L2 Informatique Général

Groupe Projet 82

Benakli Rabha William

Saad David

Année 2020-2021

Nom et Prénom	Numéro Etudiant	e-mail
Benakli Rabha William	21960601	william.benakli.4.2@gmail.com
Saad David	21955132	david2maykel@gmail.com

Sommaire

I) Description du projet

1) Diagramme de classes

II) Déroulement du projet :

1) Les parties adressées

2) Logique et implémentation

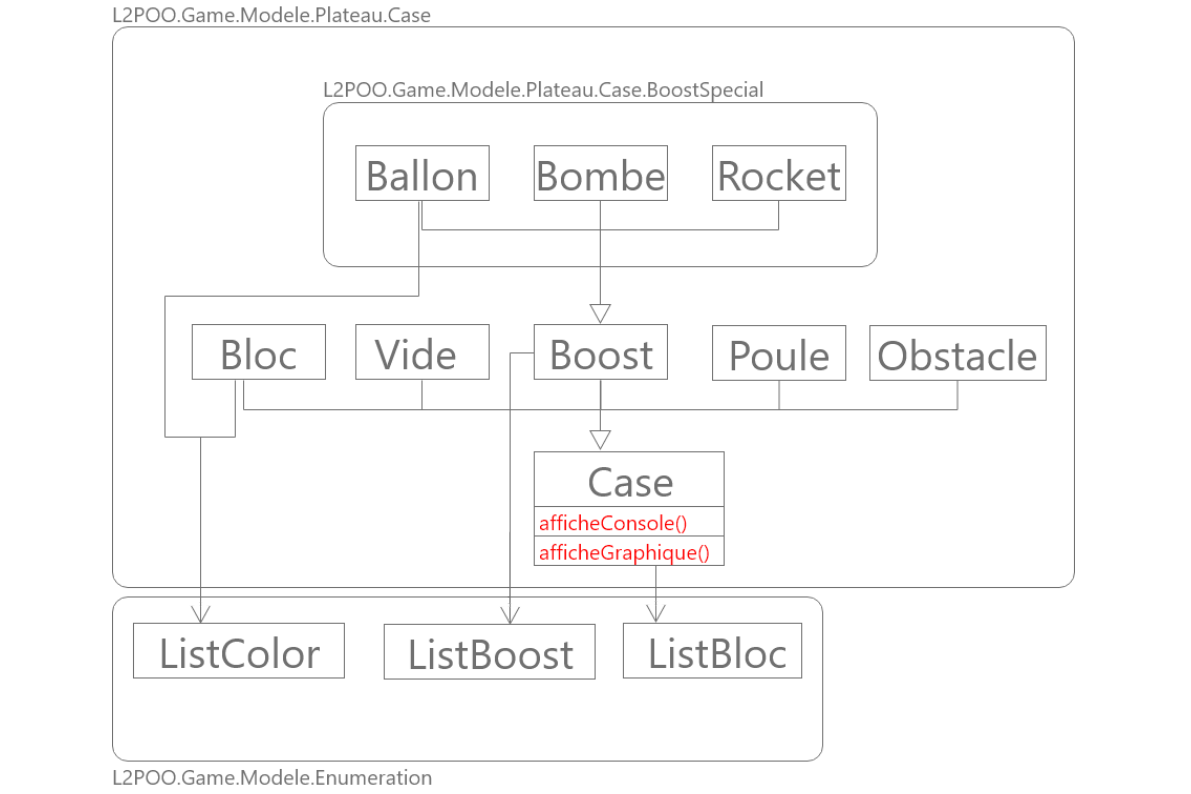
3) Les problèmes rencontrés

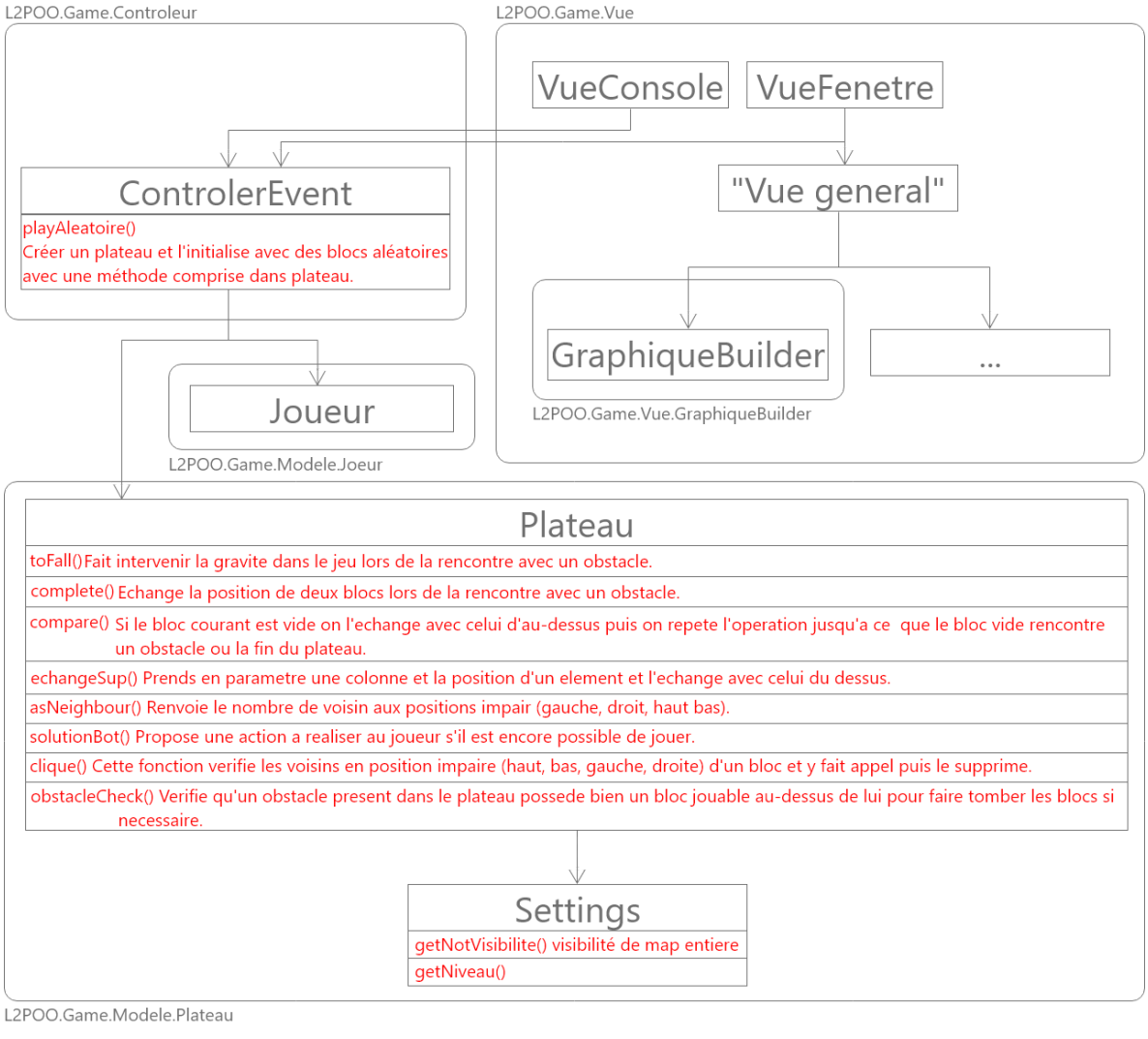
4) Les problèmes connus et les bogues

III) Pistes d'extensions du projet

1) Des améliorations possibles

I) Diagramme de classes





II) Déroulement du projet

1- Parties adressées

Dans un premier temps nous avons défini une liste de tâches à effectuer. Cette liste nous a permis de nous concentrer sur les points les plus importants du projet. Nous nous sommes basés sur nos travaux réalisés en cours pour produire un travail ayant les mêmes notions acquises lors des TP et des TD. Puis nous avons longuement réfléchi à la manière de répartir les tâches. *Qui devait faire quoi ? et pour quand ?* Cependant nous sommes arrivés à la conclusion qu'il valait mieux pour la partie principale faire la structure du projet ensemble. Les seules parties qui ont été réalisées séparément sont les deux types de vue mis à disposition de l'utilisateur:

-la vue console réalisé par David

-le mode fenêtré réalisé par William

Les parties complexes telles que les algorithmes de détection de victoire, de gravité, de déplacements, de vérification de cas etc... ont été réfléchis et produits à deux. Cela a eu un gros impact sur notre vitesse dans l'avancement du projet, cependant cela a permis de limiter les erreurs dans la conception et la compréhension globale du sujet..

2- Logique et implémentation

Le choix de la structure d'un projet est primordial pour le lancement et la maintenance de celui-ci. C'est pourquoi nous avons mis en place le design pattern MVC (Modèle-Vue-Contrôleur) qui permet d'avoir une conception claire et efficace.

Pour la conception des Classes il fallait réfléchir à une manière simple et plus adaptée pour la mise en relation des objets : Plateau, Bloc, Joueur etc.... C'est dans cette optique que nous avons décidé de mettre en relation certains types d'objets.

- Par exemple pour le type de bloc nous n'avons pas fait de "type" représenté par un int ou char, mais nous avons décidé d'utiliser des énumérations *(ListColor, ListObject, ListBoost)* d'objets. Cela a permis d'avoir un code plus lisible et compréhensible.

-Par ailleurs, la classe Settings que nous avons nous permet de limiter le nombre d'arguments que peut prendre un objet de type Plateau.

Enfin, le choix d'avoir de l'héritage était plus que nécessaire pour la réalisation de ce projet. Il nous fallait des liens forts et bien pensés entre chaque objet pour limiter un maximum les répétitions. Ainsi, nous avons donc fait une class abstract Case qui représenterait l'ensemble des Blocs/Obstacles/Animaux/Boost présent sur le plateau. Cela nous a permis de limiter les appels récurrents à différents types d'objets et de faire directement appel à Case pour appeler l'ensemble des objets. en utilisant le polymorphisme.

3- Les problèmes rencontrés

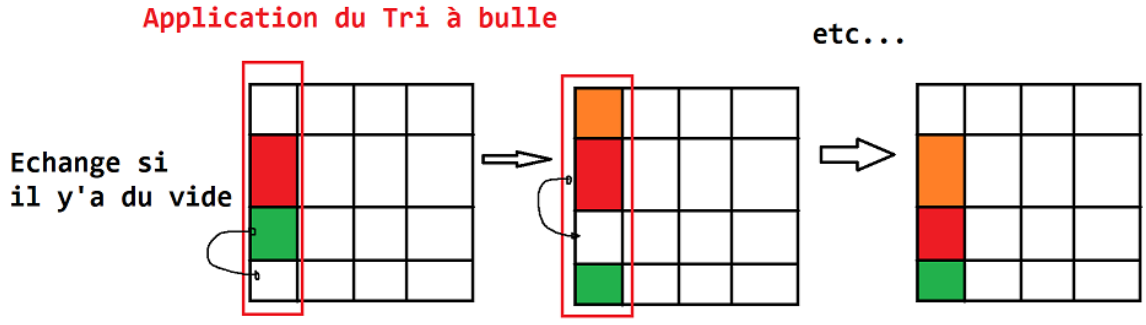
Conception des algorithmes: Pour la réalisation de nos algorithmes il a été plus que nécessaire de les conceptualiser dans un premier temps sur papier puis une fois les idées en place de les retranscrire sous forme de code.

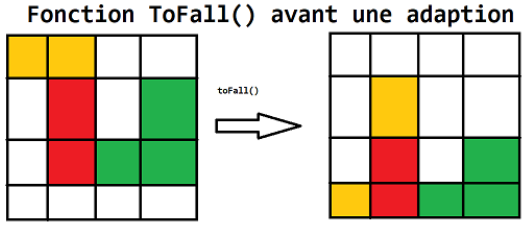
Des algorithmes essentiels pour le jeu:

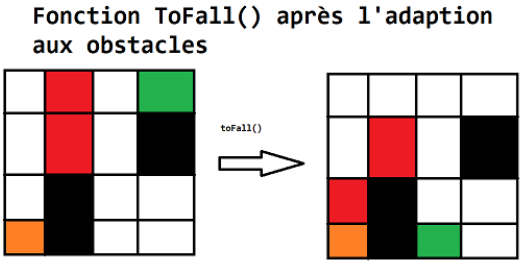
Choix du système de tri avec public void toFall(): (FaireTomber) initialement cette fonction fait office de gravité sur le plateau. Une question nous est venue assez rapidement : Comment faire un algorithme optimal et simple à la compréhension ?

Le choix d'un système de tri pour mettre en place le système de gravité dans notre jeu nous a semblé être une solution viable étant donné que le plateau de jeu est constitué de deux tableaux dynamiques (arrays liste).

Concernant le choix du type de tri, nous avons longuement cherché puis avons fini par décider d'implémenter le tri à bulles qui est un algorithme que nous avons eu l'occasion de rencontrer durant nos séances d'EA3. En effet, notre choix s'est tourné vers cette méthode étant donné que cet algorithme est plus facile à implémenter avec les obstacles que d'autres algorithmes comme le tri par insertion bien qu'il possède une complexité plus élevée de l'ordre de O(n^2) contre O(nlog(n)).







Gestion des obstacles: La gestion des obstacles dans le jeu nous a contraints à revoir et créer de nouveaux algorithmes comme public void obstacleFall(Point p) pour prendre en compte le déplacement des blocs de couleurs et les pouvoirs qui vont à un moment ou un autre devoir interagir avec eux. Par ailleurs, l'un des algorithmes ayant le plus était impacté par ce changement est public void toFall()

La réalisation d'un bouton redo: Lors de parties il est récurrent qu'un joueur réalise une action par inadvertance. Ainsi, nous avons décidé d'implémenter un bouton permettant de rejouer un coup lors de la partie.

Pour l'implémenter, nous avons décidé de faire appel à undoManager, mais après plusieurs tentatives infructueuses nous avons décidé de nous pencher sur une méthode plus classique qui est de cloner le plateau et de le stocker de manière temporaire le temps que le joueur joue un autre coup.

L'utilisation de la librairie Swing ne nous a pas simplifié la tâche: Beaucoup trop restreinte, il a fallu refaire un "GraphiqueBuilder" avec nos compétences. A l'intérieur, nous avons "extends" des classes de la librairie Swing pour les adapter et avoir un visuel plus agréable. Cela a demandé beaucoup d'efforts car nous avons dû apprendre et saisir beaucoup d'aspects de la librairie Swing, dont le paintComponent() avec lequel nous avions beaucoup de mal.

Toujours pour rester dans le thème visuel. La conception d'un multi-panel sur une seule page nous a beaucoup ralenti, nous avons essayé plusieurs issues dont le CardLayout que nous avons fini par ne pas retenir car trop peu documenté. Finalement nous avons opté pour un JPanel principal qui fait la transition entre chaque JPanel .

Pour dynamiser un peu l'aspect visuel, une idée d'animer le fond nous a paru évidente. Cependant en faisant pas mal de recherches nous avons pu constater que Swing limite ce type d'interaction. Ainsi, nous avons dû mettre cette idée de côté.

4- Les problèmes connus

Suite à de nombreux tests et correctifs nous n'avons heureusement trouvé aucun problème dans la dernière version . Cependant il est possible qu'il existe des bogues d'affichages ou de rafraîchissement.

III) Pistes d'extensions du projet

1- Améliorations possibles

Pour améliorer notre projet, nous avons pensé à implémenter plusieurs fonctionnalités:

-L'ajout de pouvoirs (en plus des boosts) comme par exemple le marteau qui pourrait permettre de détruire n'importe quelle case présente sur le plateau.

-Par ailleurs, pour améliorer le confort d'utilisation de notre jeu, il serait intéressant de se pencher sur la mise en place d'animation et de sound fx qui se déclencheront dès lors que le joueur réalise une action. De plus, la mise en place d'un affichage responsive (qui pourrait s'adapter à tout type d'écran)pourrait contribuer grandement au confort de l'utilisateur.