Projet C - DevLog

Rodolphe Herbin - Daniel Ben Said

Février 2024

Contents

1 Introduction

Le but de ce projet est de créer un jeu à un joueur. Le principe du jeu est d'insérer des pièces avec différentes formes et couleurs sur un plateau à une dimension (par défaut, 4 couleurs et 4 formes différentes seront utilisées). Trois pièces ou plus de même forme ou de même couleur à la suite disparaissent et donnent des points au joueur. Le joueur peut également décider de décaler des pièces d'une même forme ou d'une même couleur vers la gauche, c'est à dire que chaque forme prend la place de son premier voisin de gauche avec le même critère que celui choisi.

2 Implémentation

2.1 Structures de données

Le langage C manquant malheureusement pour nous de structure de liste avancée, nous allons devoir implémenter nous mêmes nos propres structures. Afin de répondre aux besoins de notre jeu, nous allons utiliser trois structures de données différentes.

2.1.1 Pièce

La structure la plus évidente à implémenter afin de faciliter grandement l'écriture et la compréhension des algorithmes mis en place est la structure de pièce. Chaque pièce stock deux entiers de type `char`, un correspondant à la forme de la pièce, l'autre correspondant à la couleur de la pièce. L'utilisation du type `char` a pour unique but de limiter l'utilisation de la mémoire et n'est pas utilisé pour représenter un caractère à part entière. Les deux entiers utilisés font office d'identificateurs pour chaque forme et couleur différentes.

2.1.2 Maillon

La structure de liste que nous utilisons est celle de la liste doublement chaînée circulaire. La structure de `Maillon` que nous avons défini rempli les standards de cette structure de liste. Elle comporte :

- Un pointeur vers un `Maillon` `next` qui représente le prochain maillon de la liste
- Un pointeur vers un `Maillon` `prev` qui représente le maillon précédent de la liste
- Un pointeur vers une `Piece` `f` qui fait office de valeur
- Un entier `mainID` qui représente l'index du maillon dans la liste principale

L'élément `mainID` de la liste ne fait pas beaucoup de sens pour l'instant, mais il sera très utile plus tard dans le programme.

2.1.3 Liste

L'élément final de notre structure de liste est la structure `Liste` en elle même. Nos listes étant circulaires et doublement chaînées, la structure `Liste` nous permet de garder un pointeur vers le premier `Maillon` de la liste ainsi qu'un entier stockant la taille actuelle de la liste.

2.2 Fonctions triviales

Afin de faciliter un maximum la programmation du jeu et d'éviter la redondance dans le code, nous avons implémenté quelques fonctions très simples et très courtes qui remplissent des fonctions spécifiques. En voici la liste avec une petite description.

- `Piece * getRandPiece(int n_shapes, int n_colors)`

  Prend en entrée le nombre de couleurs et le nombre de formes différentes et renvoie un pointeur vers une `Piece` avec une couleur et une forme choisies au hasard.

- `Liste * getList(void)`

  Ne prend rien en entrée et renvoie un pointeur vers une `Liste` initialisée et prête à être utilisée.

- `char * intToString(int n)`

  Prend en entrée un entier et renvoie un pointeur vers une chaîne de caractères contenant l'entier en question en chaîne de caractère.

2.3 Jeu

2.3.1 Insertions

La première pierre à la construction de notre est jeu est la fonction d'insertion. La fonction

```
void appendListe(Liste * l, Piece * f, char pos)
```

prend en argument un pointeur vers une `Liste`, un pointeur vers une `Piece` et un entier de type `char pos`. L'entier `pos` fait office de booléen, le langage C ne possédant pas ce type. L'insertion se fera au début ou à la fin de la liste en fonction de la valeur de `pos`, 0 ou 1.

Le programme commencera par créer un nouveau maillon dont la valeur pointera vers la forme mise en argument. L'insertion se fera d'abord entre le dernier élément et le premier, et le pointeur de la liste vers le premier élément sera changé vers notre nouveau maillon si l'insertion doit se faire au début de la liste. L'insertion se fait en temps constant.

2.3.2 Suppressions

La fonction

```
void remListe(Liste * l, Piece * rem)
```

prend en argument un pointeur vers une `Liste` et un pointeur vers une `Piece` et recherche la liste afin de lui retirer le maillon contenant la pièce en question. La fonction s'occupe d'actualiser les pointeurs vers les nouveaux maillons suivants et précédents de chaque côtés du maillon supprimé. La mémoire contenant le maillon retiré est libéré. Cette fonction tourne en $O(n)$.

2.3.3 Recherche de pièces consécutives

La prochaine étape du jeu est de trouver les pièces consécutives de même couleur ou de même forme afin de les retirer du plateau. Pour ce faire, la fonction

```
int checkListe(Liste * l, Piece ** ID)
```

prend en argument un pointeur vers une `Liste` et un pointeur vers une liste de pointeurs vers des `Piece` et retourne en entier correspondant au nombre de pièces trouvées. Les pièces trouvées sont stockées dans la liste passée en argument afin d'être retirées du plateau. La fonction commence par initialiser un compteur de formes similaires et un compteur de couleurs similaires. La liste est parcourue en ajoutant 1 à chaque compteur on en remettant à 1 chaque compteur en fonction des pièces avec des couleurs ou des formes similaires rencontrées. Pour chaque maillon parcouru, un pointeur vers sa pièce est stocké dans la liste passée en argument à l'indice 0 après que chaque autre pièce ait été décalée de 1. Si un des deux compteurs est supérieur ou égal à 3 après qu'une pièce différente ait été rencontrée ou que la liste arrive à sa fin, on renvoie le compteur en question. Sinon, on renvoie 0 à la fin de la fonction. Cette fonction tourne en $O(n)$.

2.3.4 Décalages

À ce stade de l'implémentation du jeu, on pourrait déjà créer une interface et jouer. Mais il reste la fonction la plus intéressante du jeu à mettre en place, celle des décalages. C'est ici que les choses de compliquent.

Afin de faciliter les décalages, les pièces de même couleur ou de même forme seront liées entre elles via des sous listes chaînées circulaire, en utilisant la structure présentée plus tôt. L'ordre dans la liste principale et dans les sous listes est le même. Des pointeurs vers chaque sous listes de couleurs ou de formes sont stockées dans deux listes `Liste ** collList` et `Liste ** shapList`. Les entiers `col` et `type` de la structure `Piece` sont les indices dans ces listes de la sous liste correspondant à la bonne couleur et la bonne forme. Nous avons fait deux versions différentes de la fonction, qui seront présentées ici.

La première fonction est la moins complexe algorithmiquement et utilise l'élément `mainID` de la structure `Maillon` présentée plus tôt. Le principe de

cette fonction est de parcourir la liste principale en même temps que la sous liste que l'on veut décaler. Chaque fois que l'on rencontre un élément de la sous liste dans la liste principale, on remplace la pièce du maillon en question par la pièce du maillon suivant (on aura au préalable stocké la première pièce dans une variable temporaire afin l'affecter au dernier maillon). Quand une pièce change de maillon, elle devrait potentiellement changer de position dans la deuxième sous liste dans laquelle on peut la trouver. La deuxième étape consiste donc à supprimer chaque pièce de la sous liste que l'on décale de sa deuxième sous liste puis à l'insérer dans cette même sous liste en fonction de son indice `mainID` dans la liste principale, à l'aide de la fonction

```
void appendListeID(Liste * l, Piece * f, int mainID)
```

qui marche similairement à la fonction d'insertion, mais qui ici insère la pièce juste avant le premier maillon rencontré donc le `mainID` et supérieur à celui passé en argument. Enfin, le pointeur vers le premier élément de la sous liste décalée est redirigé vers l'élément suivant afin de décaler la sous liste également.

Dans le principe, cette fonction marche et effectue correctement les décalages avec les bonnes réinsertions, excepté dans un unique cas particulier que nous n'avons pas réussi à identifier. Quelques fois, quand le décalage est effectué sur une liste de deux éléments, le décalage final de la sous liste ne s'effectue pas correctement. Quand on essaie ensuite d'effectuer le décalage d'une sous liste contenant un des éléments de cette sous liste, on observe une duplication de certaines pièces alors que d'autres sont écrasées. Si on essaie encore de décaler les mauvaises pièces, la fonction cherche alors indéfiniment une pièce inexistante afin de la remplacer par la suivante. Ce problème nous a obligé à finalement utiliser une autre fonction dans le projet, le code de cette première fonction est néanmoins disponible dans le code source sous le nom de `rotateMlist`.

Nous allons pouvoir maintenant présenter la fonction de décalage, qui marche à tous les coups cette fois ci. Cette fonction s'appelle `rotateMlist2` et prend en argument la liste principale, la sous liste à décaler, des pointeurs vers les listes de sous listes ainsi que le nombre de couleurs et de formes différentes dans la partie actuelle. Le début de la fonction est le même que la première pour le remplacement des pièces décalées, mais c'est à la réinsertion que la nouvelle fonction diffère. Après avoir décalé toutes les pièces dans la liste principale, toutes les sous listes sont vidées puis reconstruites à partir de la liste principale. Cette fonction tourne néanmoins en $O(n)$ étant donné que les insertions se font en temps constant, tout comme la fonction originale, mais fait cependant usage d'un peu moins de subtilité.

3 Interface utilisateur

Maintenant que le coeur du jeu fonctionne, il ne nous reste plus qu'à créer une interface afin de permettre à l'utilisateur de jouer. Nous allons créer deux interfaces différentes, une dans le terminal ainsi qu'une interface graphique.

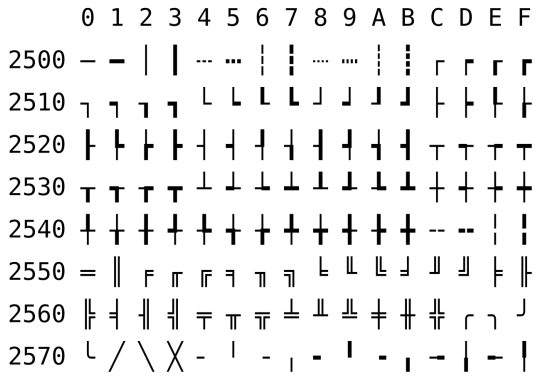
3.1 Terminal

3.1.1 Caractères d'échappement ASCII

Pour l'interface dans le terminale, nous allons utiliser les caractères d'échappement ASCII. Ces caractères spéciaux nous permettent de faire des actions spéciales dans le terminal, comme par exemple effacer des caractères ou écrire en couleur. Ces caractères spéciaux sont supportés nativement sur Linux et MacOS, mais pas sur Windows. Une fonction supplémentaire est utilisée afin d'autoriser l'utilisation des caractères d'échappement ASCII, elle se trouve dans le ficher `ansi_escape.c`.

3.1.2 Table ASCII étendue et table Unicode

Une autre partie du code qui diffère entre Linux, MacOS et Windows est l'affichage de caractères spéciaux. Afin de rendre l'interface dans le terminal plus jolie, nous allons utiliser des caractères de dessin de boite, dont voici un échantillon :



Ces caractères sont disponibles dans la table ASCII étendue ainsi que dans la table Unicode. Windows support la table ASCII étendue, en pouvant afficher des caractère dont le code ASCII est issu de deux chiffres en base 16. Cependant, sur Linux et MacOS, le seul moyen d'afficher les caractères spéciaux qui nous intéresse est la table Unicode, mais qui n'est pas supporté par toutes les versions de Windows, c'est pourquoi nous utilisons la table ASCII étendue pour Windows.

3.1.3 Affichage dans le terminal

Il n'y a que trois fonctions différentes qui sont utilisées pour l'interface dans le terminal. La fonction principale s'occupe de jouer au jeu et de gérer les instructions du joueur, tandis que deux fonctions utilitaires sont utilisée pour afficher l'état du jeu dans le terminal. La fonction principale,

```
void mainloopASCII(void)
```

commence par initialiser toutes les variables utilisées, puis la boucle principale permet d'afficher l'état du jeu après chaque action du jouer et d'effectuer les opérations nécessaires sur les différentes listes. L'affichage du plateau se fait grâce à deux fonctions. La première,

```
void printList(Liste * l)
```

est purement utilitaire permet d'afficher à la suite les éléments d'une liste passée en argument, avec la bonne couleur en utilisant les caractères d'échappement ASCII. La seconde est

```
void printState(Liste * mainlist, int score, Piece ** next)
```

et permet l'affichage de toutes les informations sur l'état du jeu, à l'aide des caractères de dessin de boite pour une meilleure mise en forme. Cette fonction existe en deux versions, une pour Windows qui utilise la table ASCII étendue et une autre pour Linux et MacOS qui utilise la table Unicode. La bonne version est choisie automatiquement à la compilation en fonction du système d'exploitation utilisé.

3.2 Interface graphique

L'interface graphique sera implémentée en utilisant la bibliothèque open source SDL 2.0, ainsi que l'extension SDL_gfx qui permet de dessiner des formes plus poussées que des points et des lignes.

3.2.1 Fonctions utilitaires

Du à la complexité de créer une interface graphique complète pour notre jeu, nous allons faire usage de quelques fonctions utilitaires.

- Afin d'afficher nos différentes pièces avec des belles couleurs dont la saturation est maximum, nous avons créé une fonction qui nous donne des valeurs RGB de ce type. Les valeurs RGB de couleur avec une saturation maximum sont de la forme : une valeur à 0, une valeur à 255, et une valeur entre 0 et 255. Cela nous donne $6 * 256 = 1536$ valeurs RGB différentes avec une saturation maximum. La fonction

  ```
  void colorFromNumber(int n, int * r, int * g, int * b)
  ```

associe à chaque valeur entre 0 et 1535 la valeur RGB correspondant à une couleur saturée au maximum.

- Pour ne pas avoir à utiliser un nombre incalculable d'assets de polygones, surtout si on souhaite jouer avec plus de 4 formes ou couleurs différentes, il serait judicieux de créer un générateur de polygones. C'est pour répondre à ce besoin que nous créé la fonction `void drawpoly`. Cette fonction prend en entrée un moteur de rendu SDL, un nombre de côté, des coordonnées x et y, une taille ainsi que des valeurs RGB, et dessine un polygone régulier avec le bon nombre de côtés, aux bonnes coordonnées et avec la bonne couleur. Cette fonction utilise les matrices de rotations afin de calculer les coordonnées de tous les sommets du polygone les uns à la suite des autres, en dessinant entre eux des lignes épaisses à l'aide de SDL_gfx. Des cercles sont également dessinés sur chaque sommet afin d'obtenir des polygones arrondis. L'angle à utiliser pour la matrice de rotation est $\frac{2\pi}{n}$, avec n le nombre de côtés du polygone à dessiner. La fonction utilitaire `void rota(double * x, double * y, double angle)` permet d'effectuer la rotation sur les coordonnées x et y d'un point.

- Malheureusement pour nous, le module SDL_gfx ne permet pas d'afficher des polygones remplis avec anti-aliasing. On peut cependant afficher des polygones non remplis avec anti-aliasing, puis le remplir nous même. La fonction `void aathickLineRGBA` rempli cette fonction, en permettant de dessiner des lignes épaisses entre deux points avec anti-aliasing. La fonction prend en argument un moteur de rendu SDL, les coordonnées x et y de deux points, l'épaisseur de la ligne, ainsi que des valeurs RGBA pour la couleur. La fonction commence par calculer les 4 coordonnées des sommets du polygone anti-aliasé qui va entourer notre ligne épaisse. On commence par calculer le vecteur entre les deux points passés en argument dans la fonction. On calcule ensuite un vecteur orthogonal à celui-ci, en réutilisant la fonction `rota` avec un angle de $\frac{\pi}{2}$. On normalise ensuite le vecteur, et on ajoute ou retire aux deux points de la ligne notre vecteur en fonction de l'épaisseur de la ligne afin d'obtenir les coordonnées des 4 sommets du polygone.

3.2.2 Différents écrans de jeu

Menu principal Le premier écran que rencontre le joueur est le menu principal. Ce menu comporte le nom du jeu, rendu à l'aide de SDL_ttf, et de boutons permettant de choisir le nombre de pièces maximum sur le plateau (de 10 à 30), ainsi que le nombre de formes et de couleurs différentes dans le jeu (de 2 à 9). Le joueur peut également décider de charger une sauvegarde du jeu d'une partie précédente. Une fois les bons paramètre sélectionnés par le joueur, le bouton `Play Game` permet de lancer la partie avec les paramètres choisis par le joueur.

Jeu Une fois les paramètres de jeu confirmés, la fonction du menu principal appelle la fonction de jeu avec en argument le nombre de pièces maximum, le

nombre de couleurs différentes et le nombre de pièces différentes. Le joueur peut insérer des pièces à gauche soit en utilisant les flèches directionnelles, soit avec deux boutons à gauche et à droite de l'écran. Pour les décalages de formes, des boutons cliquables sont dessinés en bas à gauche de l'écran en fonction du nombre de formes et de couleurs spécifiés. Le score est affiché en haut à gauche, et le nombre de formes sur le plateau est affiché en haut à droite. Les 5 prochaines formes sont affichées en bas de l'écran, la forme à insérer est affichée plus grande que les autres.

Menu de fin Une fois le nombre maximum de formes autorisées sur le plateau atteint, la fonction de jeu appelle la fonction de l'écran de fin en lui transmettant le score obtenu. L'écran de fin affiche le score final, les meilleurs scores ainsi que deux boutons pour rejouer au jeu ou quitter le jeu.

References

[1] https://en.wikipedia.org/wiki/Box-drawing_character

[2] https://theasciicode.com.ar/

[3] https://solarianprogrammer.com/2019/04/08/c-programming-ansi-escape-codes-windows-macos-linux-terminals/

[4] https://gist.github.com/fnky/458719343aabd01cfb17a3a4f7296797

[5] https://wiki.libsdl.org/SDL2/FrontPage

[6] https://gerald-burke.itch.io/geralds-keys