

INFO0030: Five or more

Groupe 07: Lucas LOLO, Marzouk OURO-GOMMA

Table des matières

1	Architecture générale du code	3
1.1	Le Modèle	3
1.2	La Vue	3
1.3	Le Controleur	3
1.4	Interaction	3
2	Les structures de données	4
2.1	Structure ModeleFive	4
2.2	Structure VueFive	4
2.3	Structure ControleurFive	5
3	Les algorithmes particuliers	6
3.1	Déplacement (labyrinthe)	6
3.2	Vérification horizontale	6
3.3	Vérification verticale	7
3.4	Vérification croissante	7
3.5	Vérification décroissante	8
3.6	Génération image aléatoire	9
4	Profiling et analyse de performance du code	10
5	L'interface graphique du jeu	10
5.1	Vue générale du jeu	10
5.2	Organisation du jeu	11
6	La gestion du code	11
7	La coopération au sein du groupe	12
8	Les améliorations possibles	12
9	Les éléments que nous avons appris	12
9.1	Conclusion	12

1 Architecture générale du code

Pour l'architecture générale du code nous avons utilisé le pattern MVC.

1.1 Le Modèle

Contient les fonctionnalités de l'application, qui agissent sur la structure pour modifier ses informations et son contenu.

1.2 La Vue

Contient les fonctions qui modifient les éléments à afficher sur la fenêtre de l'application.

1.3 Le Contrôleur

Contient les fonctions qui font le lien entre les entrées externes "click" et le modèle qui contient les données.

1.4 Interaction

Notre code utilise trois structures de données, une pour les données de jeu, une autre pour les éléments visuels (images) et la dernière combine les deux structures mentionnées précédemment. La partie modèle rassemble les données d'exploitation sans modifier l'effet visuel, tandis que la partie vue modifie les éléments en raison des données. Enfin, le contrôleur transforme l'opération de l'utilisateur en une opération sur le modèle.

2 Les structures de données

2.1 Structure ModeleFive

```
1 typedef struct modele_t{  
2     int niveau; //Contient le niveau choisit.  
3     int largeur; //Contient la largeur de la grille.  
4     int hauteur; //Contient la hauteur de la grille.  
5     int typeSymboles; //Contient le nombre de couleur au cours de la partie.  
6     int symbolesTour; //Contient le nombre de symbole a ajouté à chaque tour.  
7     int cellules; //Contient le nombre de case de la grille.  
8     int grille[20][15]; //Une grille contenant la couleur d'une image.  
9     int aleatoire[20][15]; //Une grille contenant les positions des cases blanches.  
10    int combo; //Contient le combo maximale.  
11    int avancer_tableau_horizontale; //Permet d'avancer pour la victoire d'un combo.  
12    int avancer_tableau_verticale; //Permet d'avancer pour la victoire d'un combo.  
13    int avancer_tableau_oblique; //Permet d'avancer pour la victoire d'un combo.  
14    int oblique_decroissant; //Permet d'avancer pour la victoire d'un combo.  
15    int score; //Contient le score courant de l'utilisateur.  
16    int ligneActuelle[10]; //Contient les 10 meilleures scores de l'utilisateur.  
17    int nouvelle_partie; //Permet de savoir combien de partie ont été jouée.  
18 }ModeleFive;
```

Extrait de Code 1 – Structure modele

La structure ModeleFive :

Nous permet de stocker les informations propre du modele.

Pertinence :

Cette structure nous permet de l'utiliser pour obtenir des informations au cours du jeu. Les champs sont détaillés ci-dessus. Le cout des informations en "int" est sensée car nous stockons uniquement des nombres.

2.2 Structure VueFive

```
1 typedef struct vue_t{  
2     ModeleFive *mf;  
3     GtkWidget **image; //Contient le nombre d'image a chargé et a modifié.  
4     GtkWidget **image_aleatoire; //Contient le nombre d'image a chargé et a modifié pour les  
5     informations prochaines.  
6     GtkWidget **bouton_aleatoire; //Contient le nombre de bouton pour les informations prochaines.  
7     GtkWidget *Score; //Permet d'afficher le score sur le tableau de bord.  
8     int couleur_actuelle[7]; //Est la position du bouton qui donne les informations.  
9 }VueFive;
```

Extrait de Code 2 – Structure vue

La structure VueFive :

Contient les éléments visuels du jeu qui sont aussi expliqués ci-dessus.

Pertinence :

Utilisation des malloc pour les widgets semblent une bonne idée.

2.3 Structure ControleurFive

```
1 typedef struct controleur_t{  
2     VueFive *vf;  
3     ModeleFive *mf;  
4     GtkWidget **bouton; //Contient le nombre de bouton dans le jeu.  
5     GtkWidget *Fenetre; //Est la fenêtre courante du jeu.  
6     char *fichier_score; //Est le fichier de l'utilisateur.  
7     int nombre; //Permet de savoir si l'utilisateur a bien fait les deux clicks.  
8     int x_init; //Contient la coordonnée horizontale du premier click.  
9     int y_init; //Contient la coordonnée verticale du premier click.  
10    int x_dest; //Contient la coordonnée horizontale du deuxième click.  
11    int y_dest; //Contient la coordonnée verticale du deuxième click.  
12 }ControleurFive;
```

Extrait de Code 3 – Structure controleur

La structure ControleurFive :

contient l'ensemble des autres structures puisqu'elle doit les mettre en lien avec les interaction de l'utilisateur.

Pertinence :

une structure qui rassemble toutes les autres est très utile dans un jeu aussi complexe que le jeu five or more. Petit bémol, le fichier n'a pas été utilisé par soucis de temps ou le mal compris de l'énoncé.

3 Les algorithmes particuliers

3.1 Déplacement (labyrinthe)

```
1 int deplacement(ControleurFive *cf){  
2     voir ControleurFive.c  
3 }
```

Idée :

Cette algorithme a pour but de déplacer une image d'un endroit à l'autre du plateau si le chemin existe.

Implémentation :

Cette algorithme est composé de 4 parties importantes qui servent à déplacer l'image en fonction de sa destination (monter à gauche ou droite ou descendre à gauche ou droite). L'algorithme se dirigera toujours vers la destination la plus proche et donc prendra seulement un chemin différent si le chemin le plus court s'avère bloquer.

3.2 Vérification horizontale

```
1 int verification_horizontale( ControleurFive *cf){  
2     for(int ComboAll = cf->mf->combo ; ComboAll >= 5 ; ComboAll--){  
3         for(int k = 0; k < cf->mf->avancer_tableau_horizontale; k++){  
4             for(int i = 0; i < cf->mf->hauteur; i++){  
5                 for(int j = 1; j < cf->mf->typeSymboles+1; j++){  
6                     //Avec tous les if  
7                 }  
8             }  
9         }  
10    }  
11 }
```



Idée :

Cette algorithme pourrait avoir une meilleure complexité mais dans l'idée, elle, nous sert à savoir s'il y a eu un combo horizontale ou non.

Implémentation :

Tout d'abord, dans la première boucle nous initialisons la variable "ComboAll" à la plus grande valeur possible du combo et nous la décrementons jusqu'à arriver à 5 qui est le strict minimum. La variable "k", nous permet d'avancer d'un pas vers la droite, la variable "i" sert à se déplacer vers le bas et enfin la variable "j" permet de savoir s'il y a bien une suite d'une même couleur. Voir dessin au dessus.

3.3 Vérification verticale

```

1 int verification_verticale( ControleurFive *cf){
2     for(int ComboAll = cf->mf->combo ; ComboAll >= 5 ; ComboAll--){
3         for(int k = 0; k < cf->mf->avancer_tableau_verticale; k++){
4             for(int i = 0; i < cf->mf->largeur; i++){
5                 for(int j = 1; j< cf->mf->typeSymboles+1; j++){
6                     }
7                     // Avec tous les if
8                 }
9             }
10        }
11    }

```



Idée :

Cette algorithme pourrait avoir une meilleure complexité mais dans l'idée, elle, nous sert à savoir s'il y a eu un combo verticale ou non.

Implémentation :

Tout d'abord, dans la première boucle nous initialisons la variable "ComboAll" à la plus grande valeur possible du combo et nous la décrementons jusqu'à arriver à 5 qui est le stricte minimum. La variable "k", nous permet d'avancer d'un pas vers le bas, la variable "i" nous sert à se déplacer vers la droite et enfin la variable "j" permet de savoir s'il y a bien une suite d'une même couleur. Voir dessin au dessus.

3.4 Vérification croissante

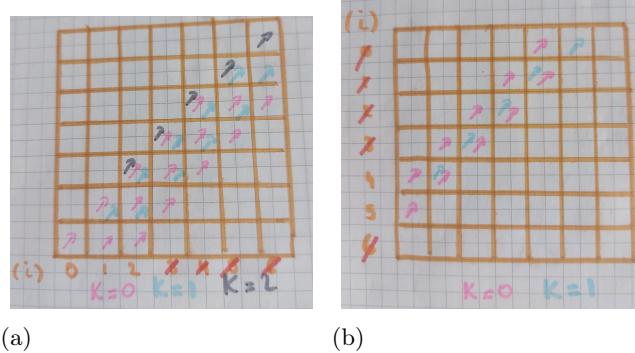
```

1 int oblique_croissant1(ControleurFive *cf){
2     for(int ComboAll = cf->mf->combo; ComboAll >= 5; ComboAll--){
3         for(int i = 0; i < cf->mf->largeur - 3; i++){
4             for(int k = 0; k < cf->mf->avancer_tableau_oblique; k++){
5                 for(int j = 1; j< cf->mf->typeSymboles +1; j++){
6                     }
7                     //Avec tous les if
8                 }
9             }
10        }
11    }
12 int oblique_croissant2(ControleurFive *cf){
13     for(int ComboAll = cf->mf->combo; ComboAll >= 5; ComboAll--){
14         for(int i = 4;i < cf->mf->hauteur - 1; i++){
15             for(int k = 0; k < cf->mf->avancer_tableau_oblique - 1; k++){
16                 for(int j = 1; j< cf->mf->typeSymboles + 1; j++){
17                     }
18                     //Avec tous les if
19                 }

```

```

20     }
21 }
22 }
```



(a)

(b)

Idée :

Cette algorithme pourrait avoir une meilleure complexité mais dans l'idée, elle, nous sert à savoir s'il y a eu un combo croissant ou non.

Implémentation :

Tout d'abord, dans la première boucle nous initialisons la variable "ComboAll" à la plus grande valeur possible du combo et nous la décrementons jusqu'à arriver à 5 qui est le stricte minimum. La variable "k", nous permet d'avancer d'un pas oblique croissant, la variable "i" nous permet de se déplacer vers le haut pour 'croissant1' \equiv (image a) et la même variable permet de se déplacer vers la droite pour 'croissant2' \equiv (image b). Et enfin la variable "j" permet de savoir s'il y a bien une suite d'une même couleur. Voir les deux dessins au dessus.

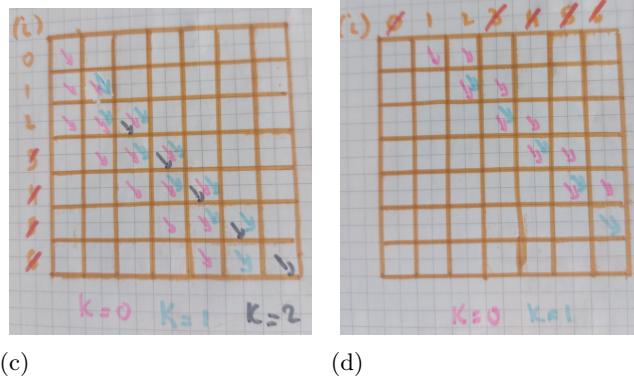
3.5 Vérification décroissante

```

1 int oblique_decroissant1(ControleurFive *cf){
2     for(int ComboAll = cf->mf->combo; ComboAll >= 5; ComboAll--){
3         for(int k = 0; k < cf->mf->avancer_tableau_horizontale; k++){
4             for(int i = 0; i < cf->mf->hauteur - 4; i++){
5                 for(int j = 1; j < cf->mf->typeSymboles + 1; j++){
6                     //Avec tous les if
7                 }
8             }
9         }
10    }
11 }
12 int oblique_decroissant2(ControleurFive *cf){
13     for(int ComboAll = cf->mf->combo; ComboAll >= 5; ComboAll--){
14         for(int i = 0; i < cf->mf->largeur - 5; i++){
15             for(int k = 0; k < cf->mf->avancer_tableau_horizontale - 1; k++){
16                 for(int j = 1; j < cf->mf->typeSymboles + 1; j++){
17                     //Avec tous les if
18                 }
19             }
20         }
21     }
22 }
```

Idée :

Cette algorithme pourrait avoir une meilleure complexité mais dans l'idée, elle, nous sert à savoir s'il y a eu un combo decroissant ou non.



(c)

(d)

Implémentation :

Tout d'abord, dans la première boucle nous initialisons la variable "ComboAll" à la plus grande valeur possible du combo et nous la décrementons jusqu'à arriver à 5 qui est le stricte minimum pour un combo. La variable "k", nous permet d'avancer d'un pas oblique décroissant, la variable "i" nous permet de se déplacer vers le bas pour 'décroissant1' \equiv (image a) et la même variable permet de se déplacer vers la droite pour 'décroissant2' \equiv (image b). Et enfin la variable "j" permet de savoir s'il y a bien une suite d'une même couleur. Voir les deux dessins au dessus.

3.6 Génération image aléatoire

```

1 int generation_image_aleatoire(ModeleFive *mf){
2     int compteur = 0;
3     for (int i = 0; i < mf->hauteur; i++){
4         for (int j = 0; j < mf->largeur; j++){
5             position_case_blanche(mf, i, j, 0);
6             if(mf->grille[i][j] == 0){
7                 compteur += 1;
8                 position_case_blanche(mf, i, j, compteur);
9             }
10        }
11    }
12    if(compteur != 0){
13        srand(time(NULL));
14        int a = rand () % compteur;
15        for (int i = 0; i < mf->hauteur; i++){
16            for (int j = 0; j < mf->largeur; j++){
17                if(mf->aleatoire[i][j] == a + 1){
18                    compteur = (i * mf->largeur) + j;
19                }
20            }
21        }
22    }
23    return compteur;
24 }
```

Idée :

Cette algorithme semble avoir une bonne complexité

Implémentation :

Tout d'abord, un compteur est incrémenté quand il trouve une case blanche et stocke ses coordonnées dans la fonction position_case_blanche. Ensuite, un nombre aléatoire va être généré et nous allons utiliser l'opérande modulo pour avoir un nombre compris entre 0 et le compteur - 1. Par après, on retourne la position d'une des cases blanches de la grille.

4 Profiling et analyse de performance du code

Pour le profiling, nous avons utilisé gprof qui nous a permis de calculer la performance de notre projet. Nous avons pu constater que certaines fonctions prenaient plus de temps que d'autres. Sur la photo ci-dessous , nous pouvons y voir que la fonction decroissant1 a pris le plus du temps vu la complexité de cette fonction.

Each sample counts as 0.01 seconds.						
	%	cumulative	self	self	total	name
time	seconds	seconds	calls	us/call	us/call	
0.00	0.02	0.02	39	\$12.92	\$12.92	oblique_decroissant1
0.00	0.02	0.02	1	0.00	0.00	position_des_jeu_principale
0.00	0.03	0.03	133735	0.00	0.00	position_case_blanche
0.00	0.03	0.03	608	0.00	0.00	change_valeur
0.00	0.01	0.00	280	0.00	0.00	charge_icouleur
0.00	0.01	0.00	280	0.00	0.00	charge_image_aleatoire
0.00	0.01	0.00	48	0.00	0.00	deplacement
0.00	0.01	0.00	41	0.00	0.00	charge_images_aleatoires
0.00	0.01	0.00	40	0.00	0.00	ajoute_images
0.00	0.01	0.00	40	0.00	0.00	ajoute_case_libre_ajoute_images_aleatoire
0.00	0.01	0.00	40	0.00	0.00	int_vers_char
0.00	0.01	0.00	39	0.00	0.00	case_blanche_compteur
0.00	0.01	0.00	39	0.00	0.00	charge_icouleur_deplacement
0.00	0.01	0.00	39	0.00	0.00	oblique_croissant1
0.00	0.01	0.00	39	0.00	0.00	oblique_croissant2
0.00	0.01	0.00	39	0.00	0.00	remplir_case
0.00	0.01	0.00	39	0.00	0.00	verification_horizontale
0.00	0.01	0.00	39	0.00	0.00	verification_verticale
0.00	0.01	0.00	4	0.00	0.00	charge_modifye_combo_horizontale
0.00	0.01	0.00	2	0.00	0.00	charge_modifye_combo_verticale
0.00	0.01	0.00	1	0.00	0.00	click_creer_five
0.00	0.00	0.00	1	0.00	0.00	creer_controleur_five
0.00	0.00	0.00	1	0.00	0.00	creer_modele_five
0.00	0.00	0.00	1	0.00	0.00	creer_vue_five

ModeleFive :

Constitue la structure complète, elle a une taille de 2492 bytes car ($13 \text{ int} * 4 \text{ bytes}$) + $2 * (20 \text{ hauteur} * 15 \text{ largeur} * 4 \text{ bytes}) + (10 * 4 \text{ bytes})$

VueFive :

Pour le niveau 0 : elle a une taille de 460 bytes car $(49 * 8 \text{ bytes}) + 2 * (3 * 8 \text{ bytes}) + 8 + (3 * 4)$

Pour le niveau 1 : elle a une taille de 716 bytes car $(81 * 8 \text{ bytes}) + 2 * (3 * 8 \text{ bytes}) + 8 + (3 * 4)$

Pour le niveau 2 : elle a une taille de 2484 bytes car $(300 * 8 \text{ bytes}) + 2 * (7 * 8 \text{ bytes}) + 8 + (7 * 4)$

ControleurFive :

Pour le niveau 0 : elle a une taille de ± 420 bytes car $(49 * 8 \text{ bytes}) + (\text{nombre caractere fichier} * 1 \text{ bytes}) + 8 \text{ bytes} + (5 * 4 \text{ bytes})$

Pour le niveau 1 : elle a une taille de ± 676 bytes car $(81 * 8 \text{ bytes}) + (\text{nombre caractere fichier} * 1 \text{ bytes}) + 8 \text{ bytes} + (5 * 4 \text{ bytes})$

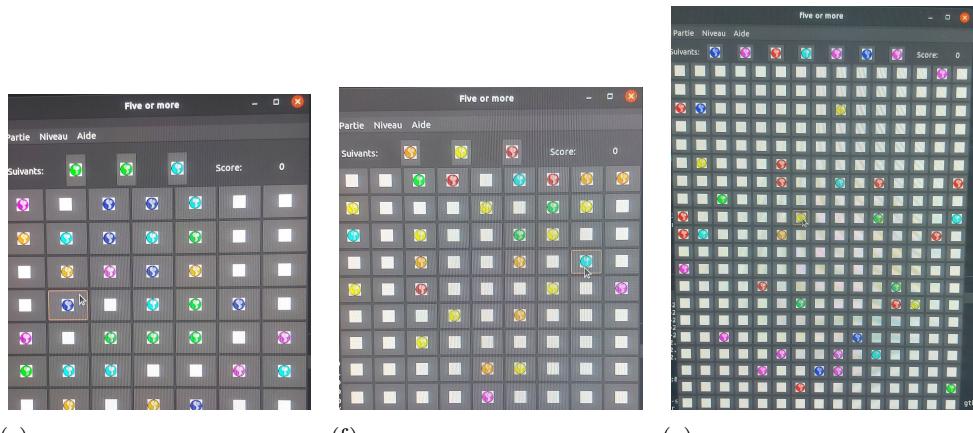
Pour le niveau 2 : elle a une taille de ± 2428 bytes car $(300 * 8 \text{ bytes}) + (\text{nombre caractere fichier} * 1 \text{ bytes}) + 8 \text{ bytes} + (5 * 4 \text{ bytes})$

5 L'interface graphique du jeu

5.1 Vue générale du jeu

Le jeu Five or more est composé de 3 niveaux :

- a) Le mode facile composé d'une grille 7x7 avec 3 boutons informations et tous ses compléments
- b) Le mode moyen composé d'une grille 9x9 avec 3 boutons informations et tous ses compléments
- c) Le mode difficile composé d'une grille 20x15 avec 7 boutons informations et tous ses compléments



(e) (f) (g)

5.2 Organisation du jeu



La barre de menu contient 3 partie :

- L'image (h) : Nouvelle partie || liste des scores || Quitter
- L'image (i) : Facile || Moyen || Difficile
- L'image (j) : Createur du jeu
- L'image (k) : Suite au click liste des scores, une fenêtre apparaît et vous pouvez consulter les 10 meilleurs scores
- L'image (l) : Suite au click Createur du jeu, une fenêtre apparaît et vous pouvez consulter des informations à propos des créateurs de ce fameux jeu.

6 La gestion du code

Pour le SCM gitlab, Lucas a créé le projet et a invité Marzouk à le rejoindre. Les débuts ont été compliqués au niveau de sa prise en main. Néanmoins, avec de la patience, nous avons su créer plusieurs branches pour chacun des patterns (Modele, Vue et Controleur) qui n'était pas la meilleure des solutions... Marzouk a incorporé tous les fichiers nécessaire pour travailler tranquillement dans chacune des branches. Quand, nous avions fini nous faisions 'git push' pour insérer les mises à jours que nous venions de faire.

7 La coopération au sein du groupe

Pour commencer, Lucas a commencé par coder l'interface graphique pour avoir un premier aperçu sur ce projet. Quand cele a été terminé, nous nous sommes partagés les taches pour savoir qui faisait quoi. De plus, pour plus de régularité, nous nous sommes données des deadlines afin de ne pas être débordé par les évènements. Globalement, le projet s'est passé dans la bonne entente.

8 Les améliorations possibles

Nous avons eu des problèmes au niveau de la gestion du temps et nous avons du faire des sacrifices pour avoir un code qui 'fonctionne'. Le plus gros des problèmes se situe au niveau de la complexité de la victoire qui aurait pu être bien meilleure. Pour en rajouter, l'algorithme de déplacement se trouve incapable de réaliser certains déplacement complexe. Une amélioration pourrait être faite au niveau du premier click de l'utilisateur. Effectivement, il n'a pas le droit de changer d'avis sauf s'il fait délibérément un déplacement impossible. Et enfin, la table de score n'est pas triée par soucis de temps ainsi que l'ajout des images aléatoire s'effectuent quand le score est incrémenté.

9 Les éléments que nous avons appris

Nous avons appris à utiliser git pour faciliter le travaille à plusieurs et sauvegarder nos données pour que l'autre membre du groupe puisse y accéder quand il le désire. En effet, se partager des fichiers sur discord n'est pas la meilleure des manières.

Le pattern MVC est une très bonne pratique a utilisé car sans lui nous aurions fait ce projet dans un seul et unique fichier ce qui serait ingérable. Gtk, une bonne interface graphique, mais pas intuitive. De même, la prise en main a été faite par la recherche sur des forums, slides du prof ainsi que par des tiers. Grace à cette interface, nous avons pu créer ce jeu qui reste tout de même assez agréable à regarder et à jouer. Personnellement, j'ai appris que développer un jeu est loin d'être aussi facile que je le pensais et qu'il est important d'y réfléchir avant de s'y attaquer la tête baissée.

9.1 Conclusion

Pour en finir, la réalisation du projet dans son ensemble a été agréable même s'il y avait quelque soucis au niveau de la gestion du temps.