Projet Autonome en Programmation C

Encadrement : Équipe pédagogique

------ SIMULATEUR DE PARKING -----

Présentation générale

L'objectif de ce projet est de programmer en langage $\mathbb{C}$, un jeu permettant de simuler le stationnement de voitures dans un parking payant. Le projet est à réaliser obligatoirement en binôme 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.

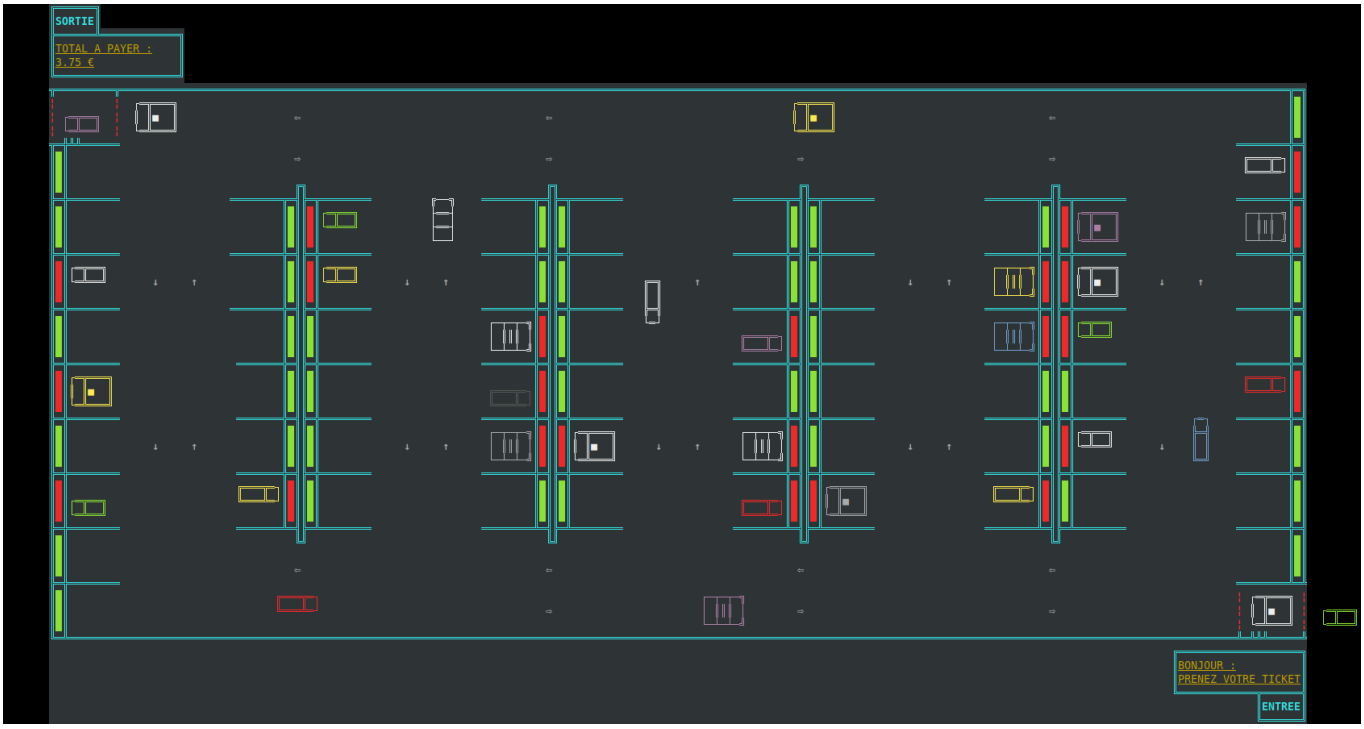

FIGURE 1 – Jeu en cours avec un affichage sur le terminal.

Sur ce plan de jeu, on peut distinguer :

- une entrée et une sortie possédant chacune deux barrières de sécurité s'ouvrant de manière alternée ;
- une borne à l'entrée du parking permettant la distribution des tickets ;
- une borne à la sortie du parking permettant aux conducteurs de payer les frais de stationnement ;
- des places de parking libres (*resp.* occupées) symbolisées par un indicateur vert (*resp.* rouge) ;
- des véhicules déjà stationnés et d'autres en mouvement soit à la recherche de places libres soit pour sortir du parking.

On peut remarquer aussi que les véhicules sont de **modèles différents** avec des couleurs variées. Vous pouvez dessiner vos propres modèles de véhicules (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 ;
- bruit lors d'une forte accélération ;
- bruit lors d'un freinage brusque ;
- 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. Par exemple, pour lancer le son "menu.mp3" suite au lancement du jeu, on utilise l'instruction :

```
system("play -q 'menu.mp3' &");
```

Pour arrêter le son lancé en tâche de fond, vous pouvez utiliser la commande `kill` de cette façon :

```
system("kill `pidof play` &");
```

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

- **Fluide :** moins de véhicules seront générés et tout roule normalement.
- **Chargé :** plus de véhicules générés pouvant entraîner des bouchons ou même des accidents dans le parking.

Rien ne vous empêche de rajouter d'autres modes si vous le désirez, d'ailleurs on vous le conseille fortement et ça sera du bonus pour votre note finale. Par exemple un mode ou vous conduirez un véhicule jusqu'au stationnement.

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 véhicule 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 et éviter surtout les collisions.

Voilà un exemple de structure utilisée pour rassembler toutes les informations liées à un véhicule :

```
-------------------------------------------------------------------------
typedef struct voiture VEHICULE;
struct voiture
{
  char direction ;  /*N => Nord, S => Sud, E => EST, O => OUEST*/
  int posx;      /*Position courante coin haut gauche x de la voiture*/
  int posy;      /*Position courante coin haut gauche y de la voiture*/
  int vitesse;      /*Vitesse du véhicule*/
  char alignement;  /*'g'=>gauche ou 'd'=>droite*/
  char type;        /*'v'=>voiture, 'c'=>camion, etc.*/
  char Carrosserie [4][30]; /*Carrosserie de la voiture, servira pour
                              l'affichage du véhicule à tout moment*/
  int code_couleur; /*Code couleur de la voiture à utiliser lors de
                      l'affichage*/
  char etat;      /*État du véhicule : '1' => actif et '0' => inactif*/
  unsigned long int tps;/*pour stocker le temps passé dans le parking*/
  struct voiture * NXT; /*Pointeur vers une prochaine voiture,
                          nécessaire pour la liste chaînée*/
  /*Vous pouvez rajouter d'autres variables si nécessaire */
};
-------------------------------------------------------------------------
```

Vous aurez sans doute besoin d'autres structures, notamment pour les listes chaînées. En effet, les véhicules doivent être utilisées **obligatoirement** dans une **liste chaînée**.

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 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 sous environnement GNU/Linux, et doit être déposé sur la plate-forme pédagogique *Moodle* (date limite à préciser), sous la forme d'une archive `.zip` à vos noms et prénoms (*i.e.* `NOM_prenom.zip`), contenant tous les fichiers sources du projet. Dans le cas où l'archive dépasse 75 Mo, merci de nous envoyer un lien de téléchargement par email (`email à préciser`), **biensur avant la date limite**.

Votre programme doit obligatoirement présenter les différentes thématiques suivantes :

- tableaux, pointeurs et allocation dynamique ;
- traitement de fichier ;
- 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 3 parties :

1. Le **design** du plan de jeu (voitures, décor, etc.), le **menu** ainsi que la **fin** du jeu.
2. Test du mode **Facile**.
3. Test du mode **Chargé**.