mettre en pratique les différentes notions acquises durant les cours, les TDs et les TPs, nous étions invités à réaliser un logiciel de simulation de l'utilisation de robots-pompiers OLE.

Pour réaliser ce jeu nous avons utilisé plusieurs bibliothèques on en cite : bibliothèques standard, SDL et SDL_image.

Chapitre 1 : Analyse et Conception

1.1 La description

nous imaginons un système logiciel complexe permettant d'aider les secours à combattre un ou plusieurs départs de feu sur une zone à risque délimitée et comportant un robot et un drone.

1.2 Le principe

Les robots sont des << pompiers élémentaires >> qui peuvent se déplacer, éteindre un feu et qui connaissent leur position à tout instant. Ils sont également capables de calculer le plus court chemin dans une forêt, et à l'aide d'une drone qui va se balader dans la zone donnée et les informer des coordonnées des feux. Après les robots doit se déplacer vers l'endroit où se trouve le feu et vider son réservoir d'eau sur ce feu puis retourner à sa position.

1.3 Choix de la bibliothèque graphique

Les deux bibliothèques graphiques de C les plus populaires sont SDL et GTK, on a étudié les deux avant de décider laquelle choisir.

Pour faire simple, GTK est plus approprié pour faire des fenêtres avec des boutons et des menus etc..., de l'autre côté SDL est plus approprié pour créer des jeux, en plus de sa gratuité et qu'il est open source ça veut dire que ses bugs sont corrigés rapidement et qu'il est toujours à jour.

C'est pour cela qu'on a décidé de choisir la bibliothèque SDL.

1.4 Détails techniques

- Pour éviter la collision entre les robots, on a réduit le nombre de robots à un robot.
- On partitionné notre code en d'autres codes plus petits et abordables pour maîtriser et ne pas être perdu dans les lignes de codes.
- On sait mis d'accord sur les notations et sur le contenu de chaque fonction.

1.5 Notions utilisées

Nous avons implémenté ce travail en C et nous avons utilisé plusieurs notions, par exemples :

- Les structures.
- Les pointeurs.
- Les fonctions.

Chapitre 2 : La réalisation

2.1 Description des fonctions du code

Pour réaliser ce travail nous avions besoin de réaliser plusieurs fonctions, qu'on a réparti sur deux fichiers afin d'alléger le code et d'optimiser la compilation.

2.1.1 Les structures

➢

```
typedef enum bool {
        false,
        true,
}bool;
```

C'est une structure booléenne qui représente un type de variable à deux états. Les variables de ce types sont ainsi soit à l'état vrai soit à l'état faux (en anglais, **true** et **false**).

➢

```
typedef struct point {
        int x;
        int y;
}point;
```

C'est une structure qui représente les coordonnées d'un endroit dans la MAP.

➢

```
typedef  struct  queueNode{
        point  pt;    // Les coordonnées d'un point
        int  dist;   // La distance de ce point de la source
}queueNode;
```

C'est une structure qui représente les informations qui vont être enfiler lors de la recherche du plus court chemin.

➢

```
typedef enum smth {
        road,
        robot,
        obstacle,
        fire,
}smth;
```

C'est une structure représente ce qui existe dans une coordonnée d'un coordonne.

➢

```
typedef  struct  type_coord {
    smth something;
    point coord;
     int  special; //sera soit le degre de danger du feu soit le
                   //reservoire que contient un robot
                  //il sera -1 pour montrer que c'est un chemin
```

```
}type_coord;
```

Chaque endroit dans notre carte va porter les informations de ce structure.

➢

```
typedef  struct  Element {
queueNode n;
struct Element *suivant;
}Element;
```

C'est une structure que le nomme **Element** qui contient les informations de chaque nœud du file.

➢

```
typedef  struct  File {
Element *premier;
}File;
```

C'est une structure de la file que l'on va utiliser de trouver le plus court chemin.

2.1.2 Les fonctions

Avant d'expliquer le rôle de ces fonctions on a déclaré un tableau de deux démentions **type_coord MAP[max_point][max_point];** qui représente une carte que l'on va travailler, avec max_point est une constante égale à 20.

✓ **void Load_MAP( )**

Cette fonctions nous permet de remplir les champs de la carte d'une manière manuelle, à savoir les obstacles, les faux, les robots, les chemins, ...etc.

✓ **void afficher( )**

Cette fonctions nous permet d'afficher ce qui existe dans une coordonnée d'une carte.

✓ **void BFS(type_coord MAP[max_point][max_point], queueNode T[max_point][max_point], point src, point dest)**

C'est une fonction qui permet de remplir un tableau **T** et calculer la distance entre source et destination et prend en argument :

- **MAP[max_point][max_point]** notre carte
- **T[max_point][max_point]** le tableau qu'on va initialiser par des coordonnée visiter
- **Src** la source de robot
- **dest** la destinataire que se trouve le feu

Au sein de cette fonction on avait besoin d'autres fonctions :

- **File *initialiser( )** qui permet de déclarer et allouer une espace mémoire d'une file et l'initialiser par une valeur NULL.
- **void enfiler(File *file, queueNode s)** qui permet d'ajouter un élément au début de la file.
- **bool empty(File *F)** qui retourner **true** si la file est vide et **false** sinon.
- **queueNode front(File *F)** qui permet de retourner le premier élément de la file.
- **void defiler(File *file)** qui permet de supprimer au début de la file.
- **bool isValid(int row, int col)** qui permet de ne pas dépasser la carte.

✓ **point* GetPath(queueNode T[max_point][max_point], point dest)**

C'est une fonction qui prend en argument un tableau **T** et le point où se trouve le feu **dest**, et retourner un pointeur sur un tableau de points, ces points construit le plus court chemin entre le robot et le fau.

✓ **void move_robot(int distance, point* path, SDL_Surface* screen)**

Cette fonction qui prend en argument le plus court distance et le plus court chemin entre le robot et le feu ainsi que un pointeur **screen** sur **SDL_Surface**. Elle permet de déplacer le robot sur le chemin donnée en paramètre vers le feu.

Au sein de cette fonction on avait besoin d'un autre fonction **void make_road(point A)** qui permet de remplacer une coordonnée de la carte par la rote.

✓ **void ReturnToBase(int distance, point * path, SDL_Surface * screen)**

Cette fonction est la même que la fonction précédente mais elle permet de retourner le robot vers la source.

✓ **point drone( )**

Cette fonction a le même rôle d'un drone qui se balader sur la carte et informe le robot des coordonnée des feux (elle retourne un **point**).

✓ **void LoadImagesToThe(SDL_Surface *screen)**

Cette fonction va représenter la matrice (la MAP) et donc au lieu de voire des nombres an va voire des images appropriées à chaque type de coordonnées :

0 -> route

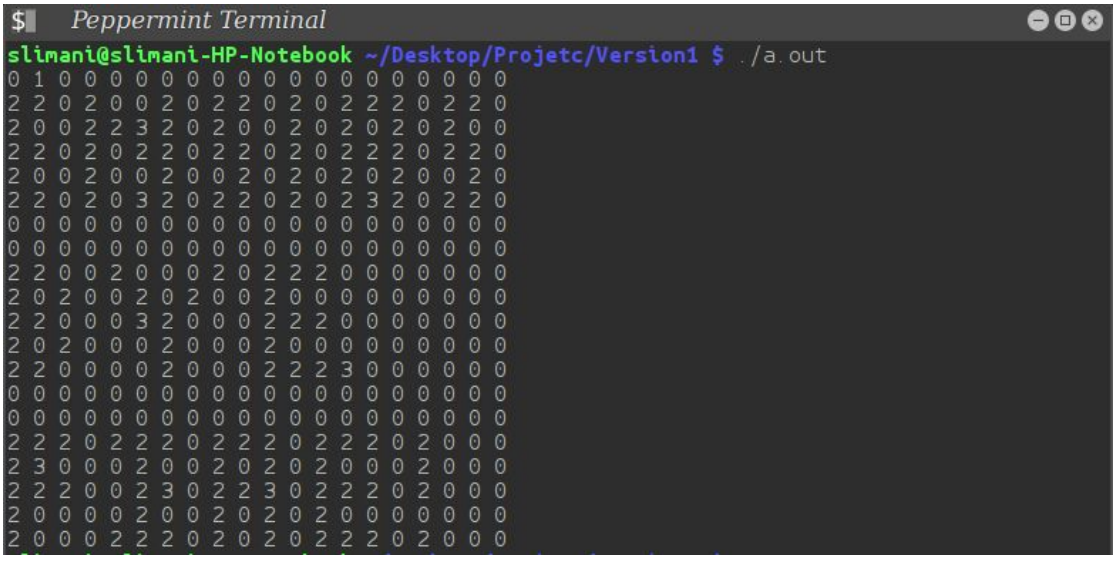
1 -> robot

2 -> obstacle

3 -> feu

le fait d'appeler cette fonction plusieurs fois est celui qui va montrer les mouvements de notre robot.

2.2 La console



0 - représente la route

1 - représente le robot

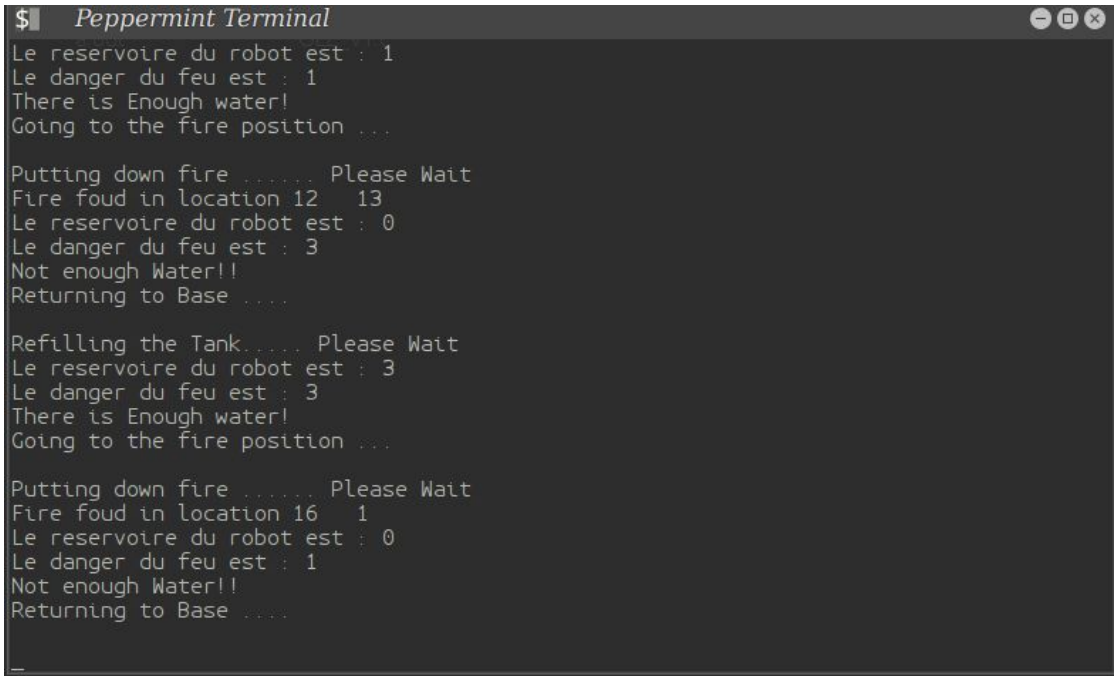
2 - représente les obstacles

3 - représente le feu

Le robot au cours du temps va se déplacer vers les lieux des feux tout en respectant les obstacles et en choisissant le plus court chemin a chaque fois

2.3 L’interface graphique

Le robot affirme ce qu’il est entrain de faire dans le console.



donc on peut voire la logique avec laquelle notre robot pense : Tant qu'il y a de l'eau dans le reservoire qui peut éteint le feu le robot va au lieu du feu pour l'éteindre.

conclusion

A la fin de ce rapport, on avoue que la réalisation de ce travail était une expérience assez enrichissante, il nous a permis de bénéficier de nouvelles connaissances venues compléter celles que nous avons acquis tout au long de notre formation.

Les principales difficultés rencontrées résident dans l'apprentissage et l'utilisation de la bibliothèque de SDL qui n'a pas une très bonne documentation.

Bibliographie

- https://www.popsci.com/scitech/article/2008-03/firefighting-robot#page-3