Compléments d'informatique

Projet 1 : Bejeweled

6 novembre 2020

Dans ce projet, on se propose d'implémenter une version du jeu Bejeweled et de la résoudre automatiquement. L'objectif pédagogique principal est de vous apprendre à organiser votre code de façon claire et modulaire. Pour ce faire, vous allez devoir écrire des algorithmes de difficulté intermédiaire en exploitant la plupart des concepts vus au cours théorique jusqu'à présent (i.e. algorithmes itératifs et récursifs, structures opaques, pointeurs de fonction...). Ce projet est à réaliser par **groupe de deux étudiants maximum**. La date limite de remise du projet sur la plateforme de soumission est indiqué dans Ecampus.

1 Bejeweled : les règles du jeu

Bejeweled est un jeu de tile-matching [1] où l'objectif est de faire exploser des gemmes de sept couleurs différentes situées sur un plateau rectangulaire et vertical à deux dimensions (voir la figure 1 pour une illustration du plateau jeu). Le principe du jeu est d'échanger la position de deux gemmes adjacentes (horizontalement ou verticalement) afin de créer des alignements horizontaux ou verticaux de trois gemmes d'une même couleur. Lorsqu'un échange entraine un alignement d'au moins trois gemmes, celles-ci sont détruites. Les gemmes se situant au dessus de ces dernières prennent alors leurs places et ainsi de suite jusqu'à ce que de nouvelles gemmes soient générées tout en haut du plateau. Suite à la destruction de gemmes et le repositionnement des restantes, il est possible que de nouveaux alignements d'au moins 3 gemmes se forment. Celles-ci sont alors également détruites, initiant une réaction en chaine qui s'arrête lorsque plus aucun alignement (d'au moins 3 gemmes) ne subsiste. La partie s'arrête dans deux conditions :

— Il n'est plus possible de générer un nouvel alignement par l'échange de deux gemmes.
— Un nombre maximum d'échanges a été atteint.

L'objectif étant de maximiser le nombre de gemmes détruites sur la durée de la partie.

1. `https://fr.wikipedia.org/wiki/Jeu_de_tile-matching`


```
  01234567
0 BBYGYOBR
1 BGBPOYRG
2 YRGWYORP
3 YOYBBWGY
4 OOYBWPBG
5 GPWOPYRY
6 BBGWRBWP
7 RORORGRO
```

FIGURE 1 – Plateau de jeu de Bejeweled 2 (Source : `https://bejeweled.fandom.com/wiki/Classic`), à gauche, et la représentation attendue de ce plateau par la fonction `boardPrint`, à droite. Si l'on échange les gemmes des positions $(0, 1)$ et $(1, 1)$, les trois gemmes bleues se retrouvant au début de la ligne 1 (qui est la deuxième ligne) exploseront. Si on joue au mode explosif, cette explosion entrainera également l'explosion de la gemme bleue en position $(0, 0)$.

Mode explosif. Dans la suite, nous appellerons le mode de jeu décrit ci-dessus le mode *classique*. Dans ce mode, un échange entrainera la destruction d'au minimum 3 gemmes et d'au maximum 10 gemmes (2 alignements de 5 gemmes de couleurs différentes), sans compter les réactions en chaine. Un deuxième mode de jeu sera également implémenté, appelé le mode *explosif*. Le principe de ce mode est le même que le précédent si ce n'est que l'explosion d'une gemme au sein d'un alignement entrainera l'explosion de toutes les gemmes adjacentes de la même couleur, ainsi que celles adjacentes à ces dernières, et ainsi de suite. L'adjacence des gemmes de même couleur ne peut être que verticale ou horizontale, pas diagonale. Le même principe d'explosion des gemmes adjacentes à celles alignées s'applique pour les explosions issues de la réaction en chaine.

Indices. Pour aider le joueur à se débloquer, un indice peut lui être fourni s'il en fait la demande (voir le bouton "hint" sur la figure 1). Cet indice est donné sous la forme d'une paire de gemmes adjacentes dont l'échange mènera à au moins un alignement de gemmes. Un joueur pourrait ainsi jouer une partie entière de manière automatique en utilisant à chaque étape l'indice qui lui est proposé. Nous utiliserons cette possibilité pour implémenter un joueur artificiel du jeu et étudier ses performances.

2 Implémentation

Le code du jeu est divisé en trois modules à implémenter séparément. Tout d'abord, il y a le plateau de jeu où sont implémentées les règles du jeu. Ensuite, il y a l'indice, qui permet d'indiquer au joueur un échange possible afin de créer un alignement. Ce module permettra également de déterminer s'il reste des échanges possibles sur le plateau courant. Finalement, il y a le joueur, qui est responsable d'initialiser le jeu et d'y jouer jusqu'à avoir effectué le bon nombre d'échanges.

2.1 Plateau de jeu

Le premier module à implémenter dans les fichiers `board.c/h` est un type de données abstrait permettant de gérer un plateau de jeu. Vous définirez votre propre structure de données `Board` (lisez bien l'énoncé en entier avant), qui devra être rendue opaque, et implémenterez les fonctions de manipulations du plateau en respectant les spécifications ci-dessous. Dans les arguments de ces fonctions, les coordonnées (i, j) d'une case seront telles que la case $(0, 0)$ sera située dans le coin supérieur gauche du plateau et la case $(h - 1, w - 1)$ dans le coin inférieur droit, où h et w sont respectivement la hauteur et la largeur du plateau. Une structure (non opaque) de type `Swap` est définie dans `board.h` qui permet d'encoder les coordonnées des cases à échanger. La couleur d'une case sera codée par un caractére (type `char` en c) pris parmi `{'R', 'G', 'B', 'Y', 'P', 'O', 'W'}` (pour rouge (R), vert (G), bleu (B), jaune (Y), mauve (P), orange (O), blanche (W)).

Fonctions de l'interface. Les fonctions à implémenter dans le fichier `board.c` sont les suivantes :

`Board *boardInit(int width, int height, int explosive)` : crée un nouveau plateau de largeur `width` et de hauteur `height`. L'argument `explosive` permet d'indiquer le mode de jeu : 0 pour le mode classique et 1 pour le mode explosif. Cette fonction sera appelée au début de la partie afin d'allouer l'espace mémoire nécessaire en fonction des paramètres de taille. Lors de la création, les gemmes du plateau de jeu devront être initialisées au moyen de la fonction `boardFillRandom` ci-dessous. La largeur et la hauteur d'une grille ne pourront pas être inférieures à 3 (pour permettre d'avoir des alignements). Si c'est le cas, la fonction devra renvoyer un pointeur `NULL`.

`void boardFillRandom(Board *board)` : remplit l'entièreté du plateau aléatoirement avec des gemmes. On supposera que cette fonction est capable de créer directement un plateau qui ne contient aucun alignement de plus de 3 gemmes d'une même couleur (voir plus bas pour les contraintes d'implémentation de cette fonction).

`void boardFree(Board *board)` : libère la mémoire occupée par le plateau pointé par

`board`.

`void boardPrint(Board *board)` : affiche le plateau pointé par `board` de manière textuelle. La grille sera affichée ligne par ligne, colonne par colonne. Chaque case sera représentée par la lettre correspondant à la couleur de la gemme. Au sommet de la table et au début de chaque ligne, on affichera le chiffre des unités de l'indice de la colonne et de la ligne pour faciliter l'encodage de l'échange par l'utilisateur. Un exemple de sortie attendue de la fonction `boardPrint` est donné à la figure 1.

`int boardSwap(Board *board, Swap swap, int execute)` :
Cette fonction renvoie le nombre total de gemmes détruites lors de l'échange des gemmes aux positions (`s.i1`, `s.j1`) et (`s.i2`, `s.j2`). Le booléen `execute` indique si l'échange doit être exécuté ou non sur le plateau. Si `execute` est différent de 0, le plateau est modifié et la fonction renvoie le nombre de gemmes détruites, y compris celles détruites par la réaction en chaine créée par la destruction d'un possible alignement formé par cet échange. Si `execute` est égal à 0, l'échange n'aura pas lieu et aucune gemme ne sera détruite mais la fonction renverra le nombre de gemmes qui auraient été détruites si l'échange avait eu lieu. On ne comptabilisera pas dans ce nombre les gemmes détruites par les réactions en chaine qui auraient été provoquées par la première salve de destruction. Dans le mode explosif, on prendra par contre en compte les gemmes adjacentes de même couleur qui seront détruites. Quelle que soit la valeur de `execute`, si aucun alignement de plus de 3 cases n'est formé par l'échange demandé, les deux gemmes ne sont pas échangées. Il en est de même si les coordonnées ne sont pas valides. Dans ces deux derniers cas, la fonction devra renvoyer 0.

`void boardAssignGem(Board *board, int i, int j, char c)` : assigne la couleur `c` à la position (`i`, `j`) dans le plateau pointé par `board`.

`char boardGetGem(Board *board, int i, int j)` : renvoie la couleur de la gemme située à la position (`i`, `j`) dans le plateau pointé par `board`.

`int boardWidth(Board *board)` : renvoie la largeur du plateau pointé par `board`.

`int boardHeight(Board *board)` : renvoie la hauteur du plateau pointé par `board`.

2.2 Indice

Le second module, à implémenter dans les fichiers `hint.c/h`, est une librairie implémentant deux fonctions permettant de donner un indice au joueur. La première fonction, `hintRandom`, fournit comme indice un échange au hasard parmi tous les échanges possibles qui entrainent une destruction. La deuxième, `hintBestExplosion` fournit l'échange (ou un des échanges) menant à la destruction du plus grand nombre de gemmes (sans prendre en

compte les réactions en chaine). Nous vous demanderons dans la section 3 de comparer les performances de ces deux fonctions si le joueur choisit de les utiliser systématiquement pour jouer. Ces fonctions permettront également de déterminer à chaque étape du jeu s'il reste des échanges dans le plateau courant.

Fonctions de l'interface. Les fonctions à implémenter dans le fichier `hint.c` sont les suivantes :

`Swap hintRandom(Board *board)` : fournit un échange aléatoirement sélectionné parmi tous ceux qui créent un alignement. Si aucun échange ne peut créer d'alignement, les champs de la structure de type `Swap` vaudront -1 par convention.

`Swap hintBestExplosion(Board *board)` : renvoie l'échange qui détruit le plus de gemmes, sans compter les réactions en chaine causées par l'apparition de nouvelles gemmes au sommet du plateau. Si le joueur joue au mode explosif, le nombre de gemmes adjacentes à l'explosion doit être pris en considération. Comme pour la fonction précédente, si aucun échange ne peut créer d'alignement, les champs de la structure de type `Swap` devront valoir -1.

2.3 Joueur

Le module implémenté dans les fichiers `play.c/h` est une librairie contenant une fonction principale, `game`, permettant de jouer une partie de Bejeweled. Le comportement du joueur est déterminé par une fonction, fournie en argument à la fonction `game`, qui, à partir du plateau et d'un indice, détermine l'échange à exécuter. On implémentera deux joueurs :

- un joueur humain entrant au clavier les coordonnées des gemmes à échanger (et ayant également la possibilité de visualiser l'indice s'il est bloqué)
- un joueur artificiel qui joue systématiquement l'indice.

Fonctions de l'interface. L'interface de ce module contiendra trois fonctions :

`Swap humanPlayer(Board *board, Swap hint)` : cette fonction devra lire, au moyen de la fonction `scanf`, quatre entiers rentrés par l'utilisateur au clavier et renvoyer une structure de type `Swap` encodant l'échange choisi dans le plateau pointé par `board`. Si le joueur entre l'échange invalide `-1 -1 -1 -1`, la fonction lui affichera l'indice et il pourra entrer à nouveau un échange.

`Swap onlyHintPlayer(Board *board, Swap hint)` : Cette fonction renverra toujours l'échange `hint`.

```
int game(int width, int height, int explosive, int nMaxSwaps,
         Swap (*computeHint)(Board *),
         Swap (*chooseSwap)(Board *, Swap), int show):
```

Cette fonction sert à gérer une partie complète du jeu. Les quatre premiers paramètres de cette fonction, `width`, `height`, `explosive` et `nMaxSwaps` définissent respectivement la largeur et la hauteur du plateau, le mode de jeu (0 pour pour le mode classique, 1 pour le mode explosif) et le nombre maximum d'échanges avant d'atteindre la fin de la partie. Les fonctions `computeHint` et `chooseSwap` passées en argument par pointeur déterminent respectivement la fonction utilisée pour fournir un indice et la fonction implémentant le joueur. Enfin, `show` est un booléen qui, s'il est différent de zéro, indique que la fonction doit afficher le plateau après chaque échange de gemme, en ce compris le plateau initial et final. L'affichage ne tient pas compte des explosions intermédiaires lors des réactions en chaine. Si le paramètre `show` est égal à 0, aucune information ne sera affiché à l'écran par la fonction. La fonction `game` devra se charger d'initialiser le plateau. Ensuite, chaque étape de jeu consistera à vérifier à l'aide de la fonction `computeHint` si un échange est possible (et si ce n'est pas le cas, arrêter la partie), à obtenir un échange du joueur via la fonction `chooseSwap`, et à l'appliquer au plateau de jeu. La fonction devra gérer le calcul du nombre de gemmes détruites par le joueur et arrêter la partie lorsque le nombre d'échanges atteindra nMaxSwaps. En fin de partie, la fonction devra renvoyer le nombre total de gemmes qui auront été détruites.

3 Performance des indices

Afin de déterminer s'il y a réellement un intérêt à déterminer l'échange maximisant le nombre de gemmes détruites, il vous est demandé d'utiliser votre implémentation pour calculer les statistiques suivantes (sur 1000 parties) :

- La moyenne et l'écart-type du nombre de gemmes détruites lors d'une partie complète
- La moyenne et l'écart-type du nombre d'échanges avant d'arriver en fin de partie

Vous ferez ce calcul pour les deux fonctions d'indices et dans les deux modes de jeu, en supposant que la plateau de jeu a une taille standard de 8×8 et qu'un nombre maximum de 1000 échanges est permis. Ces résultats seront à indiquer dans un fichier à remettre avec le projet (voir la section 5).

4 Conseils d'implémentation

Le premier fichier à compléter est le fichier `board.c`, puisque les deux autres en dépendent, suivi de `hint.c` et ensuite `play.c`. Il est néanmoins possible d'implémenter ces fichiers en parallèle (si vous travaillez en groupe), leur interface étant connue.

`board.c :` Dans `board.c`, commencez par les fonctions de création et de manipulation de la grille : `boardInit`, `boardAssignGem`, `boardGetGem`, `boardWidth`, `boardHeight` et `boardPrint`. Ces fonctions ne devraient pas vous poser de problème une fois que vous aurez décidé de la structure. Cette dernière doit être opaque et définie uniquement dans le fichier `board.c`.

Les fonctions les plus complexes du module sont les fonctions `boardFillRandom` et `boardSwap`.

Pour `boardFillRandom`, on souhaite que votre fonction remplisse la table en une passe et un temps borné dans tous les cas [2]. Cela est possible en remplissant la table de haut en bas et de gauche à droite en s'assurant de ne pas créer d'alignement à chaque ajout d'une nouvelle gemme.

Pour `boardSwap`, plusieurs solutions sont possibles. Une suggestion est d'écrire une fonction qui va marquer dans la grille les gemmes qui devraient être détruites si l'échange avait lieu et une fonction qui va ensuite détruire les gemmes en fonction de ce marquage. La première fonction vous permettra de tester l'effet d'un échange dans le cas où l'argument `execute` sera à 0 et vous pourrez gérer les réactions en chaine par des applications successives de ces deux fonctions. Pensez à la récursivité pour gérer le mode explosif.

Pour détecter vos erreurs d'implémentation, nous vous encourageons vivement pour ce module (et aussi pour les suivants) d'écrire un fichier test.c incluant une fonction `main` testant les différentes fonctions de l'interface. Vous pourrez vous servir de la fonction `boardAssignGem` pour fixer manuellement les valeurs dans la grille et de la fonction `boardPrint` pour afficher cette dernière. Il pourrait être intéressant également de modifier cette fonction lors du débuggage pour afficher des informations intermédiaires (les gemmes marquées par exemple).

Pour information, notre implémentation pour `board.c` fait approximativement 300 lignes.

`hint.c :` L'implémentation des fonctions de cette librairie est grandement facilitée par la disponibilité du mode `execute=0` de la fonction `boardSwap`. La fonction `hintBestExplosion` est plus simple à implémenter que la fonction `hintRandom` et ne devrait pas vous poser de problème. Pour `hintRandom`, la principale difficulté est de trouver un mécanisme pour pouvoir parcourir les échanges de manière aléatoire.

Notre implémentation pour `hint.c` fait approximativement 100 lignes.

`play.c :` L'implémentation de ce fichier n'appelle pas de commentaire particulier. Notre implémentation fait approximativement 80 lignes.

Gestion des erreurs : Nous ne testerons pas vos fonctions avec des arguments non conformes. Cependant, n'hésitez pas à adopter une programmation défensive et à afficher

2. Par exemple, tirer des grilles au hasard jusqu'à en avoir une bonne n'est pas une solution acceptable.

un message d'erreur approprié en cas problème. Vous pouvez dans ce cas simplement arrêter l'exécution du programme.

Réalisation des tests : Pour réaliser les mesures demandées à la section 3, vous devrez créer un nouveau fichier contenant une fonction `main` dédiée. A priori, l'interface qu'on vous propose ne vous permet pas de récupérer le nombre d'échanges avant d'atteindre la fin de la partie. Vous devrez faire des modifications dans la fonction `game` et dans les fichiers `play.c/.h` pour pouvoir récupérer cette information. Attention à ne pas soumettre les fichiers modifiés avec votre projet.

5 Soumission

Le projet doit être soumis via la plateforme de soumission sous la forme d'une archive au format `zip` ou `tar.gz` contenant les fichiers suivants :

- `board.c`
- `hint.c`
- `play.c`
- `rapport.txt`

Les fichiers `board.h`, `hint.h` et `play.h`, qui ne doivent pas être modifiés, ne doivent pas être fournis dans l'archive. S'ils le sont malgré tout, ils seront remplacés par nos fichiers lors de la compilation de votre code, ce qui risque d'empêcher la compilation de votre code si vous avez modifié ces fichiers. Si besoin, vous pouvez soumettre des modules supplémentaires sous la forme de fichiers `.c/.h` dont vous auriez besoin pour votre implémentation. Ces fichiers doivent cependant être inclus dans les fichiers `.c` des autres modules, pas dans les fichiers `.h`.

Le fichier `rapport.txt` contiendra les résultats demandés dans la section 3 et devra respecter scrupuleusement le format du fichier fourni (il sera analysé automatiquement). Notez que si le code que vous remettez ne permet vraisemblablement pas de faire les tests demandés, vous n'obtiendrez pas la partie des points relative à ces tests pour le projet même si vous indiquez des résultats dans `rapport.txt`.

Dans ce même fichier, en cas de travail en groupe, il vous est demandé d'indiquer la **contribution de chacun au projet en pourcentage**. Ces valeurs doivent être acceptées par les deux étudiants du groupe lors de la soumission. Comme annoncé au début de l'année, la cote d'un étudiant pourrait être rabaissée en cas de participation trop faible aux projets.

Un fichier `bejeweled.c` permettant d'exécuter une partie utilisant un joueur humain ou l'heuristique (en fonction des paramètres fournis sur la ligne de commande) vous est fourni pour tester votre implémentation. Il ne doit pas non plus être inclus dans votre archive.

Vos fichiers seront compilés et testés sur la plateforme de soumission en utilisant le fichier `Makefile` fourni via la commande `make all` ou de manière équivalente en utilisant la commande suivante :

```
gcc -o bejeweled bejeweled.c board.c hint.c play.c
```

en ajoutant les flags de compilation habituels (`-std=c99 -pedantic -Wall -Wextra -Wmissing-prototypes`), qui ne devront déclencher aucun avertissement lors de la compilation, sous peine d'affecter votre cote.

Votre programme devra alors s'utiliser comme suit :

```
./bejeweled h w v m p i s
```

où `h` et `w` sont la hauteur et la largeur du plateau, `v` indique le mode de jeu (0 pour le mode classique, 1 pour le mode explosif), `m` indique le nombre maximum d'échanges avant que la partie ne soit arrêtée, `p` est le type de joueur (0 si humain, 1 si l'indice), `i` est le type d'indice (0 si aléatoire ou 1 sinon) et `s` indique si on affiche le plateau à chaque échange (1 si oui, 0 si non)). Par défaut, la commande `make run` lancera la commande :

```
bejeweled 8 9 1 10 0 0 1
```

utilisant une grille 8×9, le mode explosif, 10 échanges maximum pour un joueur humain avec l'indice aléatoire et l'affichage du plateau de jeu.

Toutes les soumissions seront soumises à un programme de détection de plagiat. En cas de plagiat avéré, les étudiant concernés se verront affecter une cote nulle au projet (au minimum).

Bon travail !