Robots programmables

COTREZ Léo ORNIACKI Thomas

Université Paris-VIII, Saint-Denis, France

Licence 2 Informatique

Premier Semestre 2018

Table des matières

1 Procédures mise en œuvre

1.1 Création de l'air de jeu

Afin de générer une aire de jeu personnalisable nous avons decidé de stocker les airs de jeu dans des fichiers, notre jeu devait donc être en mesure de récupérer ces données et les utiliser. Pour ce faire nous avons commencé par coder une fonction qui récupére chaque caractère du fichier et les stocke dans une structure map qui contient un tableau de caractère pour la map, associé à la largeur et la hauteur, un tableau de zone de réaparition.

1.2 Gestion client/serveur

Notre compréhension de l'enoncer nous a ammené a faire un jeu en temps reel. La solution qui a été retenue est de séparer le jeu en plusieurs executables, un executable serveur et des clients qui communiquent entre eux grace à des files de messages. Les clients demandent des actions au serveur qui leur répond en les syncronisants entre eux. Nous avons choisi d'utiliser *mqueue.h*, une librairie de gestion de file de messages qui nous permet d'écrire des chaîne de caractères dans un flux. Cette solution nous a mené à de nouvelles problematiques.
Les messages sont de tailles variables et il faut que le serveur et le client ai connaissance de la taille de l'information à récuperer et comment l'interpréter. De plus si on envoi plusieurs messages sur une file de messages lu par plusieurs processus, il y a un risque que les messages se mélange avec une partie de message par client.
Pour resoudre ces problemes, nous avons créé une structure *msg*, qui contient l'identifiant du robot, qui envoie l'action, et l'identifiant de l'action choisi. Chaque action est differencié des autres grâce à un identifiant qui indique au processus la taille du message à recuperer. Puis pour eviter que les messages soit capté par le mauvais processus, ils sont concaténés avant d'être envoyés.

1.3 Commande et script

Le dernier problème auquel nous avons été confronté fût la gestion des commandes et du script. L'objectif était d'integré un langage minimaliste pour permettre au joueur de controler les robots aussi bien en tapant des commandes une par une qu'en lire plusieurs dans un fichier. nous avons donc penser a une structure commande qui contiendrais le nom de la commande, le nombre d'argument, le nombre de sous commande et un tableau de sous commande. Chaque élément du script est une commande qui est situé dans le tableau de sous commande d'une autre commande et qui peut elle même

contenir des sous commandes.

2 Mode d'emploi

2.1 Language minimaliste

Le langage minimaliste que nous avons developpé utilise une notation préfixé sans parenthèses, l'arité des opérateurs est fixé de la manière détaillée ci-dessous.

Commande	Arité	Description	Exemple
quit	0	Abandonner la partie	quit
pv	0	Donne les points de vie	pv
steer	0	Donne la direction	steer
money	0	Donne le solde	money
nb_bullet	0	Donne le nombre de balle	nb_bullet
armor	0	Donne les points d'armure	armor
pick	0	Ramasse l'objet à porté le plus proche	pick
script	1	Execute le script *nom de fichier*	script bot_1
move	1	Déplace le robot de n	move 42
turn	1	Tourne la direction de n	turn -2
shoot	1	Tir d'un angle n	shoot 90
coord	1	Donne la coordonnée sur l'axe x ou y	coord x
seek	2	Donne la coordonnée x ou y d'un objet (B,L,A,C,R)	seek R x
aim	2	Donne l'angle de tir pour des coordonnées	aim 15 6
!=	2	Opérateur d'inégalité	!= pos x steer
==	2	Opérateur d'égalité	== pv armor
>	2	Opérateur de supériorité	> 9 2
<	2	Opérateur d'infériorité	< 8 5
>=	2	Opérateur de supériorité et égalité	>= armor pv
<=	2	Opérateur de infériorité et égalité	<= 5 9
+	2	Opérateur d'addition	+ - 5 2 pv
-	2	Opérateur de soustraction	- pv armor
*	2	Opérateur de multiplication	* pv coord x
/	2	Opérateur de division	/ 28 6
mod	2	Modulo	mod 8 6
=	2	Affectation n à la valeur m	= hello 66

Syntaxe du *while* et du *if* dans un script :

```
while == nb_bullet 5 {
       move 12
```

```
        shoot 42
}
```

```
if < coord x seek R x {
        = lavie pv
        + armor lavie
}
```

2.2 Lancer le jeu

Afin de lancer le jeu il est nécessaire d'ouvrir trois consoles en executant les commandes comme détaillées ci-dessous.
Il suffit de deux clients pour lancer une partie mais plus en est de fou plus on rit.

Console - Serveur

```
$ make
$ ./server map_2
```

Console - Client 2

```
$ ./client Piscsou
```

Console - Client 3

```
$ ./client Harpagon
```

2.3 Jouer

À la fin du décompte ça y est l'aire jeu s'affiche et on peut commencer à jouer
Deux s'offre alors à nous télécommander le robot en inscrivant les commandes, vues précédement, unes par unes, ou bien programmer son robot à l'aide de scripts préalablement réalisés. Voici des exemples de scripts on remarquera qu'un script peut en appeler un autre tant que ce dernier ne fait de même sur le précédent.

```
while != nb_bullet 0 {
  if == move 2 -1 {
    turn 1
  }
  pick
  if != seek R x coord x {
```

```
    if != seek R y coord y {
      shoot aim seek R x seek R y
      script bot_2
    }
  }
}
```

```
shoot 0
shoot 45
shoot 90
shoot + 90 45
shoot 180
shoot + 180 45
shoot + 180 90
shoot + + 180 90 45
```

Votre robot est représenté par les symboles (ˆ,>, V,<), les objets collectibles, les coffres C, les balles B, les points de vie L, et les armors A ces dernières bien que collectibles ne sont pas effectives. Les murs et les tirs sont respectivement représentés par W et $*$.
Un robot gagne la partie en détruisant le robot adverse.

3 Listing du programme

3.1 Les fonctions de *fct_mini.c*

```
float get_coord(robot *bot, char *axis)
```

La fonction *get_coord*, prend en argument un pointeur sur une structure *robot* et une chaîne de caractères *axis* si cette dernière est x ou y, alors la fonction retournera alors respectivement la valeur de la position du robot sur l'axe des abscisses ou celui des ordonnées.

```
short get_direction(robot *bot)
```

La fonction *get_direction*, prend en argument un pointeur sur une structure *robot* et retourne la valeur de la direction du robot 0, 1, 2, ou 3 respectivement pour haut, droite, bas, gauche.

```
short get_pv(robot *bot)
```

La fonction *get_pv*, prend en argument un pointeur sur une structure *robot* et retourne le nombre de point de vie du robot.

```
unsigned long long get_money(robot *bot)
```

La fonction *get_money*, prend en argument un pointeur sur une structure *robot* et retourne le solde du robot (*unsigned long long*, il aime vraiment beaucoup l'argent ce robot).

```
short get_nb_bullet(robot *bot)
```

La fonction *get_nb_bullet*, prend en argument un pointeur sur une structure *robot* et retourne le nombre de balle du robot.

```
short get_armor(robot *bot)
```

La fonction *get_armor*, prend en argument un pointeur sur une structure *robot* et retourne le nombre de point d'armure du robot.

Les fonctions suivantes peuvent prendre en argument des fils de messages *server*, *client*, une chaîne de caractères *buffer* et un entier *taille* afin de communiquer avec le serveur.

```
int avancer(robot *bot, int move, mqd_t server, mqd_t client,
    ↪ char* buffer, int taille)
```

La fonction *avancer*, prend en argument un pointeur sur une structure *robot*, un entier *move* et demande au plus possible le déplacement du robot de *move* dans sa direction actuelle au serveur.

```
int aim(robot *bot, int x, int y)
```

La fonction *aim*, prend en argument un pointeur sur une structure *robot*, deux entiers x, y et retoune angle avec lequel le robot doit tirer afin d'atteindre le point formé par ces coordonnées, depuis sa position actuelle.

```
int seek(robot *bot, char *obj, char *axis, mqd_t server,
    ↪ mqd_t client, char* buffer, int taille)
```

La fonction *seek*, prend en argument un pointeur sur une structure *robot*, deux chaines de caractères *obj*, *axis* et retoune si possible la coordonnée, x ou y selon l'*axis*, de l'objet *obj* le plus proche dans son champ visuel.

```
int ramasser(robot *bot, mqd_t server, mqd_t client, char*
    ↪ buffer, int taille)
```

La fonction *ramasser*, demande au serveur si le *bot* pourrait rammasser un collectible tel qu'un coffre , de l'armure ou bien des balles.

```
int tourner(robot *bot, short direc, mqd_t server)
```

La fonction *tourner*, demande au serveur de changer la direction du *bot* de *direc* fois dans le sens des aiguilles d'une montre dans le référentiel haut, droite, bas, gauche.

```
int tirer(robot *bot, float angle, mqd_t server)
```

La fonction *tirer*, demande au serveur de un tir d'angle *angle* depuis la position courrante du *bot*.

Les deux fonctions suivantes prennent en arguments la structure *cmd sub_com* représentant la commande saisie par l'utilisateur manuellement ou par un script ainsi qu'un pointeur sur un pointeur de structure *aff*, *dico* qui va permettre de stocker les variables définit par l'utilisateur.

```
int eval(cmd sub_com, robot *bot, mqd_t server, mqd_t client,
    ↪ char* buffer, int taille, aff **dico)
```

La fonction *eval*, va se charger de retourner la valeur de retour de la fonction associée à la commande *sub_com*.

```
int interp(cmd sub_com, robot *bot, mqd_t server, mqd_t client
    ↪ , char* buffer, int taille, aff **dico)
```

La fonction *interp*, va se charger d'interpréter correctement la structure commande *sub_com*.

3.2 Les fonctions de *interpreteur.c*

```
char* get_line(FILE *fd)
```

La fonction *get_line*, prend en argument *fd*, un pointeur sur un descripteur de fichier et retourne une chaîne de caractères correspondant au contenu du fichier se trouvant entre le descripteur et le reste de la ligne.

```
cmd create_cmd(char **ligne, FILE *fd)
```

La fonction *create_cmd*, prend en argument *ligne*, un pointeur sur chaîne de caractères et *fd*, un pointeur sur un descripteur de fichier et retourne le contenu du fichier sur lequel pointe *fd* en une structure *cmd*.

3.3 Les fonctions de *server.c*

```
int server(char* map\_name)
```

La fonction *server*, prend en argument *map_name*, une chaine de caractère qui represente le nom de l'aire de jeu à charger. Cette fonction éxécute dans l'ordre les differentes fonctions pouvant être appelé par le serveur.

```
coord bot\_interact(map mapOfGame, robot\_liste listOfBot,
    ↪ robot* bot, char* buffer)
```

La fonction *bot_interact*, prend en argument *mapOfGame*, la structure qui contient les données de l'aire de jeu, *listOfBot*, la liste des robots present dans le jeu, *bot*, un pointeur sur le robot qui demande une interaction et *buffer* la chaîne de caractères qui contient la demande du robot. Cette fonction renvois les coordonnées de l'objet, demandé par le client, au serveur.

```
void start(mqd\_t* list\_mqueue)
```

La fonction *start*, prend en argument *list_mqueue*, la liste des file de message des clients connecté au serveur. Cette fonction envoie a tout les clients un message de debut de partie toutes les secondes pendant 5 secondes.

```
void move_bullet(bullet_liste* list_bullet, robot_liste*
    ↪ bot_list, map mapOfGame, mqd_t* list_mqueue)
```

La fonction *move_bullet*, prend en argument *list_bullet*, la liste des balles presente sur l'aire de jeu, *bot_list*, un pointeur sur la liste des robots present dans le jeu, *mapOfGame*, la structure qui contient les données de l'aire de jeu et *list_mqueue* la liste des file de message des clients connecté au serveur. Cette fonctions permet d'actualiser les balles de l'aire de jeu et d'informer les clients des dégâts reçus.

```
coord bot_interact(map mapOfGame, robot_liste listOfBot, robot
    ↪ * bot, char* buffer)
```

La fonction *bot_interact*, prend en argument *mapOfGame*, la structure qui contient les données de l'aire de jeu, *listOfBot*, la liste des robots present dans le jeu, *bot*, un pointeur sur le robot qui demande une interaction et *buffer* la chaîne de caractères qui contient la demande du robot. Cette fonction retourne la position de l'objet que le client a demandé.

```
void affichage(map mapOfGame, robot_liste listOfBot,
    ↪ bullet_liste listOfBullet)
```

La fonction *affichage*, prend en argument *mapOfGame*, la structure qui contient les données de l'aire de jeu, *listOfBot*, la liste des robots present dans le jeu, *listOfBullet*, la liste des balles presente dans le jeu. Cette fonction affiche l'etat du jeu.

```
robot* isBot(int x, int y, robot_liste listOfBot)
```

La fonction *isBot*, prend en argument x, l'abscisse du point testé, y, l'ordonné du point testé, *listOfBot*, la liste des robots present dans le jeu. Cette fonction teste si il y a un robot au coordonnées indiqué et retourne un pointeur sur le robot et NULL si il n'y en a pas.

```
int isBullet(int x, int y, bullet_liste listOfBullet)
```

La fonction *isBullet*, prend en argument x, l'abscisse du point testé, y, l'ordonné du point testé, *listOfBullet*, la liste des balles presentes dans le jeu. Cette fonction teste si il y a une balle au coordonnées indiqué.

```
int in_range(coord pos,robot_liste listOfBot)
```

La fonction *in_range*, prend en argument *pos*, les coordonnées du point testé, *listOfBot*, la liste des robots present dans le jeu. Cette fonction teste si le point de coordonnées *pos* est à portée du robot

```
int win(robot_liste listOfBot)
```

La fonction *win*, prend en argument *listOfBot*, la liste des robots present dans le jeu. Cette fonction teste si un joueur a gagné, retourne l'identifiant du gagnant, -1 sinon.

```
int search_place(char* place,int nb_place)
```

La fonction *search_place*, prend en argument *place*, un tableau qui represente les places prise et les places restante dans le jeu, et *nb_place* le nombre de place totale. Cette fonction retourne l'index d'une case libre, -1 sinon.

```
int create_map(char* path_file, map* new_map)
```

La fonction *create_map*, prend en argument *path_file*, une chaîne de caractère qui represente le nom du fichier à lire, et *new_map* un pointeur sur l'aire de jeu du jeu. Cette fonction remplie l'aire de jeu avec les informations du fichier.

3.4 Les fonctions de *client.c*

```
int reception(mqd_t fdem, char** buffer, int taille, robot*
    ↪ bot, char obj)
```

La fonction *reception*, prend en argument *fdem*, un descripteur sur une file de message en lecture, *buffer*, un pointeur sur une chaine de caractère, *taille*, la taille des message lu, *bot*, un pointeur sur un robot et *obj*, la valeur recherché par la fonction qui appelle *reception*. Cette fonction va traduire les messages envoyer par le serveur et retourner l'information demandé par client.

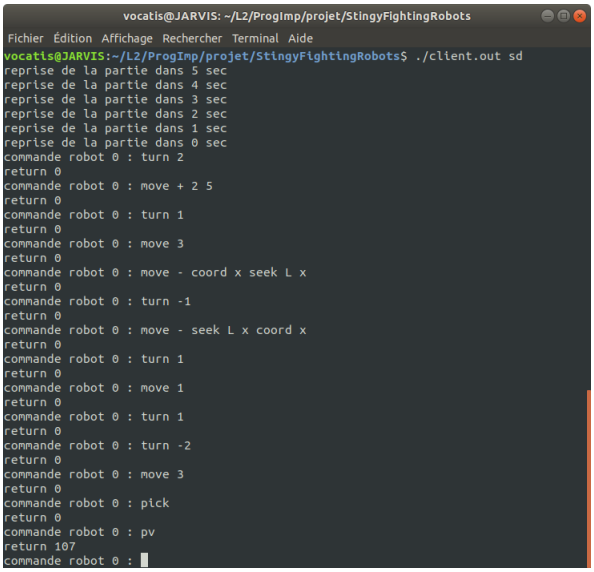
```
int client(char* name)
```

La fonction *client*, prend en argument *name*, une chaîne de caractères qui represente le nom du robot associé à ce client. Cette fonction éxécute dans l'ordre les differentes fonctions pouvant être appelées par le client.

4 Traces d'utilisation

4.1 Captures d'écran

Terminal ayant saisit plusieurs commandes à son robots



Terminal affichant l'aire de jeu lors d'un échange de tirs entre deux robots

A Code complet

A.1 Les déclarations

```
#ifndef GAME_H_
#define GAME_H_

#include <unistd.h>
#include <mqueue.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <math.h>

/*#define _POSIX_C_SOURCE 199309L */
#define CYCLE 10000
#define RAD (3.14159265/180)

typedef struct message msg;
typedef struct coordonnees coord;
typedef struct map map;
typedef struct robot robot;
typedef struct elem_robot elem_robot;
typedef struct elem_robot* robot_liste;
typedef struct inventaire inventaire;
typedef struct bullet bullet;
typedef struct elem_bullet elem_bullet;
typedef struct elem_bullet* bullet_liste;
typedef struct chest chest;
typedef struct commande cmd;
typedef struct affect aff;

/* fonctions de server.c */
int create_map(char* path_file, map* new_map);
robot* isBot(int x, int y, robot_liste listOfBot);
int isBullet(int x, int y, bullet_liste listOfBullet);
int server(char* map_name);
int win(robot_liste bot_list);
coord bot_interact(map mapOfGame, robot_liste listOfBot, robot
    ↪ * bot, char* buffer);
void affichage(map mapOfGame, robot_liste listOfBot,
    ↪ bullet_liste listOfBullet);
```

```
void move_bullet(bullet_liste* list_bullet, robot_liste*
    ↪ bot_list, map mapOfGame, mqd_t* mq_list);
int search_place(char* place,int nb_place);
int in_range(coord pos,robot_liste listOfBot);
void start(mqd_t* mq_list);

/* debug.c */
void test(robot_liste listOfBot);
void test2(bullet_liste test);
void test3(aff* dico);
void glup(cmd com, int nb_tab);

/* fonctions de client.c */
int client(char* name);
int reception(mqd_t fdem, char** buffer, int taille, robot*
    ↪ bot, char obj);

/* fonctions fct_mini.c */
float get_coord(robot *bot, char *axis);
short get_direction(robot *bot);
short get_pv(robot *bot);
unsigned long long get_money(robot *bot);
short get_nb_bullet(robot *bot);
short get_armor(robot *bot);
int seek(robot *bot, char *obj, char *axis, mqd_t server,
    ↪ mqd_t client, char* buffer, int taille);
int ramasser(robot *bot, mqd_t server, mqd_t client, char*
    ↪ buffer, int taille);
int avancer(robot *bot, int move, mqd_t server, mqd_t client,
    ↪ char* buffer, int taille);
int tourner(robot *bot, short direc, mqd_t server);
int tirer(robot *bot, float angle, mqd_t serveur);
int interp(cmd sub_com, robot *bot, mqd_t server, mqd_t client
    ↪ , char* buffer, int taille, aff **dico);

/* fonctions de game.c */
robot create_robot(char* name, char id, coord spawn,
    ↪ inventaire* inventaire);
bullet create_bullet(robot *bot, float speed_x, float speed_y)
    ↪ ;
void str_concat(char* str, char* elem1, int t_elem1, char*
    ↪ elem2, int t_elem2);
int search(char* string, char element);
char* str_tok(char** test, char* delim);
```

```
int add_bot(robot bot, robot_liste* listOfBot);
int suppr_bot(char id, robot_liste* listOfBot);
int add_bullet(bullet bullet, bullet_liste* listOfBullet);
int suppr_bullet(bullet bullet, bullet_liste* listOfBullet);
robot* search_robot(char id, robot_liste listOfBot);
float distance(coord p1, coord p2);
int affect_dico(char* name, int data, aff** dico);
aff* search_in_dico(char* var ,aff* dico);

/* fonctions de interpreteur.c */
char* get_line(FILE *fd);
cmd create_cmd(char **ligne, FILE *fd);

/* structure pour stocker les coordonnées des elements sur la
    ↪ map */
struct coordonnees{
    float x;
    float y;
};

/* structure pour stocker la map */
struct map{
    int nbSpawn; /* nombre de spawn de la map */
    coord* spawn; /* liste des spawn de la map */
    int width; /* largeur de la map */
    int height; /* hauteur de la map */
    char* map ; /* tableau pour stocker la map */
};

/* structure pour stocker un robot */
struct robot{
    char* name; /* nom du script du robot */
    char id; /* num de la file_de_message */
    char winner; /* indicateur pour pouvoir gagner */
    char wait_player; /* indicateur pour etre en attente */
    char reach; /* champ de vision */
    char pick; /* porté de ramassage */
    coord pos; /* position */
    char direction; /* N=0 E=1 S=2 O=3 */
    char pv; /* points de vie */
    int speed; /* vitesse de déplacemment mm/s a diviser par
        ↪ 1000 */
    char bullet_damage; /* puissance de l'attaque àdistance */
```

```
    int speed_bullet; /* vitesse de la balle pour le robot
        ↪ basique mm/s a diviser par 1000 */
    inventaire* inventory; /* inventaire du robot */
};

/* structure maillon pour la liste chainé de robot */
struct elem_robot{
    robot element; /* element robot du maillon */
    robot_liste suite; /* pointeur sur la suite de la liste */
};

/* structure pour stocker les items ramasser par les robots */
struct inventaire{
    short nb_bullet; /* nombre de balle du robot */
    unsigned long long int money; /* scors du robot */
    short armor; /* nombre d'armure du robot */
};

/* structure pour stocker les données d'une balle */
struct bullet{
    char size; /* taille de la balle */
    coord pos; /* position */
    float speed_x; /* vitesse de la balle en x */
    float speed_y; /* vitesse de la balle en y */
    char damage; /* dégats de la balle */
};

/* structure maillon pour la liste chainé de balle */
struct elem_bullet{
    bullet element; /* element balle du maillon */
    bullet_liste suite; /* pointeur sur la suite de la liste */
};

/* structure pour stocker les données d'un coffre */
struct chest{
    char size; /* taille du coffre */
    coord pos; /* position */
    char value; /* L'ARGENT !!!!!! dans le coffre */
};

/* structure de header de message a envoyer au server */
struct message{
    char client; /* id du client qui envoie le message */
    char action; /* action demander par le client */
```

```
};

/* structure des commandes pour l'interpréteur */
struct commande {
    char* name; /* le nom de la commande */
    int nb_args; /* nombre d'arguments */
    int nb_subcom; /* nombre de sous commandes */
    cmd* subcom; /* tableau des sous commandes */
};

/* structure des affectations variable du joueur */
struct affect {
    char* name; /* le nom de la variable */
    int data; /* valeur de la variable */
    aff* next; /* struct de la prochaine affectation */
};


#endif /* GAME_H_ */
```

A.2 Fonctions utilitaires

```
#include "game.h"

/* fichier de fonction utiliser dans tout les executables */

/* fonction de creation des robots */
robot create_robot(char* name, char id, coord spawn,
    ↪ inventaire* inventaire){
    robot new_robot;
    new_robot.name = malloc(strlen(name));
    strcpy(new_robot.name, name);
    new_robot.id = id;
    new_robot.reach = 10;
    new_robot.winner = 0;
    new_robot.wait_player = 0;
    new_robot.pick = 2;
    new_robot.pos = spawn;
    new_robot.direction = 0;
    new_robot.pv = 100;
    new_robot.speed = 5;
    new_robot.bullet_damage = 10;
    new_robot.speed_bullet = 200;
    new_robot.inventory = inventaire;
    return new_robot;
}

/* fonction de creation des balles */
bullet create_bullet(robot *bot, float speed_x, float speed_y)
    ↪ {
    bullet new_bullet;
    new_bullet.size = 0;
    new_bullet.speed_x = speed_x;
    new_bullet.speed_y = speed_y;
    new_bullet.pos.x = (int) ((bot->pos.x + CYCLE*speed_x));
    new_bullet.pos.y = (int) ((bot->pos.y + CYCLE*speed_y));
    new_bullet.damage = bot->bullet_damage;
    return new_bullet;
}

/* fonction pour concat les msgs avant de les envoyer */
void str_concat(char* str, char* elem1, int t_elem1, char*
    ↪ elem2, int t_elem2){
    for (int i = 0; i < t_elem1+t_elem2; i++) {
```

```
        if(i<t_elem1){
            str[i] = elem1[i];
        }else{
            str[i] = elem2[i-t_elem1];
        }
    }
}

/* fonction qui cherche la presence d'un caractere dans un
    ↪ string */
int search(char* string, char element){
        for(int i = 0; i < strlen(string); i++){
                if(string[i] == element) return 0;
        }
        return 1;
}

/* fonction qui separe la string selon des separateur */
char* str_tok(char** str, char* delim){
        char* start_str = NULL;
        int i = 0;
        if (str == NULL) return NULL;
        if (*str == NULL) return NULL;
        while (*(*str+i) != '\0') {
                if (search(delim, *(*str+i)) != 0 && start_str
                    ↪ == NULL) {
                        start_str = *str+i;
                }
                if (search(delim, *(*str+i)) == 0 && start_str
                    ↪ != NULL) {
                        *(*str+i) = '\0';
                        *str = *str+i+1;
                        return start_str;
                }
                i++;
        }
        *str = NULL;
        return start_str;
}

/* fonction pour ajouter des robot a la liste */
int add_bot(robot bot, robot_liste* listOfBot){
    elem_robot* new_bot = malloc(sizeof(elem_robot));
    new_bot->element = bot;
```

```
    new_bot->suite = *listOfBot;
    *listOfBot = new_bot;
    return 0;
}

/* fonction pour supprimer un robot de la liste avec l'id */
int suppr_bot(char id, robot_liste* listOfBot){
    if (*listOfBot == NULL) return EXIT_FAILURE;
    if ((*listOfBot)->element.id == id) {
        elem_robot* tmp = *listOfBot;
        *listOfBot = (*listOfBot)->suite;
        free(tmp);
        return 0;
    }
    for (elem_robot* actuel = *listOfBot; actuel->suite != NULL
        ↪ ; actuel = actuel->suite) {
        if (actuel->suite->element.id == id) {
            elem_robot* tmp = actuel->suite;
            actuel->suite = actuel->suite->suite;
            free(tmp);
            return 0;
        }
    }
    return EXIT_FAILURE;
}

/* fonction pour ajouter une balle a la liste */
int add_bullet(bullet bullet, bullet_liste* listOfBullet){
    elem_bullet* new_bullet = malloc(sizeof(elem_bullet));
    new_bullet->element = bullet;
    new_bullet->suite = *listOfBullet;
    *listOfBullet = new_bullet;
    return 0;
}

/* fonction pour supprimer une balle */
int suppr_bullet(bullet bullet, bullet_liste* listOfBullet){
    if (*listOfBullet == NULL) return EXIT_FAILURE;
    if ((*listOfBullet)->element.pos.x == bullet.pos.x && (*
        ↪ listOfBullet)->element.pos.y == bullet.pos.y) {
        elem_bullet* tmp = *listOfBullet;
        *listOfBullet = (*listOfBullet)->suite;
        free(tmp);
        return 0;
```

```
    }
    for (elem_bullet* actuel = *listOfBullet; actuel->suite !=
        ↪ NULL; actuel = actuel->suite) {
        if (actuel->suite->element.pos.x == bullet.pos.x &&
            ↪ actuel->suite->element.pos.y == bullet.pos.y) {
            elem_bullet* tmp = actuel->suite;
            actuel->suite = actuel->suite->suite;
            free(tmp);
            return 0;
        }
    }
    return EXIT_FAILURE;
}

/* fonction pour chercher un robot avec l'id */
robot* search_robot(char id, robot_liste listOfBot){
    if (listOfBot != NULL) {
        if (listOfBot->element.id == id) return &(listOfBot->
            ↪ element);
        robot_liste tmp_list = listOfBot;
        while (tmp_list->suite) {
            if (tmp_list->suite->element.id != id) tmp_list =
                ↪ tmp_list->suite;
            else return &(tmp_list->suite->element);
        }
    }
    return NULL;
}

/* fonction pour calculer la distance entre deux points */
float distance(coord p1, coord p2){
  return (float) sqrt(pow(p2.x-p1.x,2)+pow(p2.y-p1.y,2));
}

/* fonction du dictionnaire */
int affect_dico(char* name, int data, aff** dico){
  aff* new_aff;
  aff* i;

  new_aff = malloc(sizeof(aff));
  new_aff->name = malloc(strlen(name));
  strcpy(new_aff->name,name);
  new_aff->data = data;
  new_aff->next = NULL;
```

```
  if (dico == NULL) return EXIT_FAILURE;
  if (*dico == NULL) {
    *dico = new_aff;
    return 0;
  }
  if (strcmp((*dico)->name, name) == 0) {
    (*dico)->data = data;
    return 0;
  }
  for (i = *dico; i->next != NULL; i = i->next) {
    if (strcmp(i->next->name, name) == 0) {
      (*dico)->data = data;
      return 0;
    }
  }
  i->next = new_aff;
  return 0;
}

/* fonction de recherche dans un dico */
aff* search_in_dico(char* var ,aff* dico){
  while (dico != NULL) {
    if(strcmp(dico->name, var) == 0){
      return dico;
    }
    dico = dico->next;
  }
  return NULL;
}
```

A.3 Le client

```
#include "game.h"

int main(int argc, char *argv[]) {
    //verification des arguments
    if (argc != 2) {
        fprintf(stderr, "usage: %s joueur_name\n", argv[0]);
        return 1;
    }

    if(client(argv[1])){
        fprintf(stderr, "client : exit failure\n");
        return 1;
    }

    return 0;
}


//fonction qui traite les infos de la file de message
int reception(mqd_t fdem, char** buffer, int taille, robot*
    ↪ bot, char obj){
  msg message;
  char done;
  int timer;
  char tmp_buf[taille];
  struct timespec tw;

  done = 1;
  clock_gettime(CLOCK_REALTIME,&tw);
  while (done) {
    if (mq_timedreceive(fdem,tmp_buf,taille,NULL,&tw) > 0){
      message = *((msg*) tmp_buf);
      if (message.client == -1) {
        bot->pv -= message.action;
        bot->reach = (int) (bot->reach * bot->pv/100);
        bot->speed = (int) (bot->speed * bot->pv/100);
        if (bot->pv <= 0) {
          bot->winner = -1;
          return -1;
        }
      }else {
        if (message.action == -1) {
```

```
          printf("en␣attente␣des␣joueur\n");
          bot->wait_player = 1;
          return -1;
        } else if (message.action == 0) {
          bot->winner = 1;
          return -1;
        } else if (message.action == 4) {
          timer = *((int*) &(tmp_buf[sizeof(msg)]));
          printf("reprise␣de␣la␣partie␣dans␣%d␣sec\n", timer);
          if (timer == 0) {
            return 0;
          }
        } else if (message.action == obj && obj != 0) {
          *buffer = tmp_buf;
          return 1;
        }
      }
    }else if (obj == 0) {
      done = 0;
    }
  }
  return -1;
}

/* fonction de gestion du client */
int client(char* name){
    mqd_t server, client;
    msg message;
    int taille, done;
    char *buffer, *FdeM, *concat_msg;
    char *com_scan, *exec_com;
    struct mq_attr attr;
    robot bot;
    inventaire inventory;
    aff *dico;

    server = mq_open("/server",O_WRONLY,0600,NULL);
    client = mq_open("/new_Client",O_RDONLY,0600,NULL);
    if (server == -1 || client == -1) {
        fprintf(stderr, "server␣offline\n");
        return EXIT_FAILURE;
    }

    if (mq_getattr(server,&attr)) {
```

```
    perror("mq_getattr");
    return EXIT_FAILURE;
}

taille = attr.mq_msgsize;
buffer = malloc(taille);
concat_msg = malloc(taille);

message.client = -1;
message.action = -1;
str_concat(concat_msg,(char*) &message,sizeof(msg),name,
    ↪ strlen(name)+1);
mq_send(server,concat_msg,taille,1);
mq_receive(client,buffer,taille,0);
message = *((msg*) buffer);
if (message.client == -1) {
    fprintf(stderr, "game␣is␣full\n");
    return EXIT_FAILURE;
}
FdeM = calloc(0,3);
FdeM[0] = '/';
sprintf(FdeM+1,"%d",message.client);
mq_close(client);

client = mq_open(FdeM,O_RDONLY,0600,NULL);
if (client == -1) {
    perror("mq_open");
    return EXIT_FAILURE;
}
inventory.nb_bullet = 100;
inventory.money = 0;
inventory.armor = 0;
dico = NULL;
done = 1;
bot = create_robot(name, message.client, *((coord*) &(
    ↪ buffer[sizeof(msg)])), &inventory);
com_scan = malloc(40);
while (reception(client,&buffer,taille,&bot,0) < 0);
while (done) {
    printf("commande␣robot␣%d␣:␣",bot.id);
    com_scan = realloc(0,40);
    fgets(com_scan,40,stdin);
    exec_com = malloc(strlen(com_scan));
    strcpy(exec_com,com_scan);
```

```
        reception(client,&buffer,taille,&bot,0);
        if (bot.winner == 1){
          printf("GAGNÉ\n");
          done = 0;
          break;
        }else if (bot.winner == -1) {
          printf("PERDU\n");
          done = 0;
          break;
        }
        printf("return␣%d\n", interp(create_cmd(&exec_com,NULL)
            ↪ ,&bot,server,client,buffer,taille,&dico));
    }
    mq_close(client);
    mq_unlink(FdeM);
    return 0;
}
```

A.4 Le serveur

```
#include "game.h"

int main(int argc, char *argv[]) {
    /* verification des arguments pour eviter les erreur */
    if(argc != 2){
      fprintf(stderr, "usage: %s nom_map\n", argv[0]);
      return 1;
    }

    if(server(argv[1])){
        fprintf(stderr, "server : exit failure\n");
        return 1;
    }

    return 0;
}

/* fonction de gestion du server */
int server(char* map_name){
    map mapOfGame;
    robot_liste listOfBot;
    bullet_liste listOfBullet;
    mqd_t server, new_Client, *list_mqueue;
    int size_msg_receive, cycle_display, mvp, add, nbclient,
        ↪ done, time_set, random;
    char* buffer, *place, *concat_msg;
    msg demande;
    robot* cur_bot;
    coord buffer_pos;
    struct mq_attr mqueue_attr;
    struct timespec tw, current_time, time_out;

    /* on verifie que les files de messages viennent d'etre
        ↪ ouverte */
    if (create_map(map_name,&mapOfGame)) return EXIT_FAILURE;
    server = mq_open("/server", O_RDONLY, 0600, NULL);
    if (server != -1){
        close(server);
        mq_unlink("/server");
    }
    new_Client = mq_open("/new_Client", O_RDONLY, 0600, NULL);
    if (new_Client != -1) {
```

```
    close(new_Client);
    mq_unlink("new_Client");
}
server = mq_open("/server", O_RDONLY | O_CREAT, 0600, NULL)
    ↪ ;
new_Client = mq_open("/new_Client", O_WRONLY | O_CREAT,
    ↪ 0600, NULL);
if (mq_getattr(server,&mqueue_attr)) {
    perror("mq_getattr");
    return EXIT_FAILURE;
}
size_msg_receive = mqueue_attr.mq_msgsize;
buffer = malloc(size_msg_receive);
listOfBot = NULL;
listOfBullet = NULL;
list_mqueue = malloc(mapOfGame.nbSpawn * sizeof(mqd_t));
place = calloc(0,mapOfGame.nbSpawn);
concat_msg = malloc(size_msg_receive);
nbclient = cycle_display = time_set = 0;
mvp = -1;
done = 1;
clock_gettime(CLOCK_REALTIME,&tw);
/* boucle principale */
while (done) {
  cycle_display++;
  if (nbclient == 0) {
    clock_gettime(CLOCK_REALTIME,&current_time);
    if (!time_set) {
      time_out = current_time;
      time_out.tv_sec += 15;
      time_set++;
    }else {
      if (current_time.tv_sec > time_out.tv_sec) {
        done = 0;
      }
    }
  }else time_set = 0;
  if (mq_timedreceive(server,buffer,size_msg_receive,NULL,&
      ↪ tw) > 0) {
    demande = *((msg*) buffer);
    cur_bot = search_robot(demande.client,listOfBot);
    if (cur_bot == NULL && demande.action == -1) {
      add = search_place(place,mapOfGame.nbSpawn);
      if (add == -1) {
```

```
      mq_send(new_Client,(char*) &demande, sizeof(msg),
          ↪ 1);
    }else{
      nbclient++;
      char* id = calloc(0,((int) (mapOfGame.nbSpawn/10))
          ↪ +1);
      id[0] = '/';
      sprintf(id+1,"%d",add);
      list_mqueue[add] = mq_open(id,O_WRONLY,0600,NULL);
      if (list_mqueue[add] != -1) {
        close(list_mqueue[add]);
        mq_unlink(id);
      }
      list_mqueue[add] = mq_open(id,O_WRONLY | O_CREAT
          ↪ ,0600,NULL);
      cur_bot = malloc(sizeof(robot));
      *cur_bot = create_robot(buffer+sizeof(msg),add,
          ↪ mapOfGame.spawn[add],NULL);
      add_bot(*cur_bot,&listOfBot);
      demande.client = add;
      demande.action = 1;
      str_concat(concat_msg,(char*) &demande,sizeof(msg)
          ↪ ,(char*) &(search_robot(add,listOfBot)->pos),
          ↪ sizeof(coord));
      if (mq_send(new_Client,concat_msg,sizeof(msg)+
          ↪ sizeof(coord),1) < 0) {
        perror("mq_send");
        return 1;
      }
      place[add] = 1;
      free(id);
      if (nbclient == 2) {
        start(list_mqueue);
      }
    }
  }else{
    if (demande.action == 1) {
      printf("suppr\n");
      mq_close(list_mqueue[(int) demande.client]);
      suppr_bot(demande.client,&listOfBot);
      place[(int) demande.client] = 0;
      nbclient -=1;
      if (nbclient == 1) {
        demande.client = listOfBot->element.id;
```

```
    demande.action = -1;
    mq_send(list_mqueue[(int) demande.client],(char*)
        ↪ &demande,sizeof(msg),1);
  }
}else if (nbclient == 1) {
  demande.client = listOfBot->element.id;
  demande.action = -1;
  mq_send(list_mqueue[(int) demande.client],(char*) &
      ↪ demande,sizeof(msg),1);
}else if (demande.action == 2) {
  buffer_pos = *((coord*) &(buffer[sizeof(msg)]));
  if (mapOfGame.map[((int) (buffer_pos.y+0.5))*
      ↪ mapOfGame.width+((int) (buffer_pos.x+0.5))]
      ↪ != 'W') {
    cur_bot->pos = buffer_pos;
  }
  str_concat(concat_msg,(char*) &demande,sizeof(msg)
      ↪ ,(char*) &(cur_bot->pos),sizeof(coord));
  mq_send(list_mqueue[(int) demande.client],
      ↪ concat_msg,sizeof(msg)+sizeof(coord),1);
}else if (demande.action == 3) {
  buffer_pos = bot_interact(mapOfGame,listOfBot,
      ↪ cur_bot,NULL);
  char* tmp_msg = malloc(sizeof(msg)+1);
  random = (int)(rand() / (double)RAND_MAX * (10 - 1)
      ↪ );
  str_concat(tmp_msg,(char*) &demande,sizeof(msg),&
      ↪ mapOfGame.map[((int) buffer_pos.y)*mapOfGame.
      ↪ width+((int) buffer_pos.x)],1);
  str_concat(concat_msg,tmp_msg,sizeof(msg)+1,(char*)
      ↪  &random,sizeof(int));
  mq_send(list_mqueue[(int) demande.client],
      ↪ concat_msg,sizeof(msg)+1+sizeof(int),1);
  mapOfGame.map[((int) (buffer_pos.y+0.5))*mapOfGame.
      ↪ width+((int) (buffer_pos.x+0.5))] = '␣';
}else if (demande.action == 4) {
  cur_bot->direction = buffer[sizeof(msg)];
}else if (demande.action == 5) {
  add_bullet(create_bullet(cur_bot,((coord*) &(buffer
      ↪ [sizeof(msg)]))->x,((coord*) &(buffer[sizeof(
      ↪ msg)]))->y),&listOfBullet);
}else if (demande.action == 6) {
  buffer_pos = bot_interact(mapOfGame,listOfBot,
      ↪ cur_bot,buffer);
```

```
            str_concat(concat_msg,(char*) &demande,sizeof(msg)
                ↪ ,(char*) &buffer_pos,sizeof(coord));
            mq_send(list_mqueue[(int) demande.client],
                ↪ concat_msg,sizeof(msg)+sizeof(coord),1);
          }
        }
      }
      move_bullet(&listOfBullet,&listOfBot,mapOfGame,
          ↪ list_mqueue);
      if (nbclient > 1) {
        mvp = win(listOfBot);
      }
      if (mvp != -1) {
        done = 0;
        demande.client = mvp;
        demande.action = 0;
        printf("{%d,%d}\n",demande.client,demande.action);
        if (mq_send(list_mqueue[mvp],(char*) &demande,sizeof(
            ↪ msg),1) < 0) perror("mq_send");
        cur_bot = search_robot(mvp,listOfBot);
        printf("%s␣à␣gagné␣!!!!\n", cur_bot->name);
      }
      if (cycle_display > CYCLE && nbclient > 1) {
        printf("\f");
        affichage(mapOfGame,listOfBot,listOfBullet);
        cycle_display = 0;
      }
    }
    mq_unlink("/server");
    mq_unlink("new_Client");
    return 0;
}

/* fonction d'interaction entre la map et les joueurs (sert a
    ↪ detecter un objet pour pick et seek) */
coord bot_interact(map mapOfGame, robot_liste listOfBot, robot
    ↪ * bot, char* buffer){
  coord pos_object;
  robot* tmp_bot;
  int range;

  pos_object = bot->pos;
  if (buffer == NULL)
    range = bot->pick;
```

```
  else
    range = bot->reach;
  for (int r = 0; r < range; r++) {
    for (int i = bot->pos.y-r; i < bot->pos.y+r; i++) {
      for (int j = bot->pos.x-r; j < bot->pos.x+r; j++) {
        if (buffer == NULL) {
          if (search("ABCL",mapOfGame.map[i*mapOfGame.width+j])
              ↪  == 0) {
            pos_object.x = j;
            pos_object.y = i;
            return pos_object;
          }
        }else if (buffer[sizeof(msg)] == 'R') {
          tmp_bot = isBot(j,i,listOfBot);
          if (tmp_bot != NULL && (tmp_bot->pos.x != bot->pos.x
              ↪ || tmp_bot->pos.y != bot->pos.y)) {
            return tmp_bot->pos;
          }
        }else if (mapOfGame.map[i*mapOfGame.width+j] == buffer[
            ↪ sizeof(msg)]) {
          pos_object.x = j;
          pos_object.y = i;
          return pos_object;
        }
      }
    }
  }
  return pos_object;
}

/* fonction start pour le decompte des 5 secondes avant le jeu
   ↪  */
void start(mqd_t* list_mqueue) {
  int sec;
  msg message;
  char concat_msg[sizeof(msg)+sizeof(int)];

  sec = 5;
  message.action = 4;
  while (sec >= 0) {
    message.client = 0;
    str_concat(concat_msg,(char*) &message,sizeof(msg),(char*)
        ↪ &sec,sizeof(int));
```

```
    mq_send(list_mqueue[(int) message.client],concat_msg,sizeof
        ↪ (msg)+sizeof(int),1);
    message.client = 1;
    str_concat(concat_msg,(char*) &message,sizeof(msg),(char*)
        ↪ &sec,sizeof(int));
    mq_send(list_mqueue[(int) message.client],concat_msg,sizeof
        ↪ (msg)+sizeof(int),1);
    sleep(1);
    sec--;
  }
}

/* fonction de deplacement des balles */
void move_bullet(bullet_liste* list_bullet, robot_liste*
    ↪ bot_list, map mapOfGame, mqd_t* list_mqueue){
    bullet_liste tmp_list;
    coord tmp_coord;
    robot* tmp_bot;
    msg message;

    tmp_list = *list_bullet;
    while (tmp_list) {
        tmp_coord.x = tmp_list->element.pos.x + tmp_list->
            ↪ element.speed_x;
        tmp_coord.y = tmp_list->element.pos.y + tmp_list->
            ↪ element.speed_y;
        tmp_bot = isBot((int) tmp_coord.x, (int) tmp_coord.y, *
            ↪ bot_list);
        if(tmp_bot){
            tmp_bot->pv -= tmp_list->element.damage;
            tmp_bot->reach = (int) (tmp_bot->reach * tmp_bot->
                ↪ pv/100);
            tmp_bot->speed = (int) (tmp_bot->speed * tmp_bot->
                ↪ pv/100);
            suppr_bullet(tmp_list->element,list_bullet);
            message.client = -1;
            message.action = tmp_list->element.damage;
            mq_send(list_mqueue[(int) tmp_bot->id],(char*) &
                ↪ message,sizeof(msg),1);
        }else if (search("wW",mapOfGame.map[((int) tmp_coord.y)
            ↪ *mapOfGame.width+((int) tmp_coord.x)]) == 0) {
            suppr_bullet(tmp_list->element,list_bullet);
        }else{
            tmp_list->element.pos = tmp_coord;
```

```
        }
        tmp_list = tmp_list->suite;
    }
}

/* fonction d'affichage printf() */
void affichage(map mapOfGame, robot_liste listOfBot,
    ↪ bullet_liste listOfBullet){
  coord test_range;
  for (int y = 0; y < mapOfGame.height; y++) {
    for (int x = 0; x < mapOfGame.width; x++) {
      robot* tmp = isBot(x, y, listOfBot);
      if (tmp != NULL) {
          if (tmp->pv < 0) {
            printf("O");
          }else{
            switch (tmp->direction) {
              case 0:
                printf("^");
                break;
              case 1:
                printf(">");
                break;
              case 2:
                printf("v");
                break;
              case 3:
                printf("<");
                break;
            }
          }
      }else if (isBullet(x,y,listOfBullet)) {
        printf("*");
      }else{
        test_range.x = (float) x;
        test_range.y = (float) y;
        if (in_range(test_range,listOfBot)==0) {
          printf("?");
        }else{
          printf("%c", mapOfGame.map[y*mapOfGame.width+x]);
        }
      }
    }
    printf("\n");
```

```
  }
}

/* fonction qui test si un robot est present au coordonnées x,
    ↪  y */
robot* isBot(int x, int y, robot_liste listOfBot){
    robot_liste tmp_list = listOfBot;
    while (tmp_list != NULL) {
        if ( (int) (tmp_list->element.pos.x+0.5) == x && (int)
            ↪ (tmp_list->element.pos.y+0.5) == y ) return &(
            ↪ tmp_list->element);
        tmp_list = tmp_list->suite;
    }
    return NULL;
}

/* fonction qui test si il y a une balle au coordonnées x, y
    ↪ */
int isBullet(int x, int y, bullet_liste listOfBullet){
    bullet_liste tmp_list = listOfBullet;
    while (tmp_list != NULL) {
        if ( (int) (tmp_list->element.pos.x+0.5) == x && (int)
            ↪ (tmp_list->element.pos.y+0.5) == y ) return 1;
        tmp_list = tmp_list->suite;
    }
    return 0;
}

/* fonction pour savoir si le point est dans le range d'un bot
    ↪  */
int in_range(coord pos,robot_liste listOfBot){
  int distance;
  while(listOfBot){
    distance = sqrt(pow(listOfBot->element.pos.x-pos.x,2)+pow(
        ↪ listOfBot->element.pos.y-pos.y,2));
    if (distance < listOfBot->element.reach) return 1;
    listOfBot = listOfBot->suite;
  }
  return 0;
}

/* fonction pour detecter quelle joueur a gagner */
int win(robot_liste listOfBot){
    int winner = -1;
```

```
    while (listOfBot != NULL) {
        if (winner == -1 && listOfBot->element.pv > 0) {
            winner = listOfBot->element.id;
        }else if (listOfBot->element.pv > 0) {
            winner = -1;
            break;
        }
        listOfBot = listOfBot->suite;
    }
    return winner;
}

/* fonction qui renvois une place sur la map si il y en a */
int search_place(char* place,int nb_place){
    for (int i = 0; i < nb_place; i++) {
        if (place[i] == 0) return i;
    }
    return -1;
}


/* fonction de creation de la map */
int create_map(char* path_file, map* new_map){
    char c;
    FILE* f = NULL;
    f = fopen(path_file, "r");
    if(f == NULL) return -1;
    fseek(f,0,SEEK_SET);
    int tmp = 0;
    do {
        c = fgetc(f);
        tmp++;
    } while(c != 10);
    new_map->width = tmp;
    fseek(f,0,SEEK_END);
    new_map->height = ftell(f) / new_map->width;
    new_map->map = malloc(new_map->width * new_map->height *
        ↪ sizeof(char));
    int i = 0, j = 0;
    new_map->spawn = malloc(sizeof(coord));
    new_map->nbSpawn = 0;
    fseek(f,0,SEEK_SET);
    c = fgetc(f);
    while(c != EOF){
```

```
        if(c == 10){
            i++;
            j=0;
        }else{
            if(c == 'S'){
                coord tmp = {j,i};
                new_map->spawn = realloc(new_map->spawn,(new_map
                    ↪ ->nbSpawn+1)*sizeof(coord));
                new_map->spawn[new_map->nbSpawn] = tmp;
                new_map->nbSpawn++;
                c = '␣';
            }
            new_map->map[(i * new_map->width) + j] = c;
            j++;
        }
        c = fgetc(f);
    }
    if(fclose(f)) return -1;
    return 0;
}
```

A.5 Les fonctions minimalistes

```
#include "game.h"

/* fonction associé àla commande "coord" */
float get_coord(robot *bot, char *axis){
  if (strcmp(axis,"x")==0)
    return bot->pos.x;
  else if (strcmp(axis,"y")==0)
    return bot->pos.y;
  return 0;
}

/* fonction associé àla commande "steer" */
short get_direction(robot *bot){
  return bot->direction;
}

/* fonction associé àla commande "pv" */
short get_pv(robot *bot){
  return bot->pv;
}

/* fonction associé àla commande "money" */
unsigned long long get_money(robot *bot){
  return bot->inventory->money;
}

/* fonction associé àla commande "nb_bullet" */
short get_nb_bullet(robot *bot){
  return bot->inventory->nb_bullet;
}

/* fonction associé àla commande "armor" */
short get_armor(robot *bot){
  return bot->inventory->armor;
}

/* fonction associé àla commande "move x" */
int avancer(robot *bot, int move, mqd_t server, mqd_t client,
    ↪ char* buffer, int taille) {
  float* modif_axis, speed;
  msg message;
  char concat_msg[sizeof(msg)+sizeof(coord)];
```

```
int recep, dist_obj;
coord last_pos, init_pos;
float d1, d2;
struct timespec remain, request;

message.client = bot->id;
message.action = 2;
speed = ((float) bot->speed) / CYCLE;

switch (bot->direction) {
  case 0:
    modif_axis = &(bot->pos.y);
    speed = (-1 * speed * move)/fabs(move);
    dist_obj = fabs((bot->pos.y-move)-bot->pos.y);
    break;
  case 1:
    modif_axis = &(bot->pos.x);
    speed = (speed * move)/fabs(move);
    dist_obj = fabs((bot->pos.y+move)-bot->pos.y);
    break;
  case 2:
    modif_axis = &(bot->pos.y);
    speed = (speed * move)/fabs(move);
    dist_obj = fabs((bot->pos.y+move)-bot->pos.y);
    break;
  case 3:
    modif_axis = &(bot->pos.x);
    speed = (-1 * speed * move)/fabs(move);
    dist_obj = fabs((bot->pos.x-move)-bot->pos.x);
    move *= -1;
    break;
}

init_pos = bot->pos;
while(distance(init_pos,bot->pos) < dist_obj) {
  last_pos = bot->pos;
  d1 = distance(init_pos,last_pos);
  *modif_axis += speed;
  clock_gettime(CLOCK_REALTIME,&request);
  request.tv_nsec += 1;
  clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME,&request,&
      ↪ remain);
  str_concat(concat_msg,(char*) &message,sizeof(msg),(char*)
      ↪ &(bot->pos),sizeof(coord));
```

```
    mq_send(server,concat_msg,sizeof(msg)+sizeof(coord),1);
    recep = reception(client,&buffer,taille,bot,2);
    if (recep != 1) return recep;
    bot->pos = *((coord*) &(buffer[sizeof(msg)]));
    d2 = distance(init_pos,bot->pos);
    if (d1 >= d2) {
      return -1;
    }
  }

  str_concat(concat_msg,(char*) &message,sizeof(msg),(char*)
     ↪ &(bot->pos),sizeof(coord));
  mq_send(server,concat_msg,sizeof(msg)+sizeof(coord),1);
  recep = reception(client,&buffer,taille,bot,2);
  if (recep != 1) return recep;
  bot->pos = *((coord*) &(buffer[sizeof(msg)]));
  return 0;
}

/* fonction associé àla commande "aim x y" */
int aim(robot *bot, int x, int y){
  double angle;
  int ix, igrec;
  double argshit=0;
  double tbl_angle_convert[4] = {90,0,270,180};
  ix = x - bot->pos.x;
  igrec = y - bot->pos.y;
  if (ix == 0) {
    angle = 90;
  }else{
    angle = atan(igrec / ix) / RAD;
  }
  if (x - bot->pos.x >= 0) {
    argshit = 0;
  }else{
    argshit = 180;
  }
  angle = angle + tbl_angle_convert[(int) bot->direction] +
     ↪ argshit;
  return angle;
}

/* fonction associé àla commande "seek 0 a" */
```

```
int seek(robot *bot, char *obj, char *axis, mqd_t server,
    ↪ mqd_t client, char* buffer, int taille){
    msg message;
    char concat_msg[sizeof(msg)+1];
    coord pos_object;
    int recep;

    message.client = bot->id;
    message.action = 6;

    str_concat(concat_msg,(char*) &message,sizeof(msg),obj,1);
    mq_send(server,concat_msg,sizeof(msg)+1,1);
    recep = reception(client,&buffer,taille,bot,6);
    if (recep != 1) return recep;
    pos_object = *((coord*) &(buffer[sizeof(msg)]));
    if (strcmp(axis, "x") == 0) {
        return pos_object.x;
    }else if (strcmp(axis, "y") == 0) {
        return pos_object.y;
    }
    return 0;
}

/* fonction associé àla commande "pick" */
int ramasser(robot *bot, mqd_t server, mqd_t client, char*
    ↪ buffer, int taille){
  int recep;
  msg message;

  message.client = bot->id;
  message.action = 3;
  mq_send(server,(char*) &message,sizeof(msg),1);
  recep = reception(client,&buffer,taille,bot,3);
  if (recep != 1)
    return recep;
  if (buffer[sizeof(msg)] == 'C' ) {
    bot->inventory->money += *((int*) &(buffer[sizeof(msg)+1]))
        ↪ ;
    return 0;
  }
  if (buffer[sizeof(msg)] == 'A') {
    bot->inventory->armor += *((int*) &(buffer[sizeof(msg)+1]))
        ↪ ;
    return 0;
```

```
  }
  if (buffer[sizeof(msg)] == 'B') {
    bot->inventory->nb_bullet += *((int*) &(buffer[sizeof(msg)
        ↪ +1]));
    return 0;
  }
  if (buffer[sizeof(msg)] == 'L') {
    bot->pv += *((int*) &(buffer[sizeof(msg)+1]));
    return 0;
  }
  return -1;
}

/* fonction associé àla commande "turn x" */
int tourner(robot *bot, short direc, mqd_t server){
  bot->direction = (bot->direction + direc) % 4;
  if (bot->direction < 0) {
    bot->direction = 4 + bot->direction;
  }
  msg message = {bot->id,4};
  char* tmp_msg = malloc(sizeof(msg) + sizeof(char));
  str_concat(tmp_msg, (char*) &message, sizeof(msg), &(bot->
      ↪ direction), sizeof(char));
  mq_send(server,tmp_msg,sizeof(msg)+sizeof(char),1);
  free(tmp_msg);
  return 0;
}

/* fonction associé àla commande "shoot x" */
int tirer(robot *bot, float angle, mqd_t server){
    int cycle = CYCLE*100;
    //printf("demande de tir\n");
    if (bot->inventory->nb_bullet > 0) {
        bot->inventory->nb_bullet -= 1;
        coord speed = { (float) cos((angle+bot->direction
            ↪ *90-90)*RAD)/cycle*bot->speed_bullet, (float)
            ↪ sin((angle+bot->direction*90-90)*RAD)/cycle*bot
            ↪ ->speed_bullet};
        //printf("speed %f , %f\n", speed.x,speed.y);
        msg message = {bot->id,5};
        char* tmp_msg = malloc(sizeof(msg)+sizeof(coord));
        str_concat(tmp_msg, (char*) &message, sizeof(msg), (
            ↪ char*) &speed, sizeof(coord));
        mq_send(server, tmp_msg, sizeof(msg)+sizeof(coord), 1);
```

```
        return 0;
    }
    return -1;
}

/* fonction d'evaluation des commandes */
int eval(cmd sub_com, robot *bot, mqd_t server, mqd_t client,
    ↪ char* buffer, int taille, aff **dico){
  if (sub_com.name == NULL) {
    return -1;
  }else if (strcmp(sub_com.name, "quit") == 0) {
    msg message = {bot->id,1};
    mq_send(server, (char*) &message, sizeof(msg), 1);
    bot->winner = -1;
  }else if(strcmp(sub_com.name, "move") == 0)
    return avancer(bot,eval(sub_com.subcom[0], bot,server,
        ↪ client,buffer,taille,dico),server,client,buffer,
        ↪ taille);

  else if(strcmp(sub_com.name, "pick") == 0)
    return ramasser(bot, server, client, buffer, taille);

  else if(strcmp(sub_com.name, "turn") == 0)
    return tourner(bot,eval(*sub_com.subcom, bot,server,client,
        ↪ buffer,taille,dico),server);

  else if(strcmp(sub_com.name, "shoot") == 0)
    return tirer(bot,eval(*sub_com.subcom, bot,server,client,
        ↪ buffer,taille,dico),server);

  else if(strcmp(sub_com.name, "aim") == 0)
    return aim(bot,eval(sub_com.subcom[0], bot,server,client,
        ↪ buffer,taille,dico),eval(sub_com.subcom[1], bot,
        ↪ server,client,buffer,taille,dico));

  else if(strcmp(sub_com.name, "seek") == 0)
    return seek(bot, sub_com.subcom[0].name, sub_com.subcom[1].
        ↪ name, server, client, buffer, taille);

  else if(strcmp(sub_com.name, "pv") == 0)
    return get_pv(bot);

  else if(strcmp(sub_com.name, "steer") == 0)
    return get_direction(bot);
```

```
else if(strcmp(sub_com.name, "money") == 0)
  return get_money(bot);

else if(strcmp(sub_com.name, "nb_bullet") == 0)
  return get_nb_bullet(bot);

else if(strcmp(sub_com.name, "armor") == 0)
  return get_armor(bot);

else if(strcmp(sub_com.name, "coord") == 0)
  return get_coord(bot,sub_com.subcom->name);

else if(strcmp(sub_com.name, "!=") == 0)
  return eval(sub_com.subcom[0],bot,server,client,buffer,
      ↪ taille,dico) != eval(sub_com.subcom[1],bot,server,
      ↪ client,buffer,taille,dico);

else if(strcmp(sub_com.name, "==") == 0)
  return eval(sub_com.subcom[0],bot,server,client,buffer,
      ↪ taille,dico) == eval(sub_com.subcom[1],bot,server,
      ↪ client,buffer,taille,dico);

else if(strcmp(sub_com.name, ">") == 0)
  return eval(sub_com.subcom[0],bot,server,client,buffer,
      ↪ taille,dico) > eval(sub_com.subcom[1],bot,server,
      ↪ client,buffer,taille,dico);

else if(strcmp(sub_com.name, "<") == 0)
  return eval(sub_com.subcom[0],bot,server,client,buffer,
      ↪ taille,dico) < eval(sub_com.subcom[1],bot,server,
      ↪ client,buffer,taille,dico);

else if(strcmp(sub_com.name, ">=") == 0)
  return eval(sub_com.subcom[0],bot,server,client,buffer,
      ↪ taille,dico) >= eval(sub_com.subcom[1],bot,server,
      ↪ client,buffer,taille,dico);

else if(strcmp(sub_com.name, "<=") == 0)
  return eval(sub_com.subcom[0],bot,server,client,buffer,
      ↪ taille,dico) <= eval(sub_com.subcom[1],bot,server,
      ↪ client,buffer,taille,dico);

else if(strcmp(sub_com.name, "+") == 0)
```

```
    return eval(sub_com.subcom[0],bot,server,client,buffer,
        ↪ taille,dico) + eval(sub_com.subcom[1],bot,server,
        ↪ client,buffer,taille,dico);

  else if(strcmp(sub_com.name, "-") == 0)
    return eval(sub_com.subcom[0],bot,server,client,buffer,
        ↪ taille,dico) - eval(sub_com.subcom[1],bot,server,
        ↪ client,buffer,taille,dico);

  else if(strcmp(sub_com.name, "*") == 0)
    return eval(sub_com.subcom[0],bot,server,client,buffer,
        ↪ taille,dico) * eval(sub_com.subcom[1],bot,server,
        ↪ client,buffer,taille,dico);

  else if(strcmp(sub_com.name, "/") == 0)
    return eval(sub_com.subcom[0],bot,server,client,buffer,
        ↪ taille,dico) / eval(sub_com.subcom[1],bot,server,
        ↪ client,buffer,taille,dico);

  else if(strcmp(sub_com.name, "mod") == 0)
    return eval(sub_com.subcom[0],bot,server,client,buffer,
        ↪ taille,dico) % eval(sub_com.subcom[1],bot,server,
        ↪ client,buffer,taille,dico);

  else if (strcmp(sub_com.name, "=") == 0)
    affect_dico(sub_com.subcom[0].name,eval(sub_com.subcom[1],
        ↪ bot,server,client,buffer,taille,dico),dico);

  else {
    aff* elem = search_in_dico(sub_com.name,*dico);
    if (elem != NULL) {
      return elem->data;
    }
  }
  return atoi(sub_com.name);
}

/* fonction d'interpretation des commandes */
int interp(cmd sub_com, robot *bot, mqd_t server, mqd_t client
    ↪ , char* buffer, int taille, aff **dico){
  if (sub_com.nb_subcom > 0 && sub_com.nb_args == 0) {
    if (strcmp(sub_com.name, "script") == 0) {
      for (int i = 0; i < sub_com.nb_subcom+sub_com.nb_args; ++
          ↪ i) {
```

```
      if (bot->winner != 0 || bot->wait_player != 0) return
          ↪ -1;
      interp(sub_com.subcom[i],bot,server,client,buffer,
          ↪ taille,dico);
    }
  }
 }
 if (sub_com.nb_subcom == 0){
  if (bot->winner != 0) return -1;
  return eval(sub_com,bot,server,client,buffer,taille,dico);
 }else {
  if(strcmp(sub_com.name, "while") == 0){
   while (eval(sub_com.subcom[0],bot,server,client,buffer,
       ↪ taille,dico)) {
    for (int i = 1; i <= sub_com.nb_subcom; ++i) {
     if (bot->winner != 0 || bot->wait_player != 0) return
         ↪  -1;
     interp(sub_com.subcom[i],bot,server,client,buffer,
         ↪ taille,dico);
    }
   }
  }
  if(strcmp(sub_com.name, "if") == 0){
   if (eval(sub_com.subcom[0],bot,server,client,buffer,
       ↪ taille,dico)) {
    for (int i = 1; i <= sub_com.nb_subcom; ++i) {
     if (bot->winner != 0 || bot->wait_player != 0) return
         ↪  -1;
     interp(sub_com.subcom[i],bot,server,client,buffer,
         ↪ taille,dico);
    }
   }
  }
 }
 return -1;
}
```

A.6 L'interpréteur

```
#include "game.h"

/* fonction de recuperation des lignes dans un fichier */
char* get_line(FILE *fd){
  int ch,i = 0;
  char *ligne = malloc(100);

  while(ch != EOF){
    ch = fgetc(fd);
    if(ch == '\n' || ch == EOF){
      ligne[i] = '\0';
      return ligne;
    }
    ligne[i] = ch;
    ++i;
  }
  return NULL;
}


/* fonction de creation des commande */
cmd create_cmd(char **ligne, FILE *fd){
  if(ligne == NULL){
    cmd new_cmd = {"script",0,0,NULL};
    new_cmd.subcom = malloc(sizeof(cmd));
    while (feof(fd)==0) {
      new_cmd.subcom = realloc(new_cmd.subcom,(new_cmd.
          ↪ nb_subcom+1)*sizeof(cmd));
      char *line[100];
      *line = get_line(fd);
      new_cmd.subcom[new_cmd.nb_subcom] = create_cmd(line,fd);
      ++new_cmd.nb_subcom;
    }
    return new_cmd;
  }
  char *name_cmd = str_tok(ligne,"␣\n");
  cmd new_cmd = {name_cmd,0,0,NULL};
  cmd null = {NULL,0,0,NULL};
  if (name_cmd == NULL) {
    return new_cmd;
  }else if(strcmp(name_cmd,"script")==0){
    char* name_file = str_tok(ligne,"␣\n");
```

```
  FILE *fd2 = fopen(name_file, "r");
  cmd tmp = create_cmd(NULL,fd2);
  fclose(fd2);
  return(tmp);
}else if(strcmp(name_cmd,"move")==0)
  new_cmd.nb_args = 1;

else if(strcmp(name_cmd,"turn")==0)
  new_cmd.nb_args = 1;

else if(strcmp(name_cmd,"coord")==0)
  new_cmd.nb_args = 1;

else if(strcmp(name_cmd,"shoot")==0)
  new_cmd.nb_args = 1;

else if(strcmp(name_cmd,"aim")==0)
  new_cmd.nb_args = 2;

else if(strcmp(name_cmd,"seek")==0)
  new_cmd.nb_args = 2;

else if(strcmp(name_cmd,"while")==0){
  new_cmd.nb_args = 1;
  new_cmd.nb_subcom = 1;
}
else if(strcmp(name_cmd,"if")==0){
  new_cmd.nb_args = 1;
  new_cmd.nb_subcom = 1;
}
else if(strcmp(name_cmd,"=")==0)
  new_cmd.nb_args = 2;

else if(strcmp(name_cmd,"!=")==0)
  new_cmd.nb_args = 2;

else if(strcmp(name_cmd,"==")==0)
  new_cmd.nb_args = 2;

else if(strcmp(name_cmd,"<=")==0)
  new_cmd.nb_args = 2;

else if(strcmp(name_cmd,">=")==0)
  new_cmd.nb_args = 2;
```

```
else if(strcmp(name_cmd,"<")==0)
  new_cmd.nb_args = 2;

else if(strcmp(name_cmd,">")==0)
  new_cmd.nb_args = 2;

else if(strcmp(name_cmd,"+")==0)
  new_cmd.nb_args = 2;

else if(strcmp(name_cmd,"-")==0)
  new_cmd.nb_args = 2;

else if(strcmp(name_cmd,"*")==0)
  new_cmd.nb_args = 2;

else if(strcmp(name_cmd,"/")==0)
  new_cmd.nb_args = 2;

else if(strcmp(name_cmd,"mod")==0)
  new_cmd.nb_args = 2;

new_cmd.subcom = malloc(new_cmd.nb_args*sizeof(cmd));
for (int i = 0; i < new_cmd.nb_args; ++i) {
  new_cmd.subcom[i] = create_cmd(ligne,fd);
  if (new_cmd.subcom[i].name == NULL) return null;
}

if (new_cmd.nb_subcom != 0) {
  new_cmd.nb_subcom = 0;
  if (fd == NULL) return null;
  if (strcmp(*ligne, "{")==0) {
    *ligne = get_line(fd);
    while (search(*ligne,'}')) {
      new_cmd.subcom = realloc(new_cmd.subcom,(new_cmd.
          ↪ nb_args+new_cmd.nb_subcom+1)*sizeof(cmd));
      new_cmd.subcom[new_cmd.nb_args+new_cmd.nb_subcom] =
          ↪ create_cmd(ligne,fd);
      ++new_cmd.nb_subcom;
      *ligne = get_line(fd);
    }
  }
}
return new_cmd;
```

```
}
```