Rapport sur le projet de Langage de Programmation : Arkanoïd

5 janvier 2025

Prénom	Nom	Matricule
Lucas	Verbeiren	000591223
Ethan	Van Ruyskensvelde	000589640

Table des matières

1 Introduction

Dans le cadre de ce projet, nous avons dû réaliser une version fonctionnelle avec une interface graphique du célèbre jeu `Arkanoid` en utilisant les principes de la programmation orientée objet.

Le développement a été réalisé en `C++` avec l'utilisation de la bibliothèque `Allegro` pour l'interface graphique. Le projet a été réalisé à l'aide du `Modèle-Vue-Contrôleur` (MVC) afin de séparer clairement les différentes responsabilités du code.

L'objectif principal est d'implémenter un niveau fonctionnel avec les différentes mécaniques de jeu :

— Déplacement de la raquette,
— Rebonds de la balle,
— Gestion des briques,
— Gestion des vies,
— Gestion du score,
— Niveaux multiples,
— Briques colorées,
— Bonus.

Enfin, ce projet vise également à montrer notre compréhension des concepts principaux de la programmation orientée objet et notre capacité à structurer un projet complet. À travers cette réalisation, nous cherchons à fournir un programme robuste, modulaire et bien documenté.

2 Tâches accomplies

Pour ce projet, nous avons réalisé toutes les tâches de base, c'est-à-dire :

— Rebond correct de la balle sur les différentes surfaces comme les briques, les murs et la raquette,
— Déplacement de la raquette,
— Un niveau comportant 8 lignes de 14 briques,
— Un affichage du score qui est mit à jour en gagnant 1 point par brique cassée. Si le joueur a cassé toutes les briques, un message de victoire est affiché,
— Un système de vie où le joueur a 3 vies maximum par partie. S'il les perd toutes, un message de défaite est affiché.

Nous avons également effectué ces différentes tâches additionnelles :

— Un système de niveau où chaque niveau est encodé dans un fichier '.txt', avec la possibilité d'encoder un bonus pour une brique,
— Déplacement de la raquette avec la souris,
— Des briques avec différentes couleurs et où chaque couleur fait remporté des points différents. De plus, nous sauvegardons après chaque partie le meilleur score dans un fichier 'score.txt' et le meilleur score est affiché en cours de partie. Nous avons également implémenté une touche qui permet de réinitialiser le meilleur score,
— L'ajout de briques argentées et dorées. Une brique dorée ne peut jamais être cassée et une brique argentée a besoin d'être touché deux fois par la balle pour pouvoir être cassée,

— Le bonus qui permet d'agrandir la raquette lorsque le joueur attrape la capsule bleue,
— Le bonus qui permet de ralentir la vitesse de la balle lorsque le joueur attrape la capsule orange,
— Le bonus qui permet de gagner une vie en attrapant la capsule grise.

3 Interface des différentes classes, leurs rôles et liens avec les autres classes

4 Logique du jeu

Nous allons maintenant décrire en détail ce qui se passe dans notre code à partir du moment où l'utilisateur lance le programme, et le moment où la balle touche une brique pour la première fois.

Quand le jeu est lancé, la première chose que fait le programme est d'initialiser les différentes classes nécessaires. En tout premier, il va créer l'objet *ControllerGame* qui permet de contrôler la grille du jeu et la vue. Ensuite, il créera l'objet *GameBoard* pour initialiser le modèle qui permet d'avoir l'état de la grille avec les différents systèmes qui permettent de la modifier. L'objet *Level* sera également créé. Il permet de charger les différents niveaux avec les murs, la balle, la raquette et les briques qu'il passera à la grille (GameBoard). Après, il créera aussi l'objet *DisplayGame* qui gère la partie graphique du jeu. Il initialisera `Allegro` pour pouvoir afficher une fenêtre. À chaque fois que nous avons besoin d'afficher l'état du jeu, *DisplayGame* ira demander à *GameBoard* l'état de la grille pour l'afficher.

Une fois que ces différents objets sont créés, le programme installera ce qu'il faut pour pouvoir utiliser le clavier et la souris dans le jeu. Puis, il créera une *clock* qui permet d'avoir un *tick* x fois par seconde pour mettre à jour la grille du jeu et également afficher l'état du jeu à l'écran.

Une fois fait, le programme chargera le premier niveau et démarrera ensuite la *clock*. Ensuite, 360 fois par seconde, la grille sera mise à jour en vérifiant si la balle a cogné contre quelque chose, puis le programme regardera si la partie est peut-être perdue ou peut-être gagnée, et affichera à l'écran le jeu. Si des événements arrivent dans la *queue* d'Allegro, ils seront traités avant d'afficher le jeu à l'écran.

Si la balle a touché une brique, un calcul repositionnera la balle là où elle devrait être et changera sa direction. Ensuite, le programme enlève une vie à la brique, et si sa vie passe à 0, alors elle sera supprimée de la liste des briques. Elle n'existera donc plus et ne sera plus affichée à l'écran.

5 Modèle-Vue-Contrôleur

Tout au long du développement du programme, nous avons essayé le plus possible de respecter le modèle de conception `'Modèle-Vue-Contrôleur'`. En effet, notre programme est divisé en trois parties distinctes :

— `Modèle`
— `Vue`
— `Contrôleur`

5.1 Modèle

Cette partie contient toute la logique pour la grille du jeu. Elle permet d'avoir l'état de la grille et de la modifier. Elle ne possède donc rien qui est lié à l'affichage comme Allegro. Elle permet de ne devoir uniquement gérer la partie logique du jeu, sans devoir s'occuper d'afficher en plus l'état du jeu. La classe *GameBoard* représente l'état du jeu et possède une méthode pour pouvoir être mise à jour.

5.2 Vue

La partie vue permet seulement d'afficher l'état de la grille à un moment donné. La vue va demander au modèle son état pour l'afficher à l'écran à l'aide de la bibliothèque Allegro. Elle ne peut donc en aucun cas modifier l'état du jeu, donc le modèle. Toutes les méthodes de modification du modèle dans la classe *GameBoard* sont privées, ce qui garantit que la classe *DisplayGame* n'a aucun moyen de modifier *GameBoard*. Les seules méthodes publiques de *GameBoard* sont des getters constants qui assurent qu'aucune modification du modèle n'est possible.

5.3 Contrôleur

Cette troisième partie permet de contrôler le modèle en lui demandant de faire certaines choses, ou en lui demandant de lui donner quelque chose. Le contrôleur permet aussi de dire à la vue quand afficher quelque chose. Grâce à lui, nous pouvons gérer le modèle et la vue en les séparant et les faire fonctionner tous les deux, avec la garantie que la vue ne saura pas modifier le modèle. C'est la classe *ControllerGame* qui joue ce rôle. Elle possède un *shared pointer* vers *GameBoard* pour lui demander de se mettre à jour, et un *shared pointer* vers *DisplayGame* pour lui demander d'afficher l'état du jeu.

6 Conclusion

Pour conclure ce rapport, nous dirons que ce projet nous a permis de développer nos compétences en programmation orienté objet grâce au développement d'une version fonctionnelle du jeu `Arkanoid`. L'utilisation du modèle `Modèle-Vue-Contrôleur` a été essentielle pour nous répartir les tâches entre nous deux, et pour structurer notre code et garantir une séparation claire des responsabilités entre la logique du jeu, l'affichage du jeu, et le contrôle des événements.

Le fait que ce projet soit un jeu avec une interface graphique nous a également permis de prendre du plaisir à le réaliser tout en approfondissant notre connaissance du langage `C++`.

Cette expérience nous a non seulement permis de développer un programme abouti, mais également d'affiner nos compétences en conception, en programmation et en travail d'équipe.