Projet Honshu - Lot B

Thomas KOWALSKI, Thibaut MILHAUD,
Matthis PILLONEL, Yoan ROCK

29 avril 2018

Table des matières

1 Introduction

Après avoir implémenté les fonctions de base pour gérer le chargement de tuiles, de parties, les types de base du jeu, nous sommes passé au lot B, à savoir une première interface (en ligne de commande) permettant à l'utilisateur de jouer au jeu.

Afin de rendre l'interface aussi pratique que possible, nous avons décidé de traiter directement les entrées du joueur (par une fonction du type `getch`) plutôt que par un système de questions réponses.

Nous avons donc séparé le travail en trois parties : gestion de la partie et des entrées utilisateur, calcul de données d'intérêt et affichage de l'interface.

2 Choix de l'interface utilisateur

2.1 Détermination des infos à afficher

La première partie du travail à consister à déterminer quelles informations nous allions afficher. Nous sommes rapidement arrivé à la conclusion que seuls les éléments suivants étaient nécessaires au joueur pour qu'il puisse effectuer une partie sans manquer d'informations :

1. l'état actuel du plateau avec la tuile active (tuile que le joueur est en train de poser) mise en valeur ;
2. La main du joueur, soit toutes les tuiles qu'il lui reste à poser, avec de nouveau un signe distinctif pour la tuile active ;
3. Le score actuel ;
4. La liste des commandes utilisables par le joueur (*cf* paragraphe suivant).

2.2 Détermination des commandes

L'étape suivante était de lister les contrôles nécessaires au joueur. Nous avons résumé les entrées possibles à celles-ci :

— `a, z` : Choisir de la tuile ;
— `r` : Faire tourner la tuile de $\frac{\pi}{2}$ dans le sens trigonométrique ;
— `haut, bas , gauche, droite` : Déplacer le curseur (coin haut gauche de la tuile sélectionnée) sur la grille ;
— `p` : Placer la tuile active à l'endroit choisi si cet endroit est légal.

3 Affichage dans le terminal

Prérequis techniques

Pour afficher des informations dans le terminal nous ne disposions que de la fonction `printf` qui est certes efficace mais qui, sans options du moins, rend l'affichage de données plus complexe de phrases très compliqué. De plus cela signifiait qu'il aurait fallu, ré-afficher la totalité de notre canevas à chaque fois que le joueur effectuait une action.

Heureusement après cinq bonnes minutes de recherche, nous avons mis la main sur des macros qui (grâce à envois spéciaux au terminal) permettent de repositionner le curseur à l'endroit désiré dans la fenêtre ou d'effacer l'ensemble des caractères de la fenêtre.

Il devenait alors bien plus simple de fractionner notre canevas en sous-parties et de les gérer indépendamment les unes des autres après chaque instruction du joueur. Voici les macros que nous avons utilisées :

```
/* nettoie le terminal */
#define clear_term() printf("\033[H\033[J")

/* positionne le curseur à la ligne x et la colonne y */
#define gotoxy(x, y) printf("\033[\%d;\%dH", (x), (y))
```

Pour augmenter la lisibilité du jeu, nous avons également trouvé des envois qui permettent de changer la couleur de la police du terminal.

```
#define KHIG  "\033[1m"
#define KNRM  "\x1B[0m"
#define KRED  "\x1B[31m"
#define KGRN  "\x1B[32m"
#define KYEL  "\x1B[33m"
#define KBLU  "\x1B[34m"
#define KMAG  "\x1B[35m"
#define KCYN  "\x1B[36m"
#define KWHT  "\x1B[37m"
```

3.1 Délimitation du canevas

Cahier des charges La première chose à faire à alors été de délimiter la zone de jeu ainsi que les zones contenant les différentes informations. Nous sommes arrivés à la conclusion que partitionner le canevas en trois zones suffirait :

1. la grille de jeu ;
2. la main du joueur ;
3. le score et les commandes.

Solution proposée Pour délimiter les différentes parties nous nous sommes contentés d'utiliser les caractères `|, - et +` pour tracer des lignes entre chaque zone.

Illustration :

```
Score           | Grille de jeu
                |
                |
Commandes       |
                |
----------------+---------------------
Main du joueur
```

Interface Ainsi, notre fonction `dessiner_cadre(largeur, hauteur, largeurMenu, hauteurTuiles);` prend pour seul paramètre la taille `n` de la grille et affiche dans le terminal les délimitations adaptées.

3.2 Grille de jeu

Cahier des charges La grille de jeu est l'élément central du jeu et doit donc être mise en valeur par l'affichage. De plus ne regardant cette grille le joueur doit être capable de repérer les informations suivantes :

— État actuel de la grille ;
— Position du curseur et position de la tuile active sur la grille ;
— Légalité du positionnement de la grille au niveau du curseur.

Solution proposée Expliciter l'état actuel de la grille ne représente pas une grande difficulté, il suffit de parcourir la grille et d'afficher sur chaque case la lettre correspondant au type de terrain. Nous avons néanmoins rajouté des couleurs pour augmenter la lisibilité de l'ensemble (lacs en bleu, forêts en vert, etc.).

Enfin pour afficher la tuile active et son positionnement, il faut modifier légèrement la fonction décrite précédemment en ajoutant une condition qui teste si oui ou non on se trouve sur une case recouverte par la tuile active

et à ce moment là afficher le terrain le plus en surface. Enfin pour qu'il n'y ait pas d'ambiguïté par rapport à la légalité du positionnement d'une tuile sur une case, nous avons établit le code couleur suivant. La tuile active est colorée uniformément, en cyan si la pose est légale et en rouge sinon.

Interface La fonction `affiche_plateau_2_jeu(Tuile t, Grille g, int x, int y, int i, int j)` prend en paramètres `g` une `Grille`, `t`, une `Tuile`, `x, y` la positon du terminal à laquelle il faut placer le coin haut gauche de `g` et `i, j` les coordonnées du coin haut gauche de `t` dans `g`. Cette fonction affiche `g` avec `t` mise en valeur et positionnée à la position demandée comme décrit dans le paragraphe précédent.

3.3 Main du joueur

Cahier des charges L'affichage de la main doit permettre au joueur de savoir quelles tuiles sont à sa disposition actuellement et quelle et la tuile active.

Solution proposée Il se trouve que nous avons accès aux différentes tuiles disponibles via un tableau intégré à la structure `gameState` décrite dans la partie suivante. Il ne reste donc plus qu'à afficher les tuiles restantes côte-à-côte dans la zone prévue à cet effet (*cf* illustration 3.1).

La seule subtilité est de bien penser à différencier la tuile active des autres (en la colorant en cyan dans notre cas).

Interface La fonction `affiche_tuile_dispo(int hauteur, Tuile* tuiles, int indiceTuileSelectionnee)` prend donc en paramètres `t` un tableau de tuiles, `x` le numéro de la ligne à laquelle elle doit commencer à afficher ainsi que `i` l'indice de la tuile sélectionnée. Elle affiche la main du joueur en valorisant la tuile active comme décrit ci-dessus.

3.4 Score et commandes

Pour afficher le score et les commandes il suffit de positionner le curseur au bon endroit et d'avoir une fonction capable de calculer le score (*cf* prochaine partie).

4 Tests de recouvrement

4.1 Objectif

Il s'agit d'un ensemble de fonctions qui répondent toute par un booléen à la question suivante "Peut-on placer cette tuile à cet endroit" par rapport à UNE règle.

La liste des règles à respecter lorsque l'on pose un tuile est la suivante :

— La totalité de la tuile doit être incluse dans le grille ;
— Aucune case de lac ne doit être recouverte ;
— La tuile que l'on pose doit recouvrir au moins une case non vide ;
— La tuile que l'on pose ne doit pas finir de recouvrir l'une des tuiles précédentes.

4.2 Implémentation

Inclusion dans la grille Pour vérifier qu'une tuile, que l'on souhaite poser en i, j (coin haut gauche), est incluse dans une grille de taille n il suffit de faire deux tests qui dépendent néanmoins de l'orientation de la tuile :

— Tuile verticale : $i + 2 < n$ et $j + 1 < n$;
— Tuile horizontale : $i + 1 < n$ et $j + 2 < n$.

Ce test doit être appelé en premier pour éviter de faire par la suite appel à des cases de tableau non allouées.

Non recouvrement des lacs Il suffit ici de parcourir les cases de la grille qui vont être recouvertes et de renvoyer faux (0) si on rencontre une case de lac et vrai (1) si on termine la boucle.

Connexité des cases non vides Il suffit là encore de parcourir les cases de la grilles qui vont être recouvertes et de renvoyer vrai (1) si on croise une case non vide et faux (0) si on termine la boucle.

Non recouvrement total des tuiles Ce test est légèrement plus complexe que les autres. Pour le réaliser nous avons choisi de comparer le nombre de tuiles différentes :

1. Sur toute la grille ;
2. Sur la grille privée des cases recouvertes.

Il faut ensuite renvoyer le booléen (1.) = (2.).

5 Gestion d'une partie

5.1 La structure `gameState`

Pour représenter une état de jeu, la structure `gameState` a été implémentée pour contenir toutes les informations nécessaires au déroulement d'une partie de Honshu.

Cette structure contient toutes les informations de la partie en cours (et des prochains mouvements du joueur) : la grille représentée, les tuiles restantes, la tuile sélectionnée actuellement, où le joueur souhaite la poser, le plus grand village du plateau, etc.

L'interface du jeu fonctionne assez simplement : à chaque fois que le joueur effectue une action en appuyant sur une touche (connue) de son clavier, on effectue les modifications sur l'état du jeu (dans la boucle de jeu) puis on appelle les fonctions de dessin des bouts de l'affichage qui ont été affectés par son opération.

Par exemple, s'il change l'endroit où il compte poser sa tuile, on n'appelle que la fonction de dessin de la grille, puisque les autres composantes de l'interface (comme les tuiles restantes, en bas de la fenêtre) ne sont pas impactées par son changement.

```
struct gameState {
    Grille g;   La grille de jeu
    Tuile* tuiles;  Les tuiles disponibles pour être posées par le joueur.

    int tuilesDisponibles;   La taille de \b tuiles

    int iNouvelleTuile;  La ligne où le joueur aimerait poser sa tuile.
    int jNouvelleTuile;   La colonne où le joueur aimerait poser sa tuile.

    Tuile tuileSelectionnee;   La tuile sélectionnée par le joueur.
    int indiceTuileSelectionnee;

    int largeur;  La largeur du terminal
    int hauteur;   La hauteur du terminal
    int largeurMenu;  La largeur du menu de gauche
    int hauteurTuiles;  La hauteur de la liste de tuiles

    int tailleVillageMax; La taille du plus grand village de la grille
    int* villageMax; Le plus grand village de la grille
```

```
    int score;   Le score actuel de la grille
};
```

5.2 Description de la boucle de jeu

La boucle de jeu se compose d'un `while (1)` qui continue tant qu'il reste des tuiles à placer dans la main du joueur. Chaque itération est passée lors de l'appui d'une touche :

- Soit cette touche est reconnue, auquel cas on applique le traitement nécessaire et on rafraîchit l'affichage ;
- Soit cette touche n'est pas reconnue dans les contrôles, et on ignore l'entrée.

Pour reconnaître les entrées clavier, on définit l'énumération `touche` :

Pour représenter les entrées utilisateur, nous avons choisi une énumération contenant toute les touches utilisable dans notre jeu.

```
enum touche
{
    Haut = 65,        La touche directionnelle Haut
    Droite = 67,      La touche directionnelle Droite
    Bas = 66,         La touche directionnelle Bas
    Gauche = 68,      La touche directionnelle Gauche
    Rotation = 114,   La touche  r
    Placement = 112,  La touche  p
    Precedent = 97,   La touche  a
    Suivant = 122     La touche  z
};
```

5.3 Mise à jour des données du `gameState`

A chaque action du joueur, on doit recalculer le `gameState`. Il faut notamment retrouver le plus grand village (on cherche tous les villages grâce à `village_associe` et on garde le plus grand), recalculer le score, etc.

5.4 Calcul du village associé à une case Ville

On vérifie tout d'abord que la case donnée en paramètre est une case Ville.

Pour trouver le village maximal contenant une case Ville, on effectue une sorte de parcours de graphe. En effet, on maintient une liste de cases à

explorer, à laquelle on ajoute toutes les cases voisines de la case demandée qui sont aussi des cases Ville.

Tant que la liste des cases à explorer n'est pas vide, on continue d'y ajouter les éléments d'intérêt en explorant les voisins du premier élément.

Comme on ne dispose pas de liste dynamique en C (et pas envie d'implémenter un vecteur), on utilise un tableau de taille n^2 (avec n la taille du plateau) que l'on remplit initialement avec des -1 et qu'on parcoure de gauche à droite, en ajoutant les voisins d'intérêt à la fin (première case différente de -1).

Remarque comme on ne sait pas à l'avance quelle sera la taille du village maximal, la fonction renvoie un entier et prend en paramètre un tableau (dynamiquement alloué) d'entiers qu'il va vider puis remplir des informations intéressantes.

5.5 Calcul du score

Le score est calculé selon les règles données par le sujet à savoir :

— Chaque case de forêt compte 2 points ;

— Chaque ressource associée à une usine compte 4 points ;

— Un lac de n cases connexes vaut $3(n-1)$ points ;

— Le plus grand village rapporte un nombre de points égal à sa taille.

Pour les parties "compliquées" (taille du plus grand village par exemple), on utilise les données déjà calculées et présentes dans le `gameState`.

6 Organisation du travail

C'est Thomas qui a commencé à programmer cette partie. Il s'est ainsi occupé de la gestion de la boucle principale (donc de l'implémentation du type `gameState`), de l'interaction avec le `main`, des entrées clavier de l'utilisateur, des appels de fonctions de dessin d'interface, des modifications à effectuer sur les composants du jeu (grille, tuiles disponibles, etc.), du calcul du plus grand village et de la fonction de dessin des cadres.

Il restait ensuite trois grosses parties : dessin des tuiles restantes, dessin de la grille de jeu, calcul du score.

— Thomas : boucle de jeu (partie centrale du lot), débogage, création du `gameState`, calcul du plus grand village, calcul du village associé à une tuile ;
— Yoan : affichage des tuiles disponibles et sélection des tuiles ;
— Thibaut : affichage de l'état actuel de la grille (tuiles posées et tuiles en cours de pose par le joueur) ;
— Matthis : Calcul du score sur l'état actuel de la grille.

7 Problèmes rencontrés

Certains membres du groupe montrent une réelle réticence à prendre de l'avance sur le travail. Cela pose de graves problèmes notamment pendant les phases de rush où l'on doit être sûr que chacun a fait correctement son travail et où des confirmations écrites ou verbales sont nécessaires afin de pouvoir faire un rendu sereinement.

8 Comment compiler

Compilation de la documentation

Pour obtenir la documentation **Doxygen** :

```
make Doxygen
```

Puis ouvrir `html/index.html`.

Compiler le lot B et lancer une partie

```
make
./honshu_B
```

Lancer valgrind sur honshu_b

```
make vg_b
```

Remarque Il faut terminer la partie pour avoir un rapport correct de `valgrind`.