Dassoukhi Saleh
Fontaine Quentin
Jehanno Jules
Prud'Homme Gateau Sébastien

Projet Kassbrik

Ce projet fut pour notre groupe notre première vraie création et notre premier gros travail en équipe. Cela nous a fait prendre conscience de l'importance de la communication, de l'organisation et aussi de la difficulté de mise en place d'un projet d'une telle ampleur pour notre niveau.

Répartition et ordre de travail

Nous avons réparti, dans le groupe, les différentes classes à créer.
Nous avons commencé séparément par les classes *Balle*, *Terrain* et *Brique*. La balle et le terrain furent assez rapidement opérationnels contrairement aux briques, car nous nous sommes rendu compte qu'il fallait bien plus d'une classe pour les gérer. Pendant que certains avançaient sur les briques, un autre commença à intégrer la balle dans le terrain et gérer les collisions avec les bordures de celui-ci et le dernier commença à créer la classe *Raquette*.

La raquette et les briques étaient les plus gros problèmes :

- L'affichage et déplacement pour la raquette
- La gestion dynamique pour les briques

L'importation de la balle dans le terrain a été très simple et nous a permi de mieux comprendre comment fonctionne la classe *Window*, comment elle affiche les objets ainsi que le fonctionnement des collisions. Puis, nous avons intégré la raquette dans le terrain avec la balle. Nous avons aussi créé la classe *Joueur*. Enfin, dans le même temps, nous avons finalisé la gestion des briques, créé des options et le *Menu*, pour enfin tout assembler et régler, le plus possible, les derniers problèmes.

Le jeu est, au final, jouable avec un menu et des options fonctionnelles et contient différents niveau. Mais par manque de temps, nous n'avons pas intégré de fichier de configuration de niveau.

Choix

Pourquoi avoir commencé par la balle, les briques et le terrain ?
Nous avons fait ce choix car les briques nous faisaient penser au Jeu de la Vie, donc nous trouvions logique de s'aider des TD/TP et il était possible qu'elles demandent beaucoup plus de temps (ce fut le cas).
Pour la balle, nous pensions que gérer (pour la première fois) le déplacement d'un objet avec une classe que nous ne connaissions pas (Window) serait bien plus compliqué et surtout qu'il fallait apprendre à gérer la balle car elle interagit avec tous les autres éléments du jeu.
Enfin le terrain était un choix logique car il s'agit de la classe gérant tous les éléments du jeu.

Pour la balle : voir (Création de la classe *Balle*)

Pour le terrain, nous avions, d'abord (quand nous n'avions encore que la balle), créé une classe *Terrain* ayant en attributs des coordonnées, une largeur, une hauteur et une Window qui servait à afficher les

bordures du terrain. Cependant afin de simplifier l'utilisation et d'alléger le code, nous avons décidé de prendre directement les coordonnées et les mesures du Window en attribut, et de retirer celles de la fenêtre.

Pour les briques : voir (Création des briques)

En ce qui concerne la raquette, nous avons décidé de lui donner deux types de déplacement (rapide ou précision). Une difficulté fut donc de gérer son effacement puis son affichage de deux manières différentes lors de ses déplacements.
Un choix de gameplay fut de pouvoir (avec le bon timing) décaler la balle lorsqu'elle entre en collision avec la raquette en mouvement.

Puis, nous avons pris la décision de mettre le score en attribut de *Joueur* et de ne pas en faire une classe.

Pour les options, nous avons décidé de réutiliser celui des TP mais en enlevant son côté dynamique pour limiter à nos 3 options de base.

Nos tests et bugs

Nous n'avions pas d'ordinateur sous le système d'exploitation Linux, donc notre plus gros problème fut que nous ne pouvions pas, souvent, avancer chez nous car le logiciel X2Go (contrôle de session à distance) ne fonctionnait pas tout le temps. Nous nous sommes donc rabattu vers la fonctionnalité WSL (Windows subsystem for Linux) inclue dans windows 10 qui n'est pas complet et nous faussait certains résultats.

Pour chaque classe, nous la testions séparément dans une window, seule, afin de voir si cela marchait, et si cela n'était pas le cas, nous la réglions jusqu'à fonctionnement. Pour faire simple, chaque classe était testée seule, puis implantée dans le terrain une à une lorsque celle-ci fonctionnait.

Les plus gros tests étaient par rapport au collisions de la balle avec le terrain, les briques et la raquette qui chacun avait sa particularité.
Les briques étaient aussi problématiques pour éviter de trop en mettre, pour qu'elles s'affichent, pour éviter les chevauchements et que leur nombre s'adapte à la taille du terrain.
La raquette, nous a posé problème, car elle s'affiche comme brisée en deux, jusqu'à ce que nous le testions sur un systeme Linux et non le WSL.

Création de la classe *Balle*

Pour la classe *Balle*, on a commencé par lui donner ses cordonnées en attribut (x,y). Puis, on a créé les accesseurs en lecture et écriture, car pour le déplacement de la balle, il faut mettre à jour ses coordonnées d'où la méthode **update**. Mais, pour savoir de quel côté doit se déplacer la balle, on a dû rajouter deux autres attributs de type entier qui sont les directions (horizontale et verticale).
La balle a 2 constructeurs, un par défaut, qui maintenant n'est plus utile, (présent pour les tests, car ce fut le premier objet et classe créé du projet) et un **constructeur paramétré** pour la placer sur la raquette une fois celle-ci créée.

Une fois les briques créés, nous avons ajouté des méthodes pour savoir où est-ce que la balle percutera une brique afin de lui donner un rebond logique.

Concernant la vélocité de la balle, nous avons décidé de ne pas modifier sa vitesse au cours de la partie, car étant donné que nous utilisons la classe *Window* et lncurses, l'affichage est fait avec des cases (comme de très gros pixels), ce qui fait que lors d'un update, si la valeur de déplacement est trop importante, la balle se téléporterait, le jeu ne serait pas pas « fluide » et les collisions auraient pu être affectées.

Création des briques

Nous avons commencé à programmer les briques de manière simple, que nous avons renommé plus tard bout de briques, je vais donc employer ce terme ici.
Chaque *BoutDeBrique* devait avoir pour attributs des coordonnées (X,Y), une couleur en lien avec sa résistance, une longueur et une hauteur. Bien évidemment nous les avons dotés d'**accesseurs en lecture et en écriture**, ainsi qu'une méthode **update()** qui permet de mettre à jour la couleur de la brique en fonction de sa résistance.
Bien-sur tout ces bouts de briques sont stockés dans un tableau de bout de briques (de la classe *TableauBDB*), qui fonctionne de manière dynamique. Nous avons donc ajouté à cette classe différentes méthodes pour cela, appelées « les 4 dynamiques » : le **constructeur par copie**, l'**opérateur d'affectation** , le **destructeur**, ainsi que l'**opérateur []** pour accéder aux éléments de notre tableau. Il nous a fallu rajouter des méthodes **push_back** et **extend** pour manipuler le tableau et l'ajout d'éléments dans celui-ci.

Après mûre réflexion nous avons choisi de partir sur l'idée qu'une brique devait être composée de plusieurs petites briques (appelées bout de brique dans notre programme).
La classe *Brique* est alors composée d'un tableauBDB, de coordonnées (X,Y), d'une largeur et d'une hauteur. Cette classe a un **constructeur par défaut**, un **constructeur paramétré** ainsi que différentes méthodes comme **updateBrique**, qui comme celle pour les bouts de briques permet de modifier la couleur de chaque bout de briques composant notre brique. Nous avions également besoin premièrement de savoir si une brique était « morte » (→ sa résistance = 0) car par la suite nous avions besoin de vérifier si toutes les briques du jeu sont mortes pour terminer le jeu, pour cela nous avons créé une méthode **estMorte** qui vérifie tous les bouts de briques composant la brique en testant si ils sont tous « morts ».
A ce moment là nos briques étaient seulement de forme rectangulaire, nous avons donc intégré un paramètre à nos briques : **type**, qui permet de choisir si l'on veut une brique de **forme rectangulaire** ou de **forme triangulaire**. C'est ensuite dans les constructeurs de la brique que l'on définit une brique triangulaire ou rectangulaire.
Grâce à notre idée nous pouvons aussi créer des briques dont les bouts de brique sont de résistances différentes et ainsi avoir une brique **à moitié rouge et à moitié bleue** (par exemple). Cependant l'automatisation et la mise en place de ce principe était trop complexe, c'est pourquoi nous avons choisi de n'utiliser que des briques de même résistance.

Notre jeu ne devait bien évidemment pas avoir une seule brique (bien trop facile), c'est pourquoi nous avons intégrer la classe *PopulationBrique*, qui contient un **tableau de briques**, une taille et un entier **(level)**. Cette classe elle aussi est dynamique, elle a donc comme la précédente « les 4 dynamiques » mais surtout 2 fonctions permettant de créer « automatiquement » des « murs de briques ».
Nous avons premièrement la **fonction mur**, prenant un certain nombre de paramètre, qui crée un mur de n lignes de briques et de m colonnes de briques (n et m sont des paramètres de cette fonction), d'une résistance choisie et aux coordonnées choisies. Nous utilisons essentiellement cette fonction pour nos 2 premiers niveaux.
Ensuite nous avons la **fonction murMulticolore**, qui elle permet de créer n lignes de briques et m colonnes de briques mais qui ont une résistance aléatoire ainsi que (si on le veut) des largeurs et hauteurs aléatoires. Cette fonction est utilisée pour les 2 derniers niveaux car du fait que les résistances soient aléatoires le niveau de difficulté est augmenté.

Ces deux fonctions prennent en paramètre un string pour définir si l'on veut créer des murs de briques rectangulaires ou des murs de briques triangulaires (on peut mélanger les deux pour encore plus de difficulté).

Comme nous l'avons dit en amont, nous n'avons pas intégré de fichier de configuration, à la place nous avons directement créé les niveaux dans la classe *PopulationBrique* avec les méthodes niveau1, niveau2, niveau3 et niveau4, elles utilisent pour cela les fonctions mur et murMulticolore.

Conclusion

Nous avons un jeu jouable, la balle, la raquette, les options, le menu sont achevés. La balle peut paraître buggée, mais cela vient des méthodes de collision qui peuvent, par moment, se contredire et faire que le mouvement de la balle soit un peu différent de la normale.

Les briques sont fonctionnelles mais malheureusement pas celles ayant une formes de triangle.

Mais au final, nous avons un aperçu de notre premier projet que ce soit en équipe ou seul, nous avons réalisé un jeu proche d'un casse-brique, cependant il nous reste encore plein de choses à apprendre, à perfectionner et à maîtriser pour le futur.