INFO0947: INFO0030 : Projet 4 - Five or More

Groupe 03: Sami Ouazouz, Muhammad Qayyum

Table des matières

1 Introduction

Ce projet consistait à développer une implémentation du jeu "Five or More", où le joueur doit déplacer des boules de différentes couleurs sur une grille en créant des alignements d'au moins cinq boules de même couleur, générant ainsi du score. Notre travail a porté sur la conception d'une interface graphique utilisant la bibliothèque GTK2+, ainsi que sur l'implémentation de la logique de jeu en langage C avec divers algorithmes.

2 Architecture générale du code

Notre implémentation suit le modèle d'architecture MVC (Modèle-Vue-Contrôleur) pour structurer clairement notre code et séparer les différentes préoccupations :

- **Modèle** (`modele.h`, `modele.c`) : Contient les structures de données fondamentales du jeu (plateau, cases, scores) et les fonctions pour y accéder et les manipuler. Le modèle implémente la logique de base du jeu comme la vérification des alignements et la gestion des scores.
- **Vue** (`vue.h`, `vue.c`) : Gère tout ce qui touche à l'interface graphique, depuis la création des fenêtres et boutons jusqu'aux dialogues et à l'affichage du jeu. La vue n'a aucune connaissance de la logique de jeu elle-même.
- **Contrôleur** (`controle.h`, `controle.c`) : Fait le lien entre le modèle et la vue. Il traite les événements utilisateur (clics sur les boutons, sélections dans les menus) et met à jour le modèle et la vue en conséquence.
- **Utilitaires** (`utilis.h`, `utilis.c`) : Contient des fonctions d'assistance qui ne relèvent pas strictement d'un des trois composants principaux, comme des fonctions de gestion d'images et des générateurs de boules.

Cette architecture a permis une séparation claire des responsabilités et a facilité le développement parallèle par les membres de l'équipe.

3 Structures de données

3.1 Structures principales

Nous avons conçu plusieurs structures de données opaques pour encapsuler les éléments du jeu. Voici leurs définitions complètes :

```c
/**
 * @struct Plateau_t
 * @brief Structure principale representant le plateau de jeu
 */
typedef struct Plateau_t {
    int largeur;               // Nombre de colonnes
    int hauteur;               // Nombre de lignes
    int nb_types_symboles;     // Nombre de couleurs differentes
    int symboles_par_tour;     // Nombre de boules par tour
    int nb_cases_remplies;     // Nombre de cases occupees
    Case **cases;              // Tableau 2D des cases du plateau
    int difficulte;            // Niveau de difficulte
    Score score;               // Score actuel
    int nb_suivantes;          // Nombre de boules suivantes
    Couleur boules_suivantes[MAX_SUIVANTS];// Tableau des couleurs des boules suivantes
    GtkWidget **boutons_suivants;  // Boutons representant les boules suivantes
} Plateau;
```

Listing 1 – Structure Plateau_t

```c
/**
 * @struct Case_t
 * @brief Structure representant une case du plateau de jeu
 */
typedef struct Case_t {
    bool occupe;        // Indique si la case contient une boule
    Couleur col;        // Couleur de la boule (si presente)
    GtkWidget *image;   // Bouton GTK representant la case
} Case;
```

Listing 2 – Structure Case_t

```c
/**
 * @struct Score_t
 * @brief Structure representant un score dans le jeu
 */
typedef struct Score_t {
    char nom[MAX_NOM + 1];   // Nom du joueur
    int valeur;              // Valeur du score
} Score;
```

Listing 3 – Structure Score_t

```c
/**
 * @struct Boule_t
 * @brief Structure representant une boule
 */
typedef struct Boule_t {
    Couleur couleur;     // Couleur de la boule
    GtkWidget *image;    // Image associee a la boule
} Boule;
```

Listing 4 – Structure Boule_t

3.2 Structures auxiliaires

En plus des structures principales, nous utilisons des structures auxiliaires pour faciliter certaines opérations :

```c
/**
 * @brief Structure utilisee pour la mise a jour des cases
 * et la recherche de boutons
 */
typedef struct {
    int ligne_actuelle;       // Coordonnee ligne de la case recherchee
    int colonne_actuelle;     // Coordonnee colonne de la case recherchee
    Plateau *plateau;         // Plateau de jeu concerne
    GtkWidget *est_bouton;    // Bouton trouve
    GtkWidget *bouton_trouve;// Bouton trouve (alternative)
} MiseAJour;
```

Listing 5 – Structure MiseAJour

```c
/**
 * @brief Structure utilisee pour les recherches de widgets dans l'interface
 */
typedef struct {
    GtkWidget *est_widget;    // Widget trouve
    gboolean est_trouve;      // Indique si un widget a ete trouve
} Data;
```

Listing 6 – Structure Data

```c
   /**
    * @brief Structure utilisee pour stocker des informations dans les callbacks
    */
   typedef struct {
       int ligne_actuelle;       // Coordonnee ligne de la case courante
       int colonne_actuelle;     // Coordonnee colonne de la case courante
       Plateau *plateau;         // Plateau de jeu concerne
   } InfosCallback;
```

Listing 7 – Structure InfosCallback

4 Algorithmes particuliers

4.1 Algorithme de recherche de chemin

L'un des algorithmes les plus importants de notre jeu est la recherche de chemin entre deux cases, pour déterminer si un déplacement de boule est possible. Nous avons implémenté un algorithme de parcours en largeur (BFS) qui :

- Utilise une file (avec GQueue de GLib) pour stocker les cases à explorer
- Maintient une matrice `visited` pour marquer les cases déjà visitées
- Explore les cases adjacentes (haut, bas, gauche, droite) tant qu'elles sont libres
- S'arrête dès que la case d'arrivée est atteinte, ou quand toutes les possibilités sont épuisées

La complexité temporelle de cet algorithme est $O(n \times m)$ dans le pire des cas, où n et m sont les dimensions du plateau. Cette complexité est optimale car, dans le pire cas, il peut être nécessaire d'explorer toutes les cases du plateau.

4.2 Algorithme de vérification des alignements

Un autre algorithme clé est celui qui vérifie s'il existe des alignements d'au moins cinq boules de même couleur. Notre implémentation :

- Parcourt le plateau en vérifiant les quatre directions possibles pour chaque case : horizontale, verticale, et les deux diagonales
- Pour chaque direction, compte le nombre de boules consécutives de même couleur
- Si un alignement d'au moins cinq boules est trouvé, marque ces cases pour suppression et ajoute des points au score. Pour supprimer les cases, on a décidé d'allouer de la mémoire un tableau.
- La suppression effective est réalisée dans un second passage pour éviter des problèmes de comptage

La complexité temporelle de cet algorithme est $O(n \times m \times 4)$, soit $O(n \times m)$, car pour chaque case, nous vérifions dans les quatre directions possibles.

5 Gestion du code

Dans le cadre de ce projet, nous avons utilisé **GitLab** comme système de gestion de versions (SCM - *Source Control Management*). Cet outil nous a permis de collaborer efficacement à plusieurs sur le même dépôt de code, tout en assurant un suivi structuré des modifications apportées au fil du temps. Voici le lien vers notre dépôt : `https://gitlab.uliege.be/Muhammad.Qayyum/info0030_groupe03.git`.

Nous avons adopté une organisation basée sur trois branches principales :

— **main** : la branche principale du projet, contenant le code stable et validé ;
— **sami** : une branche dédiée au développement du binôme Sami ;
— **qayyum** : une branche dédiée au développement du binôme Muhammad.

Cette structuration nous a permis de travailler en parallèle sans interférer avec le code des autres, limitant ainsi les conflits. Chaque membre développait de nouvelles fonctionnalités ou corrections sur sa propre branche, avant de fusionner (merge) ses changements avec la branche **main** une fois les tests validés.

L'utilisation de GitLab nous a également permis de suivre l'historique des modifications, de revenir en arrière si nécessaire (par exemple avec `git checkout HEAD@{1}` ou `git reset -hard`), et de gérer efficacement les conflits lors des fusions. Le contrôle de version s'est avéré essentiel pour assurer la fiabilité et la continuité du développement tout au long du projet.

Grâce à cet outil, nous avons pu maintenir une bonne organisation du code, éviter les pertes de données, et faciliter le travail collaboratif, même en cas de modifications simultanées.

6 Coopération au sein du groupe

Notre binôme a fonctionné selon une répartition des tâches basée sur nos compétences respectives :

— Muhammad s'est principalement occupé de l'interface graphique et de l'intégration des composants (différentes structures de données), en plus de l'alogrithme permettant de vérifier les alignements.
— Quant à Sami, il s'est concentré sur toute la logique du jeu avec la génération aléatoire des boules, le chemin possible et la gestion des scores.

Nous avons organisé notre travail avec des réunions hebdomadaires pour faire le point sur l'avancement du projet. La communication s'est faite par des échanges réguliers sur Discord et via les issues et les merges sur GitLab.

Les difficultés principales que nous avons rencontrées concernaient l'intégration de nos parties respectives et la compréhension de certaines subtilités de GTK+. Nous avons résolu ces problèmes par une collaboration étroite et en consultant la documentation et les exemples vus au cours.

7 Interface graphique

Notre interface graphique est construite avec GTK+ 2.0 et s'organise de la manière suivante :

— Une fenêtre principale contenant :
  — Une barre de menu pour accéder aux fonctionnalités (nouvelle partie, niveau de difficulté, scores, etc.)
  — Une zone d'information montrant les prochaines boules et le score actuel
  — La grille de jeu, composée de boutons représentant les cases

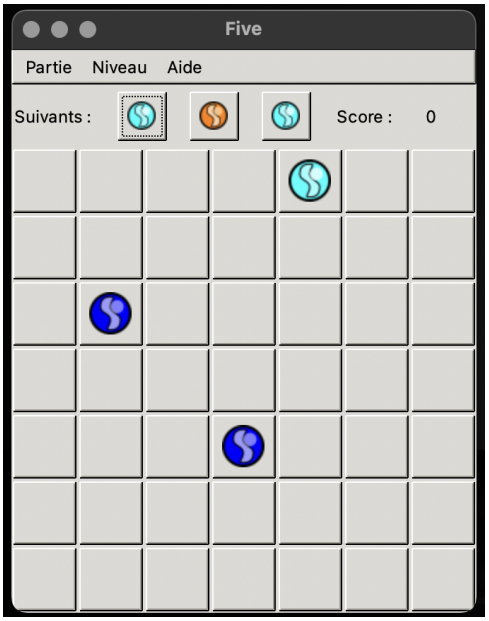

FIGURE 1 – Interface du jeu Five or More

La fenêtre est organisée avec des widgets GTK imbriqués :
— La fenêtre principale contient une VBox verticale
— Dans cette VBox, nous avons placé :
  — La barre de menu (GTK MenuBar)
  — Une HBox horizontale pour les prochaines boules et le score
  — Une Table GTK pour la grille de jeu, dont la taille varie selon le niveau de difficulté

Nous avons dû utiliser GtkTable pour cela. Chaque case du jeu est représentée par un GtkButton pouvant contenir une image de boule grâce à notre fonction :
$GtkWidget *charger\_image\_button(const\ char\ *chemin\_image)$

8 Améliorations possibles

Si nous avions disposé d'un mois supplémentaire, plusieurs améliorations auraient été envisageables :

— **Animation des déplacements** : Ajouter des animations lors du déplacement des boules pour améliorer l'expérience utilisateur.
— **Prédiction de chemin** : Afficher visuellement le chemin que la boule va emprunter lorsqu'une destination est sélectionnée.
— **Sauvegarde de partie** : Permettre d'enregistrer l'état d'une partie pour la reprendre ultérieurement avec l'ajout d'un menu.
— **Mode multijoueur** : Implémenter un mode à deux joueurs en alternance ou en compétition de score.
— **Thèmes graphiques** : Proposer différents thèmes visuels pour personnaliser l'apparence du jeu.
— **Optimisations** : Améliorer les performances de certains algorithmes, notamment la vérification des alignements qui pourrait être optimisée pour ne pas re-vérifier les cases déjà contrôlées.

9 Éléments appris

Ce projet nous a permis d'acquérir et de renforcer plusieurs compétences :

— **Programmation en C** : Approfondissement de notre maîtrise du langage C, notamment pour la gestion de la mémoire et les structures de données.
— **Programmation d'interfaces graphiques** : Découverte et pratique de la bibliothèque GTK+ pour créer des interfaces graphiques en C.
— **Architecture MVC** : Mise en pratique du pattern Modèle-Vue-Contrôleur pour structurer un projet de taille moyenne.
— **Algorithmes** : Implémentation et optimisation d'algorithmes de parcours de graphe (BFS) et de recherche de motifs.
— **Travail d'équipe** : Amélioration de nos compétences en communication, en répartition des tâches et en gestion de versions grâce à GitLab (SCM).
— **Documentation de code** : Pratique de la documentation systématique du code avec un format standardisé (Doxygen).
— **Débogage** : Développement de techniques efficaces pour identifier et résoudre les problèmes dans un code complexe.

10 Conclusion

Ce projet nous a permis de développer une version fonctionnelle du jeu Five or More, en mettant en pratique les principes de programmation en C et les concepts vus en cours.

10.1 Réalisations

Nous avons réussi à implémenter l'ensemble des fonctionnalités requises :

— Interface graphique complète avec GTK+
— Trois niveaux de difficulté avec des tailles de grille différentes
— Détection des alignements (horizontaux, verticaux et diagonaux)
— Calcul et sauvegarde des scores
— Génération aléatoire de boules de différentes couleurs
— Algorithme de recherche de chemin pour les déplacements de boules

L'architecture MVC (Modèle-Vue-Contrôleur) que nous avons adoptée nous a permis de maintenir une séparation claire entre les différents aspects du jeu, facilitant les modifications et l'évolution du code.

10.2 Difficultés rencontrées

Plusieurs défis ont émergé au cours du développement :

— **Gestion de la mémoire** : Les fuites de mémoire ont été une préoccupation constante, nécessitant une attention particulière lors de l'allocation et de la libération des ressources.
— **Algorithme de recherche de chemin** : L'implémentation d'un algorithme efficace pour déterminer si un chemin existe entre deux cases a été complexe.
— **Détection des alignements** : La vérification des alignements dans toutes les directions a nécessité une logique rigoureuse pour éviter les erreurs.
— **Interface GTK+** : L'apprentissage de la bibliothèque GTK+ a représenté une courbe d'apprentissage importante, notamment pour la gestion des événements et la mise à jour dynamique de l'interface.