Projet : Shell Invaders

{pascal.vanier}@u-pec.fr

Attention ! Le sujet peut être mis à jour dans les prochaines semaines, pensez à le reconsulter de temps à autre.

Il faut lire le sujet intégralement avant de commencer le projet [1].

1 Consignes

Le projet devra être codé intégralement en C, tout ajout non demandé ne sera pris en compte dans la note que si les fonctionnalités de base sont implémentées. Le projet est à réaliser **seul ou à deux**.

Les ouvertures/lectures/écritures de fichiers devront être faites avec les appels systèmes, vous avez cependant le droit d'utiliser les fonctions de `<string.h>` ainsi que les fonctions `sprintf`, `snprintf` et `perror`. Il devra comporter un `Makefile` et il devra compiler avec les options `-std=gnu99 -Wall -Werror`.

Vous joindrez à votre projet une documentation expliquant son fonctionnement, ses limites, les problèmes que vous avez rencontrés et les solutions que vous avez trouvées et les contributions des divers membres du groupe. Celle-ci devra se trouver dans un fichier `readme.md` se trouvant à la racine de votre dépôt [2].

La notation tiendra compte de divers paramètres : la qualité du code (lisibilité, modularité, commentaires), la qualité des structures de données, la qualité algorithmique, le fonctionnement du programme (phase réussie, présence ou absence de bugs), ainsi que la soutenance.

Il est obligatoire d'utiliser le gestionnaire de versions `git`, un compte sera créé pour chacun d'entre vous sur le serveur https://git-etudiants.lacl.fr dès que les groupes auront été formés, avec ces comptes seront créés des dépôts pour chacun des groupes. Vos `Makefile` et `README.md` doivent se trouver à la source du répertoire.

Attention : Tout projet qui ne sera pas rendu sur `git` ne sera pas noté. Vous devez vous en servir *régulièrement* [3].

2 Dates

Les groupes sont à former avant le **2 mars 2017**, sur éprel. Après cette date les personnes n'étant pas dans un groupe se verront retirer plusieurs points.

La version finale du projet doit être sur `git` avant le **14 mai 2017**. La soutenance aura lieu le **16 mai 2017**.

3 Le principe

Le but de ce projet sera de faire un jeu : un clone de *space invaders* [4]

1. Oui, je sais, cela paraît évident...
2. Les fichiers `md` sont des fichiers markdown, une documentation est disponible sur https://docs.gitlab.com/ce/user/markdown.html.
3. Cela signifie en particulier que les excuses comme "j'ai perdu mes fichiers par accident", "nous avons rajouté des bugs au dernier moment" ne seront pas acceptées.
4. Il est possible d'y jouer en ligne si vous ne connaissez pas, votre moteur de recherche préféré est votre ami.

Le jeu permettra de jouer sur différents niveaux : les niveaux pourront être chargés à partir d'un fichier. Le jeu devra gérer plusieurs ensembles de niveaux que l'on appellera `mods`. La syntaxe de ces fichiers est donnée en annexe A.

Attention : le programme ne devra être constitué que d'un seul thread.

4 Phase 1 : un film dans le terminal

La première étape sera de vous familiariser avec `termios`, allez faire un tour sur la page `man termios`, afin d'être capables d'afficher une "image" ascii dans votre terminal. On souhaite également qu'à cette étape les entrées au clavier ne soient pas affichées (le terminal ne devra donc pas rester en mode canonique).

La première étape du projet sera donc d'être capables d'afficher une image dans le terminal, sans que les vaisseaux ne bougent. Vous devez donc à cette étape déjà être capables de lire les fichiers contenant les niveaux. Vous devrez pour cela créer au moins un mod par défaut et créer des niveaux dans celui-ci [5].

5 Phase 2 : faire bouger les vaisseaux

Phase 2.1 : Il faut maintenant faire bouger votre vaisseau. Il faudra pour cela surveiller l'entrée standard, et régulièrement mettre à jour l'écran dans le temps. On pourra pour cela utiliser le `timeout` de `poll` ou `select`. Pour déplacer votre vaisseau vous utiliserez les flèches.

Phase 2.2 : Il faut maintenant gérer les collisions : les vaisseaux ennemis descendent avec un certain rythme et quand un de ces vaisseaux touche le votre, il faut traiter l'évènement.

Note : Une fois cette phase terminée, vous avez un jeu d'évitement de vaisseaux.

6 Phase 3 : gestion des tirs

Phase 3.1 : Il faut maintenant que le vaisseau du joueur et les vaisseaux ennemis puissent tirer. Un vaisseau touché doit perdre une vie. **Note :** À ce stade le jeu est réellement fonctionnel.

Phase 3.2 : Il faut gérer des *powerups*, c'est à dire des objets permettant d'améliorer le vaisseau du joueur : sa vitesse maximale de déplacement et la vitesse de ses tirs.

7 Bonus

Vous pouvez maintenant ajouter les fonctionnalités que vous voulez par exemple :

— des tirs non rectilignes (par exemple trois faisceaux de tir : un tout droit, un sur la diagonale Nord-Est et un sur la diagonale Nord-Ouest).
— des powerups
— des boss avec des formes étranges et des tirs non moins étranges
— jeu en réseau

A Syntaxe des fichiers

A.1 Niveaux

Chque fichier de niveau est constitué de la manière suivante :

5. quelques niveaux suffiront

— Une ligne qui contient le nombre de vaisseaux ennemis.
— Chaque ligne contient alors, séparés par des espaces, le type de vaisseau (un numéro), le temps auquel il apparaît (en secondes) ainsi que l'endroit où il apparait (en position relative à la largeur puis hauteur du terminal, 100 représentant tout en haut et 0 tout en bas pour un jeu en scrolling horizontal).

Par exemple pour un niveau où 50 vaisseaux apparaîtront, le début du fichier pourra être formé ainsi :

```
50
1 0 75 40
1 0 50 40
1 0 25 40
2 0 75 20
2 0 50 20
2 0 25 20
...
```

Ici, à la 0ème seconde de temps de jeu sur le niveau, doivent apparaître 3 vaisseaux de type 1, tous à hauteur 40, un à largeur 25 l'autre à largeur 50 et le dernier à largeur 75. A la 0ème seconde également on a trois vaisseaux de type 2 qui apparaissent à la hauteur 20 aux même largeurs.

A.2 Vaisseaux

Chaque fichier de vaisseau est constitué de la manière suivante :
— La première ligne contient l et h séparés par un espace : la hauteur maximale et la largeur maximale du vaisseau.
— La seconde ligne contient la longueur du cycle de déplacement.
— Puis une ligne contenant le cycle de déplacement horizontal.
— Puis une ligne contenant le cycle de déplacement vertical.
— Puis une ligne contenant le nombre d'unités de vie du vaisseau.
— Une ligne contenant la fréquence de tir du vaisseau en tirs par seconde (celle-ci pourra être ignorée pour les premières phases du projet), une valeur de 0.5, signifie que le vaisseau tire toutes les 2 secondes. Une valeur nulle signifie que le vaisseau ne tire pas.
— Une ligne contient la vitesse du tir du vaisseau, en cases/seconde.
— Une ligne contient la puissance du tir du vaisseau en unités de vie.
— Une ligne contient le symbole pour le tir du vaisseau.
— Les lignes suivantes contiennent le dessin du vaisseau.

Toutes les valeurs concernant les tirs de vaisseaux ennemis pourront être ignorées dans un premier temps : cela signifie que le fichier doit contenir ces lignes mais que votre programme ne les utilise pas, et non que le fichier ne les contient pas.

Par exemple :

```
5 3
12
1 1 1 1 1 0 -1 -1 -1 -1 -1 0
0 0 0 0 0 -1 0 0 0 0 0 -1
14
0.5
2
1
.
0-0-0
\ | /
 \*/
```

On a donc ici un vaisseau qui tient dans un rectangle 5×3, dont les dimensions sont données dans la première ligne, qui se déplace 5 fois vers la droite puis vers le bas, puis 5 fois vers la droite puis vers le

bas et qui recommence. Il a 14 unités de vie et qui tire un . de puissance 1 et de vitesse 2 toutes les 2 secondes.

A.3 Mods

Un "mod" (un ensemble de niveaux) sera constitué d'un répertoire dont le nom sera le nom du mod et qui contiendra un sous répertoire. Dans le répertoire du mod figurera un fichier `deroulement` le informations sur le jeu : la première ligne devra contenir le nom du jeu et les lignes suivantes les identifiants des niveaux par ordre d'apparition. Le répertoire du mod devra contenir un sous-répertoire `niveaux` et un sous-répertoire `vaisseaux`. Chaque niveau sera stocké dans un fichier dont le nom sera un nombre **sans extension**, le fichier `deroulement` contiendra donc dans la première ligne le titre du mod, puis sur chaque ligne le numéro correspondant au niveau à charger ensuite.

Ceci doit permettre de facilement installer de nouveaux "mods" dans votre jeu. Simplement en ajoutant un répertoire correspondant à un mod, on doit pouvoir dans le menu d'entrée du jeu sélectionner ce mod si on le souhaite.

B Manipuler le terminal

B.1 Taille du terminal

Votre jeu doit fonctionner pour une taille fixée de terminal que vous pourrez choisir. Ne choisissez pas une taille trop grande, il faut pouvoir jouer sur les ordinateurs des salles informatiques. Un bon moyen de s'assurer que cela puisse fonctionner dans les salles informatiques est de lancer votre jeu avec la commande suivante :

```
xterm -geometry 80x42 -e /chemin/vers/jeu
```

vous pouvez remplacer 80 et 42 par les valeurs que vous souhaitez. Il est également possible d'accèder à la hauteur/largeur d'un terminal en accédant aux variables d'environnement `LINES` et `COLUMNS`. Il serait de bon aloi de vérifier avant de lancer une partie sur un mod si le terminal permet d'afficher ce mod.

B.2 Mode du terminal

Le mode par défaut dans lequel est le terminal est le mode canonique, c'est à dire qu'un read débloque uniquement quand l'utilisateur tape sur la touche entrée et tout ce qui est entré au clavier est affiché à l'écran. Il faut, afin que le terminal n'affiche pas les touches utilisées et permette de lire les touches sans que l'on utilise entrée, passer le terminal en mode `raw`, on peut pour cela faire appel à la fonction `tcgetattr` afin de récupérer les attributs du terminal, modifier ces attributs avec `cfmakeraw`, puis les attribuer au terminal avec `tcsetattr`. Le `man termios` ainsi que de chacune de ces fonctions est votre ami.

B.3 De la couleur dans le terminal

On peut utiliser de la couleur dans un terminal, afin de faire cela, on peut écrire avec `write` le résultat de `sprintf(buffer, "\x1b[91m%s", acolorier);`. Après cela, toutes les autres chaînes qui seront écrites seront de cette couleur, il faut donc remettre la couleur initiale si l'on ne veut pas cela.

Vous pouvez consulter la page http://jafrog.com/2013/11/23/colors-in-terminal.html pour plus d'informations sur les couleurs et autres attributs du texte and le terminal (lien en anglais).

B.4 Se déplacer dans le terminal

Vous pouvez déplacer le curseur à la position (x, y) en écrivant le résultat de

```
sprintf(buffer, "\x1b[%d;%dH", x, y);
```

avec `write`.

B.5 Cacher le curseur

Pour cacher le curseur, il suffit d'utiliser le code `\x1b[?251` et pour le réafficher `\x1b[?25h`.