ENSIAS

Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes

RAPPORT DE PROJET :

Simulateur de robot

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

Réalisé par :

Soufiane El Habti

Oussama Khalfaoui

Encadré par :

Pr. Mahmoud El Hamlaoui

Pr. Abdellatif EL FAKER

Remerciments

Nous souhaitons remercier notre encadrant Pr. **Mahmoud El Hamlaoui** pour nous avoir donné la chance de réaliser ce projet et pour ses précieux conseils.

Aussi, nous remercions toutes les personnes qui ont contribué à la réalisation de ce projet et principalement les professeurs qui nous ont formé en tout ce qui concerne le language C.

Enfin, nous remercions les membres du jury devant lesquels nous avons l'honneur de présenter ce travail.

Sommaire :

Introduction :

L’organisation des opérations de secours lors d’un incendie représente un grand danger pour les pompiers. Pour remédier à ce probléme, le robot pompier OLE « Off-road Loescheinheit » a été établit. Il s’agit d’un robot résistant au feu équipé d’un réservoir d’eau, et tout équipement nécessaire pour éteindre un feu, dont des capteurs de chaleur et des détecteurs d’obstacles pour assurer son autonomie. Il est guidé par un GPS qui lui donne le plus court chemin de sa position actuelle au feu.

Dans ce projet, notre objectif est ainsi de simuler le comportement des robots pompiers OLE dans leurs parcours vers les départements de feu avec le language C.

1ère Partie : Conception

- **1.1 Description :**

Nous voulons un programme logiciel qui va nous permettre de combattre un ou plusieurs départements de feu dans une zone de risque délimitée qui comporte des obstacles .

- **1.2 Principe :**

Le drone qui flotte notre zone donne à notre robot les coordonnées du feu. Le robot quant à lui peut se déplacer de façon élémentaire, éteindre un feu et peut établir le plus court chemin de sa position actuelle vers le feu que lui communique le drone. Le robot se déplace ainsi vers le feu, vide son réservoire sur lui et retourne vers la station pour se charger. Le pathfinding peut etre accompli à l'aide de l'algorithme BFS ou de l'algorithme du Q-learning.

- **1.3 Bibliothèques choisies :**

Pour réaliser ce travail, en outre que les bibliothèques standarts classiques, on a eu besoin d'une bibliothèque graphique. On a choisi la bibliothèque SDL "Simple DirectMedia Layer" qu'on a trouvé plus approprié pour ce travail que la populaire bibliothèque "GTK" .

- **1.4 Petits details :**

+ On a réduit le nombre de robot à un seul, et ce pour éviter les collisions.

2ème Partie : Execution

2.1 Partie avec l'algorithme BFS.

2.1.1 Les structures utilisés :

```
typedef struct point {
    int x ;
    int y ;
}point;    // Structure qui shématise un point de notre carte/zone.
```

```
typedef enum bo{
    false,
    true
}boolean;    // True and false à la place de 1 et 0.
```

```
typedef enum whatsin{
    road,
    obstacle,
    ole,
    fire
}whatsin;  // Structure qui nous donne une information sur ce qu'il y'a dans chaque pt de notre carte.
```

```c
typedef struct caracs{
    point p ;
    whatsin w ;
    int deg ; // degré du feu.

}caracs; //Structure qui donne les caractérisque d'un point : Ses coordonnées, ce qu'il y'a dans ce point et le degré du feu.
```

```c
typedef struct node{
    point p ;
    int dist ; //distance du pt de la source

}node; /*Chaque noeud contient ses coordonées et la distance de la source. Ce qui nous donne une information sur ce qui va
        etre enfiler lors de la recherche du plus court chemin */
```

```c
typedef struct chain{
    node n ;
    struct chain* next ;

}chain; // Structure d'une liste chainée qui va nous aider pour le plus court chemin.
```

```c
typedef struct {
    chain * first ;
}Queue;
```

2.1.2 Les fonctions utilisées :

On déclare tout d'abord notre carte/jardin qui est une variable globale :

```
caracs MAP[MAX][MAX] ; //MAX est une constante. Dans notre cas, elle est égale à 20.
```

On initialise notre jardin :

```
void Load_MAP(); //Fonction qui initialise notre Carte/Zone de risque manuellement.

void loadTheGuarden(SDL_Surface * jardin ,point fire_location ); // Fonction qui initialise graphiquement notre jardin.
```

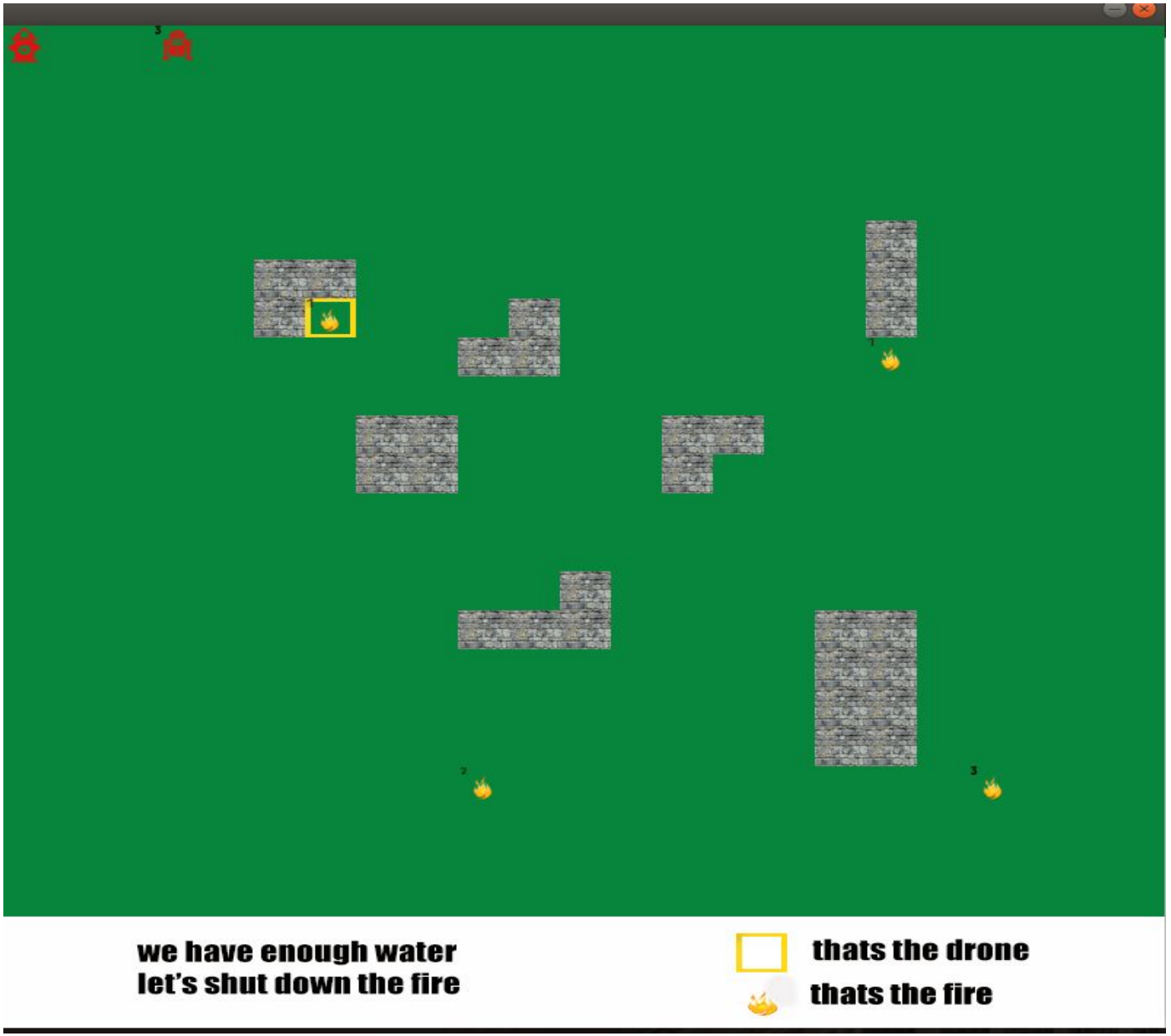
Les fonctions qui vont servir à determiner le plus court chemin :

```
boolean isinrange(int row, int col); // Fonction qui vérifie si un point est effectivement dans notre carte/jardin.

Queue *initialise(); //  Fontion qui initialise notre liste.

void enqueuenode(Queue * queue, node no); // Fonction qui permet d'enfiler un noeud à notre liste.

boolean vide(Queue *queue); // Fonction qui vérifie si notre liste est vide.

void dequeuenode(Queue *queue); //Fonction qui défile le 1er noeud de la liste.

node front(Queue * queue); //Fonction qui retourne le 1er noeud de la liste.

point  drone(); /* Fonction qui représente notre drone. Elle retourne un point ou il y'a un feu.
C'est la destination du robot auquel on determine le plus court chemin */

void BFS(caracs MAP[MAX][MAX], node T[MAX][MAX], point src, point dest) ; /* En utilisant les fonctions précedentes, cette
fonction remplit le tableau T par l'algorithme BFS. T[i][j] est un noeud dont les coordonnées sont {i,j} et la distance est
la distance correspondante entre ce point et la source (src) */

point* GetPath(node T[MAX][MAX], point dest); //Fonction qui retourne un pointeur sur un tableau de points du plus court chemin.
```

Les fonctions qui ont avoir avec le mouvement de notre robot :

```
void send_robot(int dist, point fire_location , SDL_Surface * jardin , point * path);
/*  Fonction qui déplace notre OLE dans le parcours qui correspond au tableau vers lequel pointe path.
C'est le plus court chemin */

void make_it_way(point p); // Fonction qui transforme un point de la carte en chemin/road.
                           //Ce qui est utile après mouvement de notre robot.
```

2.1.3 Interface graphique :



2.2 Partie de l'integration de l'algorithme du Q-learning :

2.2.1 Les structures :

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

- typedef enum whatsin{
  road,
  obstacle,
  robot,
  fire
  }whatsin;

- typedef struct caracs{
  point p ;
  whatsin whatsin ;
  int deg ;

  }caracs;

En addition , on a placé des constantes :

RIGHT: pour l'action de déplacement vers droit .

LEFT :pour l'action de déplacement vers gauche.

UP: pour l'action de déplacement vers l'haut .

DOWN : pour l'action de déplacement vers le bas .

Et une **variable globale** de type matrice d'entiers Q qui représente la matrice du Q-learning ou on va mettre les valeur de l'équation de bellman.

Et Q a pour indice de lignes les "states" c'est à dire les "whatsin" et pour colonnes les actions possible dans une position donnée.

2.2.2 Les fonctions utilisées :

- void loadTheGarden(SDL_Surface * jardin ,point robot_location); Définie dans la dernière partie.
- int possible_move(point location);

Une fonction qui retourne 0 si les coordonnées du point donnée en input dépasse le max des point ou négatives.

- void move_down(point * location); Pour déplacement vers bas.
- void move_up(point * location); Pour déplacement vers l'haut.
- void move_right(point * location); Pour déplacement vers droit.

- void move_left(point * location); Pour déplacement vers gauche.
- void move(point robot_location , caracs MAP[MAX][MAX], SDL_Surface * jardin); Qui deplace le robot en se basant sur la matrice de Q-learning.
- void Q_matrix(point position); Fonction qui remplie la matrice de Q-learning par l'équation de bellman. Et pour 400 cycles d'entraînement.
- float max_Qstate(int state); Pour state donnée on prend l’action qui a la valeur de Q la plus grande.
- void setQ2zeros(); Mettre les valeurs de matrice Q à zero.
- int R_value(int state , int action , point position); Retourne soit -100 si le robot pass par un obstacle. 100 si le robot trouve un feu. 0 pour road .
- void Load_MAP(); Définie dans la dernière partie.

2.2.3 Interface graphique : C’est la même utilisée dans la dernière partie.

2.2.4 Problèmes techniques : malheureusement, quand on lance l'exécution de notre programme, on constate qu’il ne marche pas comme prévu . par exemple le robot passe par des obstacles dans ses cycles d'entraînement .

Conclusion

En conclusion, on note que ce projet a été une experience bien enrichissente qui nous a permis d’aquérir de nouvelles compétences et de renforcer/compléter les conaissances qu’on avait vu en cours.

4. Bibliographie.

- Cours de structure de données du Pr. El Faker.
- Cours de techniques de programmation du Pr. Guermah.
- https://wiki.libsdl.org/
- http://www.popsci.com/scitech/article/2008-03/firefighting-robot
- https://cloud.irit.fr/index.php/s/Tcu7us44FM5fLLJ
- https://towardsdatascience.com/introduction-to-q-learning-88d1c4f2b49c