Université de Strasbourg
Structure de Données et Algorithmes Feuille de TP

TP libre : Spaceship Infinity

Le but de ce TP libre est de compléter le code d'un jeu d'arcade dans lequel le joueur contrôle un vaisseau. La Figure 1 présente une capture d'écran du jeu.

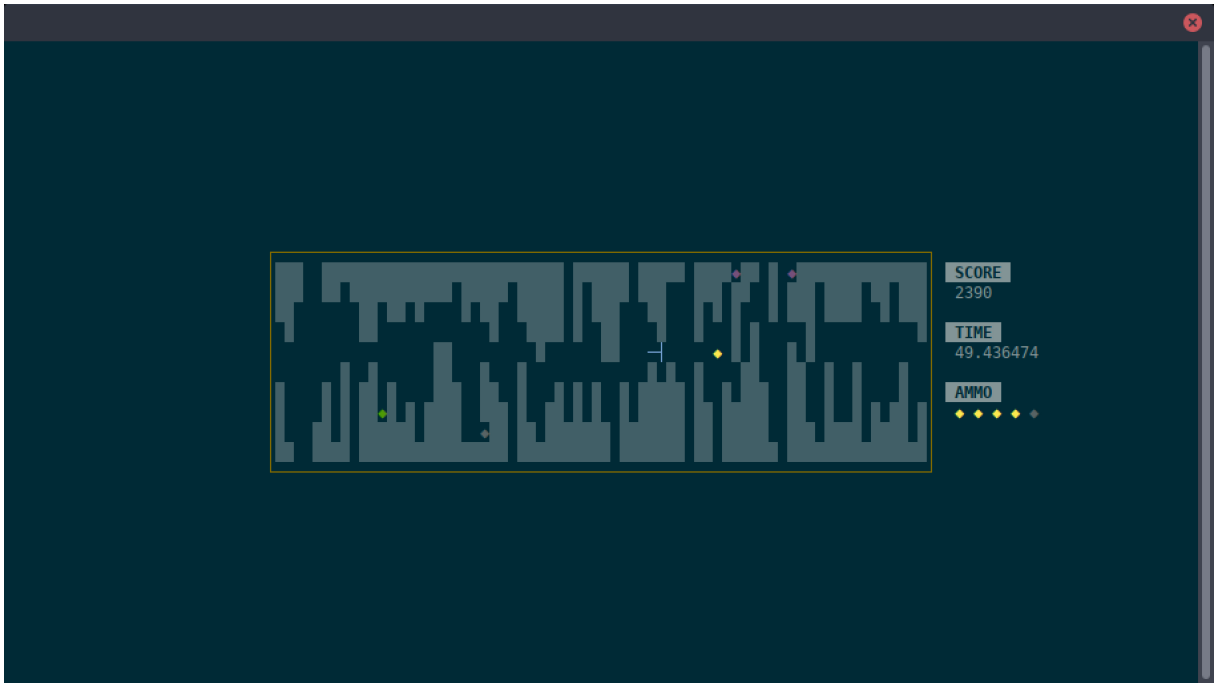

FIGURE 1 – Spaceship Infinity

Le vaisseau se déplace dans un monde en 2 dimensions et peut y récolter divers objets : de points bonus, des malus, des slots supplémentaires pour son arme ou bien des objets surprise. Le vaisseau peut tirer des projectiles pour détruire des élements sur son passage.

Vous trouverez sur Moodle une archive contenant un certain nombre de fichiers pour vous aider à démarrer. Après l'avoir téléchargée :

```
$ tar -xf spaceship-infinity.tar.gz
$ cd spaceship-infinity
```

`column`

La structure `column` représente une colonne de cellules (voir le fichier `cell.h` pour plus d'informations sur les cellules) dans le jeu. Au cours d'une partie, toutes les colonnes ont la même taille et la taille d'une colonne est fixe.

Une implémentation est déjà fournie pour les fonctions relatives à cette structure sauf pour la fonction `column_fall()`. Cette fonction fait « tomber » d'une case les cellules qui ne sont pas connectées au « plafond ». En d'autres termes, une cellule peut tomber si et seulement si les conditions suivantes sont vérifiées :

1. il ne s'agit pas de la première cellule de la colonne
2. il ne s'agit pas de la dernière cellule de la colonne
3. il existe une cellule vide entre cette cellule et la première cellule
4. il existe une cellule vide entre cette cellule et la dernière cellule

La Figure 2 présente un exemple d'utilisations répétées de cette fonction sur une colonne dans laquelle deux blocs de cellules doivent tomber.

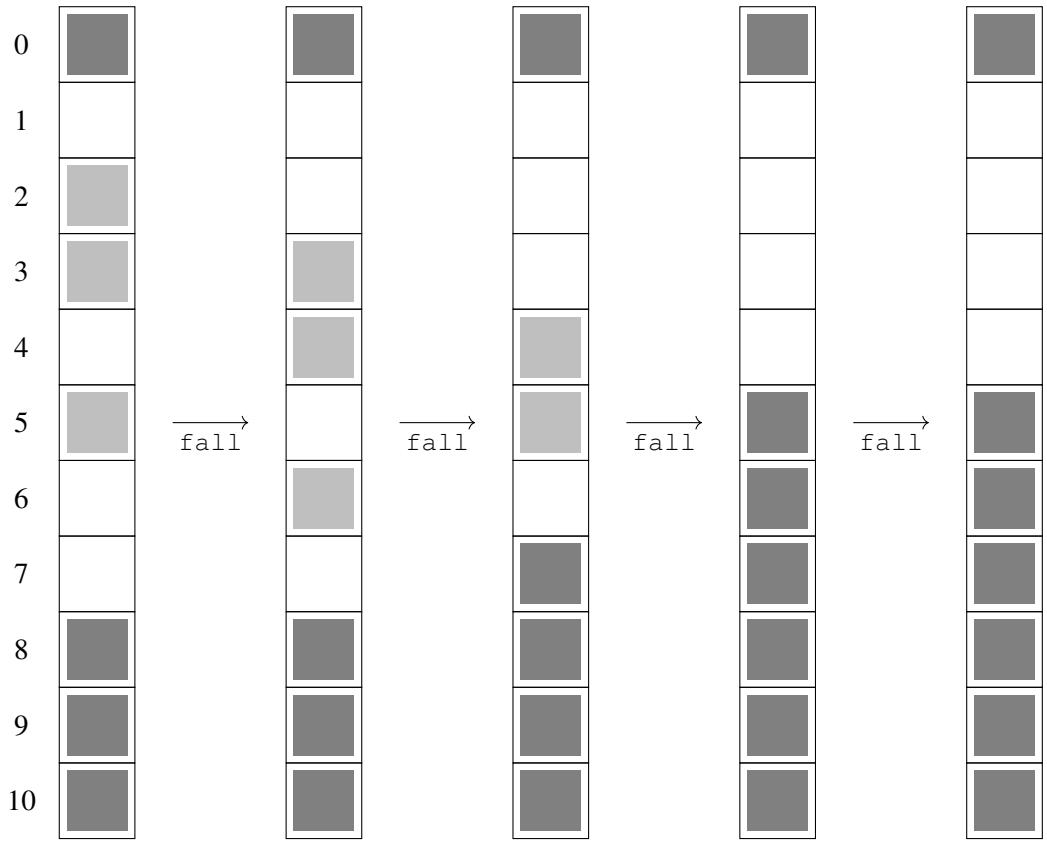

FIGURE 2 – Exemple d'appels successifs de la fonction `column_fall()` :
Dans la colonne initiale, les blocs `{2, 3}` et `5` doivent tomber. Après deux itérations, l'un des deux blocs ne peut plus tomber tandis que le second peut encore tomber. Lorsqu'aucun bloc ne peut tomber, la fonction ne modifie pas la colonne.

Il vous incombe de mettre au point et implémenter un algorithme pour cette fonction : **modifiez la fonction `column_fall()` dans le fichier `colunm.c`**.

`column_list` et `point_list`

Il vous faut implémenter deux types de listes : `column_list` (les listes de `column`) et `point_list` (les listes de `point`).

`column_list`

Le terrain (voir les fichiers `terrain.[ch]`) du jeu repose sur une liste de colonnes (`column_list`). Cette liste correspond aux colonnes affichées. Lorsque le décor bouge, il faut supprimer et ajouter de nouvelles colonnes en tête ou en queue de liste. Plusieurs mécanismes (affichage, tests de collision, chute des cellules) nécessitent de pouvoir accéder à une colonne donnée à partir de son index.

```c
#ifndef _LISTE_H_
#define _LISTE_H_

#include <stddef.h>
#include "column.h"

////////////////////////////////////////////////////////////////////////////////
// types
////////////////////////////////////////////////////////////////////////////////

typedef struct column_list column_list;

////////////////////////////////////////////////////////////////////////////////
// init./destroy etc.
////////////////////////////////////////////////////////////////////////////////

column_list* column_list_new(void);
void column_list_destroy(column_list* l);

////////////////////////////////////////////////////////////////////////////////
// getters
////////////////////////////////////////////////////////////////////////////////

column* column_list_get_column(const column_list* l, size_t i);
size_t column_list_get_size(const column_list* l);

////////////////////////////////////////////////////////////////////////////////
// setters / modifiers
////////////////////////////////////////////////////////////////////////////////

column_list* column_list_push_front(column_list* l, column* c);
column_list* column_list_push_back(column_list* l, column* c);
column_list* column_list_pop_front(column_list* l);
column_list* column_list_pop_back(column_list* l);

#endif
```

`point_list`

Au cours d'une partie, le joueur peut décider de tirer. Pour stocker la position des projectiles en vol,

une liste de points (voir le fichier `point.h`) est nécessaire. À chaque « tour », tous les projectiles se déplacent vers la droite. De même, le niveau recule lorsque le joueur recule trop (en difficulté 0) et il faut décaler les projectiles en conséquence. Le code fourni suppose que ces opérations seront réalisées dans les fonctions `point_list_shift_left()` et `point_list_shift_right()`.

Par ailleurs, il arrive que les positions de certains projectiles ne correspondent plus :

- lorsqu'un projectile explose, le jeu remplace sa position par un point « invalide » (voir le fichier `point.h`).
- lorsqu'un projectile sort des limites du jeu.

Pour que le jeu puisse gérer ces cas, il faut implémenter la fonction `point_list_prune_out_of_bounds()` pour qu'elle supprime d'une liste toutes les positions qui correspondent à des points en dehors des limites du jeu. Cette fonction prend en entrée la liste à modifier, le coin supérieur gauche et le coin inférieur droit du jeu.

```c
#ifndef _POINT_LIST_H_
#define _POINT_LIST_H_

#include <stddef.h>
#include <stdbool.h>

#include "point.h"

////////////////////////////////////////////////////////////////////////////////
// types
////////////////////////////////////////////////////////////////////////////////

typedef struct point_list point_list;

////////////////////////////////////////////////////////////////////////////////
// init./destroy etc.
////////////////////////////////////////////////////////////////////////////////

point_list* point_list_new(void);
void point_list_destroy(point_list* l);

////////////////////////////////////////////////////////////////////////////////
// getters
////////////////////////////////////////////////////////////////////////////////

size_t point_list_get_size(const point_list* l);
point point_list_get_point(const point_list* l, size_t i);
bool point_list_contains(const point_list* l, point p);

////////////////////////////////////////////////////////////////////////////////
// setters / modifiers
////////////////////////////////////////////////////////////////////////////////

void point_list_set_point(point_list* l, size_t i, point p);

point_list* point_list_push_front(point_list* l, point p);
point_list* point_list_push_back(point_list* l, point p);
```

```
38 point_list* point_list_pop_front(point_list* l);
39 point_list* point_list_pop_back(point_list* l);
40
41 void point_list_shift_left(point_list* l);
42 void point_list_shift_right(point_list* l);
43 point_list* point_list_prune_out_of_bounds(
44     point_list* l, point up_left, point bottom_right);
45
46 #endif
```

Implémentation des listes

Pour chacune de ces listes, observez :

- la nature et le prototype des fonctions requises (voir les headers `column_list.h` et `point_list.h`).
- comment les listes sont utilisées dans le reste du code.

En fonction de vos observations, implémentez chacune de ces listes :

- ajoutez les champs que vous jugez utiles dans les structures `struct column_list` et `struct point_list`.
- déclarez/définissez des structures auxiliaires si vous estimez que vous en avez besoin.
- implémentez les fonctions associées.

Vous devrez motiver vos choix pour chacune des implémentations de listes.

Fonctionnalités supplémentaires

Les plus téméraires peuvent ajouter des fonctionnalités au jeu, lorsque celui-ci fonctionne. Il sera nécessaire d'étudier plus en détail le code fourni afin d'être en mesure de modifier (par exemple) les mécanismes du jeu, la gestion des entrées claviers (ajout de nouvelles touches) ou l'affichage.

Toute idée est la bienvenue à condition de détailler et expliquer la nature de vos propositions. Voici quelques exemples :

- Ajouter de nouvelles options en ligne de commande ou à la compilation pour modifier les constantes internes du jeu.
- Ajouter des bombes ou des missiles :
  Modifier le jeu pour pouvoir tirer des missiles (déplacement horizontal) ou des bombes (déplacement vertical) qui détruisent plusieurs cellules à l'impact. En s'inspirant des projectiles déjà présents, il faudra ajouter une ou plusieurs listes de points, ajouter le support d'une nouvelle touche dans la gestion des entrées clavier et modifier l'interface pour afficher les bombes ou missiles.
- Ajouter un laser :
  Ajouter un laser (à durée limitée) qui détruit toutes les cellules sur la ligne. Le laser peut aussi avoir une puissance plus ou moins élevée et ne détruire qu'un certain nombre de cellules.
- Accélérer ou freiner :
  Ajouter la possiblité de temporairement accélérer ou freiner.
- Jauge de vie :
  Ajouter une jauge de vie et faire en sorte que la jauge de vie diminue à chaque collision avec un mur. La quantité de vie perdue pourrait dépendre de la vitesse du vaisseau.

- Ajouter de nouvelles cellules spéciales :
  — bonus d'immunité temporaire
  — vie / continue
  — mines explosives
  — débloquer des armes (projectiles de base, bombes, missiles, laser)
  — *upgrade* de la puissance des bombes, des missiles ou du laser
  — ...
- *Friendly fire* :
  Les projectiles tirés par le joueur causent des dégats au vaisseau s'ils se trouvent à la même position.
- Queue vulnérable :
  Dans la version fournie, le *sprite* du vaisseau recouvre deux cases alors que la *hit box* ne se trouve que sur la tête.

Évaluation

Vous devez rendre votre travail sous la forme d'une **archive `tar`** et l'accompagner d'un **rapport au format `pdf`**. Pour vous faciliter la création de cette archive, le fichier `Makefile` fourni avec le code départ contient un cible `archive` :

```
$ make archive NAME='nom_prenom.spaceship-infinity'
```

Le rapport doit contenir une introduction et une conclusion. Il doit également détailler :

- l'algorithme choisi pour la fonction `column_fall`.
- l'implémentation choisie pour `column_list`.
- l'implémentation choisie pour `point_list`.
- les difficultés rencontrées.
- les fonctionnalités supplémentaires :
  — effets des fonctionnalités / comment les utiliser ou les activer.
  — modifications apportées au code (dans les grandes lignes).
  — algorithmes utilisés.

L'évaluation de votre travail comporte 3 sous-parties :

- l'évaluation de vos modifications des fichiers `column.c`, `column_list.c` et `point_list.c`.
- l'évaluation du rapport.
- l'évaluation des fonctionnalités supplémentaires, s'il y en a.

La compilation de votre code final ne doit produire aucun *warning* avec les options présentes dans le `Makefile` d'origine [1]. Vous êtes libres de modifier la totalité du code fourni et d'ajouter de nouveaux fichiers ou fonctions. Néanmoins quelques restrictions s'appliquent :

- les prototypes des fonctions existantes ne doivent pas être changés.
- les modifications que vous apportez aux fichiers `column.c`, `column_list.c` et `point_list.c` doivent être suffisantes pour faire fonctionner le jeu de base. En d'autres termes, **le jeu doit pouvoir fonctionner si on prend le code d'origine et que l'on y remplace ces trois fichiers (et uniquement ceux-ci) par les votres.**

1. Le correcteur ne sera pas dupe si vous vous contentez de retirer les options de warnings de votre `Makefile`...

Annexe

ncurses

Pour pouvoir compiler le programme, vous aurez besoin de la bibliothèque `ncurses`. Si celle-ci n'est pas encore installée sur votre système (choisissez la ligne appropriée pour votre système d'exploitation, si applicable) :

```
$ apt install libncurses5-dev libncursesw5-dev ncurses-doc
$ dnf install ncurses-devel
$ pacman install ncurses
```

L'utilisation de `ncurses` peut causer l'apparition de blocs *"still reachable"* dans les rapports de `valgrind`. Si la **totalité** de ces blocs vient de `ncurses`, ce n'est pas grave... En revanche, les blocs *"definitely lost"*, *"indirectly lost"* ou *"possibly lost"* sont toujours à éliminer.

Options

Un certain nombre d'options sont disponibles, passez `'-h'` ou `'-help'` en argument de la ligne de commande pour en connaître la liste complète :

```
$ ./spaceship --help
```

On peut par exemple modifier les dimensions du jeu :

```
$ ./spaceship --width=70 --height=20
```

Ou bien activer le mode *debug* :

```
$ ./spaceship --debug
```

Certaines valeurs par défaut peuvent être modifiées à la compilation :

```
$ make clean
$ make CPPFLAGS='-DDEFAULT_HEIGHT=20 -DDEFAULT_WIDTH=70'
```

Score

Le score est calculé en fonction du temps écoulé et des bonus/malus amassés au cours de la partie. Attention : tirer des projectiles fait perdre des points ! De plus, le nombre de points perdus par tir dépend du nombre de projectiles en vol. Autrement dit, attendre qu'un projectile ait détruit sa cible avant d'en tirer un autre fera perdre moins de points que de tirer deux projectiles à la suite.

Astuce

Si vous souhaitez rapidement obtenir un jeu qui fonctionne partiellement, il ne faudra pas implémenter les structures/fonctions dans l'ordre dans lequel elles vous sont présentées dans ce document ! [2] Commencez par `column_list` puis `point_list` avant de vous attaquer à `column_fall`.

2. D'où l'intérêt de lire le sujet en entier.

Afficher du texte avec `ncurses`

Vous aurez peut-être besoin de modifier l'affichage. Vous devriez vous en sortir en observant et en vous inspirant du code existant. L'interface est séparée en plusieurs « fenêtres » (`WINDOW*`). On se repère au sein d'une fenêtre à l'aide de coordonées dans le plan.

Voici la démarche générale pour afficher du texte avec `ncurses` :

1. Se positionner dans une fenêtre avec `wmove()`.
2. Activer ou désactiver des attributs (couleurs, gras, clignotement, etc...) avec `wattron()` ou `wattroff()`
3. Afficher du texte avec `waddch()`, `waddstr()` ou `wprintw()`. (voir aussi les variantes `mvwaddch()`, `mvwaddstr()` et `mvwprintw()`)
4. Activer ou désactiver des attributs avec `wattron()` ou `wattroff()`

Si vous ajoutez des informations, pensez à trouver dans le code l'endroit où la fenêtre où vous écrivez est créée et assurez vous que ses dimensions sont assez grandes.

Vous pouvez utiliser les macros suivantes pour afficher des caractères spéciaux à l'aide de `waddch()` :

- `ACS_ULCORNER`
- `ACS_LLCORNER`
- `ACS_URCORNER`
- `ACS_LRCORNER`
- `ACS_LTEE`
- `ACS_RTEE`
- `ACS_BTEE`
- `ACS_TTEE`
- `ACS_HLINE`
- `ACS_VLINE`
- `ACS_PLUS`
- `ACS_S1`
- `ACS_S3`
- `ACS_S7`
- `ACS_S9`
- `ACS_DIAMOND`
- `ACS_CKBOARD`
- `ACS_DEGREE`
- `ACS_PLMINUS`
- `ACS_BULLET`
- `ACS_LARROW`
- `ACS_RARROW`
- `ACS_DARROW`
- `ACS_UARROW`
- `ACS_BOARD`
- `ACS_LANTERN`
- `ACS_BLOCK`
- `ACS_LEQUAL`
- `ACS_GEQUAL`
- `ACS_PI`
- `ACS_NEQUAL`
- `ACS_STERLING`

Par exemple :

```
1  /* Afficher ACS_DIAMOND à la case (x = 2, y = 1). */
2  wmove(window, 1, 2);
3  waddch(window, ACS_DIAMOND);
```

N'oubliez pas l'adage *RTFM*[3] :

```
$ man ncurses
$ man wmove
$ man waddch
$ man wattron
```

3. Read The Fucking Manual