Projet Autonome en Programmation C

Encadrement : Michael FRANÇOIS (francois@esiea.fr)

------ TANKS BATTLE -----

Présentation générale

L'objectif de ce projet est de programmer en langage C, un jeu de bataille de tanks. Le but est d'éliminer tous les tanks ennemis tout en protégeant le petit oiseau (Titi). Le projet est à réaliser obligatoirement en binôme (ancien & nouveau) sous environnement GNU/Linux. L'affichage doit être effectué uniquement sur console.
La FIG 1 montre un exemple de jeu réalisé avec un affichage sur le terminal.

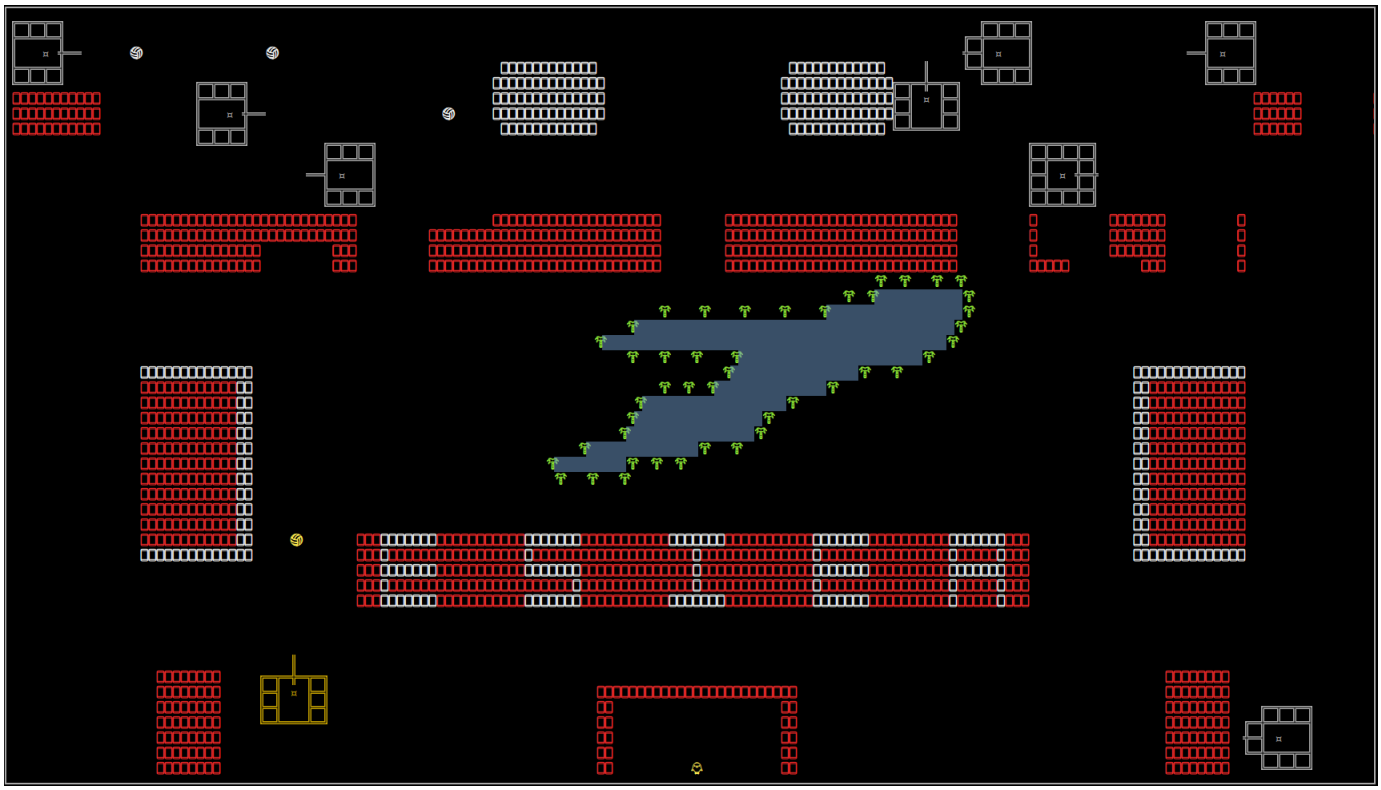
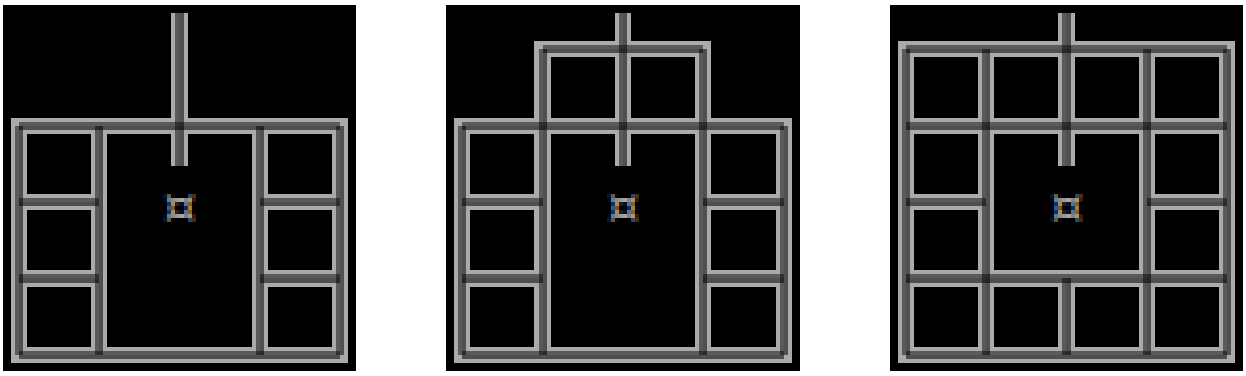


FIGURE 1 – Jeu en cours avec un affichage sur le terminal.

Sur ce plan de jeu, on peut distinguer :

- un tank jaune qui est le tank appartenant au joueur ;
- des tanks blancs qui appartiennent à l'ennemi ;
- un petit oiseau (Titi), encerclé par des briques rouges. Notre mission consiste à protéger cet oiseau des obus de l'ennemi.
- des briques rouges, qui sont légères et cassables par n'importe quel tir ;
- des briques blanches, qui sont dures et cassables uniquement que par des tirs de tanks ultra-blindés.

On peut également distinguer trois types de tanks, comme indiqué sur la FIG 2.



Tank normal | Tank blindé | Tank ultra-blindé

FIGURE 2 – Les différents types de tanks disponibles.

Vous pouvez dessiner vos propres modèles de tanks (fortement conseillé).

Ce plan de jeu se trouve à la base dans un fichier "`.txt`" y compris le décor. Pour alléger l'affichage, le plan doit être figé sur l'écran afin d'éviter de recharger ce dernier à chaque fois pour ne pas ralentir l'exécution.

Vous devrez imaginer votre propre plan avec un décor différent de celui présenté.
Vous pouvez rajouter du son pour votre jeu, afin de le rendre plus plaisant :

- musique de fond ;
- son au moment d'un tir sur brique rouge ;
- son au moment d'un tir sur brique blanche ;
- son au moment où un tank ennemi est abattu ;
- etc.

Pour cela vous pouvez utiliser la librairie `sox` et lancer les sons via la commande "play" en tâche de fond, afin de ne pas perturber l'affichage pendant le déroulement du jeu.

Au lancement du jeu, le joueur doit pouvoir choisir entre deux modes de jeu :

- **Facile :** moins de tanks seront générés (ex. 15), les tanks avancent lentement, les tirs sont moins fréquents, etc.
- **Difficile :** plus de tanks seront générés (ex. 30), les tanks avancent rapidement, les tirs sont plus fréquents et plus rapides, plus de tanks ultra-blindés, etc.

Rien ne vous empêche de rajouter d'autres modes si vous le désirez, d'ailleurs je vous le conseille fortement et ça sera du bonus pour votre note finale.

Aspects techniques

Les aspects techniques qui suivent correspondent à la réalisation précédente. Vous pouvez vous en inspirer pour votre programme.

Le plan de jeu est chargé depuis un fichier "`.txt`" et affiché classiquement sur la sortie standard. Chaque modèle de tank se trouve dans un fichier "`.txt`" à part, et doit être chargé au lancement du jeu. Un tableau bidimensionnel a été utilisé pour stocker la présence ou non de chaque objet du plan. Ceci permet de gérer plus facilement l'affichage en temps réel du plan. Voilà un exemple de structure utilisée pour rassembler toutes les informations liées à un tank :

```
-------------------------------------------------------------------------
typedef struct tank TANK;
struct tank
{
  char Direction ;          /*N => Nord, S => Sud, E => EST, O => OUEST*/
  int PosX;                 /*Position courante coin gauche X du tank*/
  int PosY;                 /*Position courante coin gauche Y du tank*/
  int Blindage;             /*niveau de blindage en cours du tank
                             (0 => rien, 1 => blindé, 2 => ultra-blindé)*/
  int Blindage_orig;        /*Blindage d'origine*/
  int Touches;              /*Nombre de fois que le tank est touché*/
  char Carrosserie [5][28];/*Carrosserie du tank, servira pour
                            l'affichage du tank à tout moment*/
  char Type;                /*'M' => mon tank, 'E' => tank ennemi*/
  int Etat;                 /*État du tank 1 => actif, 2 => en destruction,
                              3 => inactif*/
  int Mise_a_jour;          /*utile pour la suppression du tank en tenant
                              compte d'un delay*/
  struct tank * NXT;        /*Pointeur vers un prochain tank*/
  /*Vous pouvez rajouter d'autres variables si nécessaire */
};
-------------------------------------------------------------------------
```

Vous aurez sans doute besoin d'autres structures, notamment pour les obus (qui contiennent plusieurs types d'infos comme par exemple : la position, la direction ou même la provenance de l'obus). Pour faire vos choix dans le menu ou même pendant le déroulement, vous pouvez utiliser la fonction `key_pressed()` indiquée ci-dessous :

```
-------------------------------------------------------------------------
char key_pressed()
{
    struct termios oldterm, newterm;
    int oldfd; char c, result = 0;
    tcgetattr (STDIN_FILENO, &oldterm);
    newterm = oldterm; newterm.c_lflag &= ~(ICANON | ECHO);
    tcsetattr (STDIN_FILENO, TCSANOW, &newterm);
    oldfd = fcntl(STDIN_FILENO, F_GETFL, 0);
    fcntl (STDIN_FILENO, F_SETFL, oldfd | O_NONBLOCK);
    c = getchar();
    tcsetattr (STDIN_FILENO, TCSANOW, &oldterm);
    fcntl (STDIN_FILENO, F_SETFL, oldfd);
    if (c != EOF) {ungetc(c, stdin); result = getchar();}
    return result;
}
-------------------------------------------------------------------------
```

Cette fonction vous permet de récupérer la touche saisie par l'utilisateur sans pour autant retarder l'affichage car, le plan doit être dynamique. Pour cela vous aurez besoin des librairies supplémentaires suivantes :
`signal.h`, `string.h`, `termios.h`, `unistd.h` et `fcntl.h`.

Pour afficher un véhicule/obus sur l'écran, il suffit de déplacer le curseur sur le terminal à la position voulue et ainsi à l'aide d'un `printf` afficher le tableau `Carrosserie` correspondant.

Pour avoir un affichage plus joli, les caractères de l'ASCII étendu ont été utilisés. Vous pouvez les retrouver au lien suivant :
`http://www.theasciicode.com.ar/`

Vous pouvez également retrouver des émoticônes pour un meilleur rendu de votre jeu à ce lien :
`https://fr.piliapp.com/twitter-symbols/`

Il suffit de cliquer sur le symbole souhaité, le copier puis le coller simplement dans votre fichier source. Dans le cas où le symbole ne s'affiche pas correctement sur le terminal, vous devez installer le package "ttf-ancient-fonts" via la commande :
`sudo apt-get install ttf-ancient-fonts`

NB : pour ceux qui veulent avoir un graphisme plus sophistiqué, ils peuvent utiliser la SDL, mais ce n'est pas nécessaire pour obtenir la note maximale au projet.

Infos pratiques

Le projet est à réaliser obligatoirement en binôme (ancien & nouveau) sous environnement GNU/-Linux, et doit être déposé sur la plate-forme pédagogique *Moodle* au plus tard le 02/12/2018 à 23h55, sous la forme d'une archive `.zip` à vos noms et prénoms (*i.e.* `NOM_prenom.zip`), contenant tous les fichiers sources du projet et également un rapport en `pdf`. Dans le cas où l'archive dépasse 8 MO, merci de m'envoyer un lien de téléchargement par email (`francois@esiea.fr`). Vous expliquerez toutes les démarches effectuées sur chaque partie et conclure en insistant notamment sur les difficultés rencontrées et les problèmes non résolus (s'il en reste).

Votre programme doit obligatoirement présenter les différentes thématiques suivantes :

- tableaux, pointeurs et allocation dynamique ;
- structures et liste chaînée ;
- fichier *Makefile* contenant les commandes de compilation ;
- des lignes de codes commentées, lisibles et bien agencées ;
- un choix cohérent de noms de variables.

L'évaluation sera globalement scindée en 4 parties :

1. Le **design** du plan de jeu dans l'ensemble (tank, obus, décor, etc.), le **lancement** ainsi que la **fin** du jeu.
2. Test du mode **Facile**.
3. Test du mode **Difficile**.
4. Le **rapport** en `pdf` qui doit être soigneux, complet et bien détaillé.