Projet Honshu - Lot A

Thomas KOWALSKI, Thibaut MILHAUD,
Matthis PILLONEL & Yoan ROCK

25 mars 2018

Table des matières

1 Comment compiler

Compilation de la documentation

Pour obtenir la documentation **Doxygen** :

```
make Doxygen
```

Puis ouvrir `html/index.html`.

Compilation des tests unitaires

Pour compiler et exécuter les tests unitaires :

```
make u_a
./unit_A
```

Compilation des tests unitaires et vérification des fuites mémoire avec `valgrind`

Pour obtenir un rapport complet (compilation des tests unitaires et exécution de `valgrind`) :

```
make vg_a
```

2 Description des types

Enumérations

Pour représenter les différents terrains possibles nous avons choisi, de définir le type `enum kase` (avec un k pour éviter de rentrer en collision avec le mot clé `case` du langage C) de la manière suivante :

```
enum kase {
    P = 'P', /* Plaine */
    F = 'F', /* Forêt */
    L = 'L', /* Lac */
    V = 'V', /* Village */
    R = 'R', /* Ressource */
    U = 'U', /* Usine */
    N = ' '  /* Vide */
};
```

La correspondance des types de cases avec les entiers correspondants aux caractères plutôt qu'avec les sept premiers entiers est utile lors de l'affichage de la grille de jeu.

De manière semblable, le type `enum card` qui contient les différentes orientations pouvant être prises par une tuile est defini ainsi :

```
enum card{
    O_N,    /* Nord */
    O_O,    /* Ouest */
    O_S,    /* Sud */
    O_E     /* Est */
};
```

Tuiles

Pour représenter les tuiles, nous avons choisi une structure contenant un tableau de `kase` de taille 6, l'orientation de la tuile, ainsi que l'identifiant de la tuile fourni dans les fichiers. Enfin, un champ `fromFile` permet de savoir si la tuile a été générée depuis un fichier ou depuis le programme (utile lors des libérations de mémoire).

```
typedef struct {
    /* Les six cases de la tuile */
    kase cases[6];
```

```
    /* L'orientation de la tuile (utilise une énumération définie plus haut.) */
    card orient;

    /*L'identifiant fourni dans les fichiers importables */
    int identifiant;

    /* (booléen) Si la tuile a été importée d'un fichier */
    int fromFile;
} tuile;
```

Cependant afin de ne pas copier des tuiles pour chaques fonction et de faire les opérations en place, nous avons décidé d'utiliser des pointeurs et donc défini le type `Tuile` :

```
typedef tuile* Tuile;
```

Grille

`Grille` permet de représenter un tableau de jeu Honshu. Tout d'abord `grille` doit contenir l'état actuel du plateau de jeu c'est-à-dire l'ensemble des cases du plateau et ce qu'elles contiennent.

Mais le sujet nous impose d'être capable de déterminer à quelle tuile appartient telle ou telle case, ce qui nous a obligé à ajouter une information dans chaque case de la grille et donc définir un nouveau type :

```
/* Case avec numéro */
struct kase_n{
    /* Le terrain de cette case */
    kase c;

    /* L'identifiant de la tuile à laquelle appartient cette case. */
    int no_tuile;
};
```

Cependant cela ne suffisait pas pour être capable de retirer une tuile. Nous avons donc choisi de stocker l'ensemble des tuiles posées dans un tableau 13 éléments faisant partie de la structure `grille`. Voilà comment nous avons procédé :

```
struct tuile_posee{
    /* La tuile (c'est à dire liste de six éléments, etc.) à poser */
    Tuile t;
```

```
    /* Les coordonnées de la case en haut à gauche de la Tuile sur la Grille */
    int i, j;
};

typedef struct tuile_posee tuile_posee;

typedef tuile_posee* Tuile_p;

struct grille {
    /* Le plateau de jeu, un tableau de \b enum \b case */
    kase_n* plateau;

    /* Le nombre de tuiles posées sur le plateau */
    int nb_tuiles;

    /* Le côté du plateau de jeu */
    int dim;

    /* Toutes les tuiles qui ont été posées depuis le début du jeu */
    Tuile_p tuiles[13];
};

typedef struct grille grille;

typedef grille* Grille;
```

Nous avons là encore décidé d'utiliser des pointeurs pour les même raisons que celles évoquées plus haut.

3 Organisation du travail

Outils de gestion de projet

Nous avons eu deux départs à notre projet : Thomas a dans un premier temps créé un **Trello** et un dépôt **git** sur **git.iiens.net**.

Cependant comme les autres membres du groupes n'arrivaient pas à utiliser ces outils à cause d'un problème de droits et d'un manque d'expérience, Thibaut a créé un nouveau **Trello** et un nouveau dépôt **git**, sur lesquels nous travaillons depuis.

Dans le but de mieux communiquer nous avons également créé une **conversation Facebook** ainsi qu'une **conversation Discord**. Ceux-ci nous permettent de discuter en temps réel du projet tout en gardant une trace des discussions.

Partie

Enfin, pour pouvoir charger des parties depuis des fichiers, nous avons implémenté une dernière structure : `partie`.

Celle-ci contient les informations importantes au début d'une partie, à savoir :

- La taille de la grille
- Le nombre de tuiles dont dispose le joueurs
- Les identifiants desdites tuiles (ces identifiants correspondent aux identifiants des tuiles chargées depuis un autre fichier)
- L'identifiant de la tuile posée au départ sur le plateau
- Un pointeur vers la tuile posée au départ sur le plateau
- Un tableau de `Tuile` représentant la main du joueur, dans l'ordre donné dans le fichier

Nous la définissons comme ceci :

```
struct partie {
    /* La taille de la grille de la partie */
    int tailleGrille;

    /* Le nombre de tuiles dans la main du joueur */
    int nbTuiles;

    /* Les identifiants de ces tuiles (pour les retrouver depuis un fichier de t
    int* identifiantsTuiles;

    /* L'identifiant de la tuile posée au début sur le plateau */
    int idTuileDepart;
```

```
    /* La tuile de départ sur la grille */
    Tuile tuileDepart;

    /* Les tuiles que le joueur a dans sa main. */
    Tuile* mainJoueur;
};
```

Répartition des tâches

Thomas et Thibaut ont commencé par implémenter les types choisis de leurs côtés (structures et énumérations). Suite à des problèmes avec git, nous avons décidé de démarrer un nouveau dépôt commun où ces éléments de base seraient faits avec plusieurs membres de l'équipe (un lundi soir), en rassemblant les travaux précédents.

Des fonctions permettant d'agir sur les types évoqués ont été écrites à ce moment-là.

Les fonctions restantes ont été réparties entre les quatre membres.

Cependant cette répartition s'est faite avant la réception du mail indiquant des changements dans les lots. Ainsi une partie du travail réalisé ne fait pas partie du lot A.

La documentation a été écrite simultanément avec le code pour être la plus précise possible et permettre au membres du groupes n'ayant pas travaillé sur une fonction d'avoir accès à sa documentation immédiatement (afin de pouvoir l'utiliser dans leur code).

Enfin, pour l'écriture du rapport nous avons utilisé **Overleaf** et nous avons travaillé simultanément sur le document en complétant les parties manquantes et en effectuant des relectures pour corriger les erreurs.

Ce n'est que lors de la séance de projets que quelques tâches (qui restaient à faire) ont eu une personne attitrée :

- Thomas : chargement de parties, chargement de tuiles, exportation de parties, exportation de tuiles, détermination du village associé à une case, prise en main et mise en place de **CUnit**, prise en main et mise en place de **ValGrind** ;
- Yoan : derniers tests de recouvrement, implémentation de la fonction permettant de poser une tuile de manière sûre ;
- Thibaud et Matthis : fonction de retrait d'une tuile du plateau.

4 Problèmes rencontrées et améliorations envisagées

— La mise en place des outils de travail de groupe a été un peu compliquée, notamment à cause d'un manque d'expérience. En effet, les premières fusions dans Git ont demandé beaucoup de temps, mais à force d'échecs, nous finissons par progresser.
— L'organisation était chaotique au début, et certaines fonctions ont été réalisées plusieurs fois tandis que d'autres n'étaient pas encore implémentées. Une plus grande maîtrise de **Trello** et **Git** nous a permis de nous améliorer sur ce point.
— Il y a eu un manque de clarté sur la répartition des tâches, ce qui a provoqué un déséquilibre au niveau du travail effectué. Lors rôles seront donc fixés plus clairement pour les prochains lots, et ce le plus tôt possible et d'une manière qui convient à l'ensemble du groupe.
— Une dernière amélioration consiste à se conceter dès le debut du lot B et d'échanger sur les futures manières de programmer pour permettre un travail d'equipe optimal où tout le monde aura eu son mot à dire et aura des tâches **précises** à réaliser.