I - Parties traitées

• L'implémentation du fonctionnement du jeu s'est faite dans Board (toutes les méthodes permettant la modification d'un plateau y sont ajoutées au fur et à mesure, selon nos besoins).

• Un joueur est représenté par un objet de la classe Player. Le joueur robot est tout simplement une invocation à la méthode Player.autoplay(). Pour reprendre ou sauvegarder une partie, ce sera principalement Player qui sera concerné, ainsi que ses attributs (plateaux de jeu, bonus, nombre de cœurs).

• Plusieurs classes ont été créées pour la gestion des niveaux. Tout d'abord, la classe Level représente un niveau simple et elle permet l'accès ou non à son plateau de jeu. Ensuite, nous avons la classe Levels et sa classe interne AddLevels. Levels est une classe qui contient tous les niveaux d'un joueur (débloqués ou non), dans un tableau de Level. Sa classe interne va générer les niveaux (que nous avons codés) et les mettre dans le tableau en question. Levels permettra donc la gestion de l'avancée du joueur, selon ses niveaux débloqués, à travers ce tableau.

• Les bonus sont des coups spéciaux accordés aux joueurs et permis par l'utilisation d'outils. Ils sont implémentés par une classe générale Bonus, qui contient des sous-classes, représentant chacune un outil différent (comme brise-bloc ou brise-colonne).

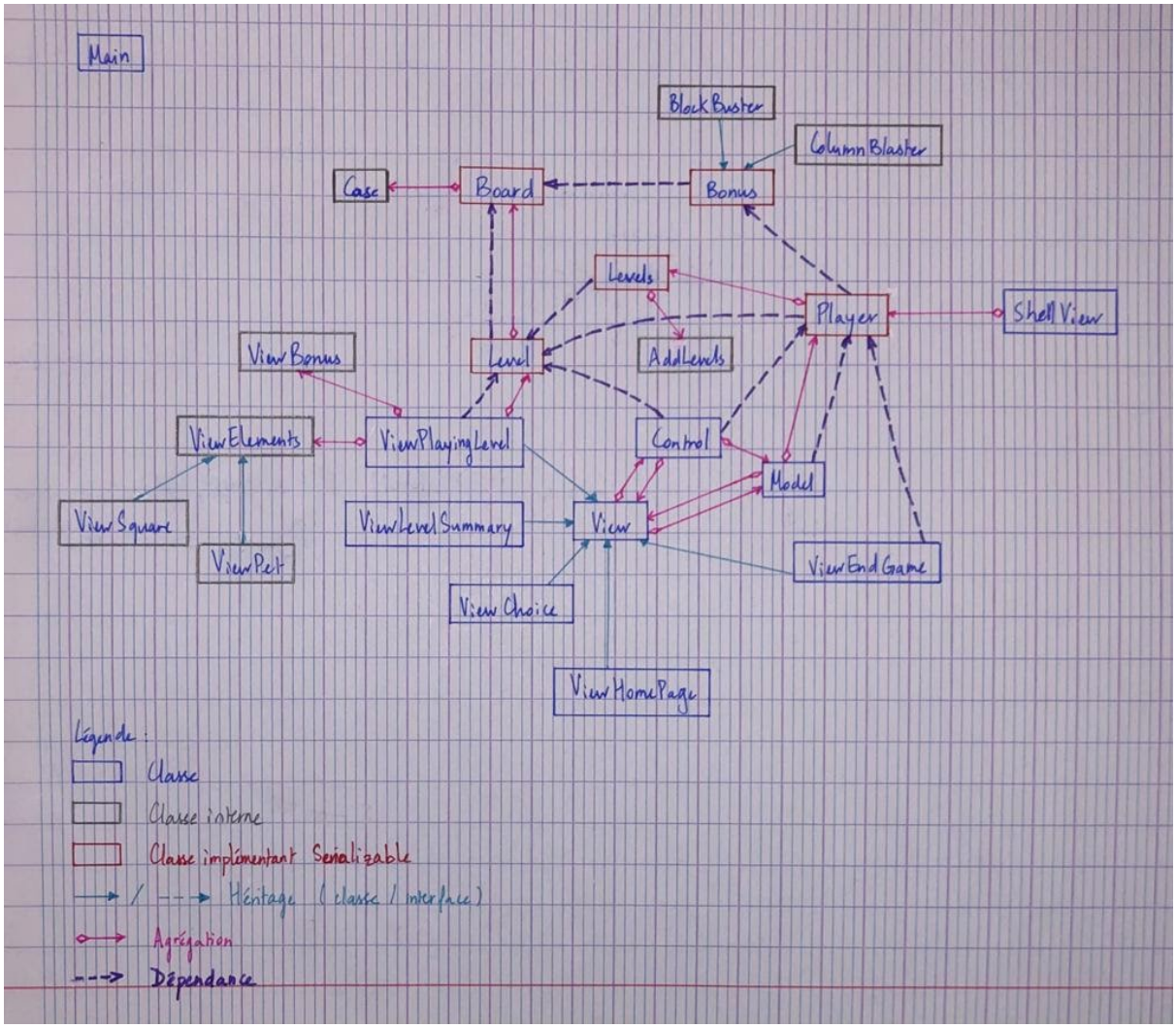
• La partie textuelle du jeu débute dans la classe ShellView. C'est en quelque sorte la page d'accueil du jeu, permettant de choisir le joueur (nouveau/ancien joueur). Une fois le joueur sélectionné, la partie est lancée et le jeu fonctionnera avec Player.playWithShell().

• La partie graphique du jeu est implémentée par essentiellement trois classes : Model, View et Control (structure en MVC). Model contient les éléments manipulés, à savoir la vue et le joueur. Control effectue toutes les modifications enclenchées par les actions sur la vue, c'est donc cette classe qui va modifier la vue et le modèle, et exclusivement elle. Enfin, View gère bien entendu l'affichage des composants graphiques.

• La vue se divise en cinq sous-classes, chacune ayant un rôle différent. La première vue qui se lance est ViewChoice, qui permettra de choisir entre interface graphique et textuelle. Une fois le mode graphique choisi, ViewHomePage, la page d'accueil, nous propose de continuer une partie, d'en commencer une nouvelle ou de lire les règles du jeu. Dans les deux premiers cas, la fenêtre de ViewLevelSummary, le sommaire des niveaux, apparaît pour permettre la sélection des niveaux, le retour à l'accueil ou la sauvegarde de la partie. ViewPlayingLevel, quant à lui, s'occupe de l'affichage d'un niveau du jeu. Le bas de la fenêtre contient les différentes fonctionnalités permises au joueur pendant la partie, à savoir faire une pause (retourner au sommaire), recommencer la partie, activer ou arrêter le jeu automatique et les différents bonus. Le reste de la fenêtre donne sur l'affichage du plateau, un affichage en réalité découpé en ViewElements (plus précisément en ViewSquare et ViewPet). Enfin, selon la victoire ou la défaite et même le niveau joué et le nombre de cœurs restant, une fenêtre de fin de jeu, ViewEndGame, est affichée et personnalisée selon les conditions citées.

Concernant la répartition du travail, à cause notamment des problèmes cités plus tard et aussi pour la maîtrise générale de tout le programme, il n'y en a pas tellement eu. En effet, nous avons préféré ne pas se donner de tâches fixées à chacun pour que nous puissions nous adapter au code écrit par l'autre, et aussi dans le cas où d'éventuelles fonctionnalités imprévues viennent s'ajouter dans le programme. Ainsi, nous avons commencé l'implémentation du fonctionnement du jeu ensemble, puis chacun est parti coder les parties qu'il voulait (en prévenant l'autre au préalable, puis en faisant un petit rapport après les éventuels changements).

II - Représentation graphique des classes



III - Problèmes rencontrés

• Problèmes techniques :

En rapport avec les conditions sanitaires, l'impossibilité de tester notre production sur les machines du script, alors que ce sont les machines sur lesquelles il nous est demandé d'être certain du fonctionnement de notre code, a été gênante. En effet, bien que nous puissions tester sur au moins deux machines, nous n'obtenions parfois pas les mêmes affichages (caractères spéciaux illisibles chez l'un, mais pas chez l'autre) : comment savoir quelles difficultés pourraient être présentes lors de la présentation alors qu'une partie de nos machines n'a pas réussi à faire fonctionner le programme à partir d'un terminal unix tandis que ce fut possible sur l'autre partie ?

Pour la même raison, le partage du travail ainsi que la communication, sans être impossibles, ont été rendues difficiles par le peu de temps de présence en face à face.

Enfin, la méconnaissance de git a réduit notre efficacité.

• Problèmes avec l'affichage graphique :

La mise en page lorsque le plateau de jeu dépasse la fenêtre a entraîné l'introduction d'attributs supplémentaires, afin de permettre une implémentation d'affichage depuis le haut ou depuis le bas

de la fenêtre, selon le nombre de lignes à afficher.

La gestion de l'affichage du plateau (notamment de la couleur et de l'emplacement des cases) après chaque mouvement n'a pu être faite que par l'actualisation totale du plateau à chaque tour : cette solution ne nous semble pas optimale.

Un bloc supprimé peut parfois demander plusieurs réaménagements du plateau (c'est-à-dire plusieurs appels à la méthode Control.stepByStep() pour l'animation), requérant ainsi une bonne gestion avec les timers et la méthode Board.needACall().

En lien avec le point précédent, l'utilisation du Timer de Swing ne nous a pas semblé aisé à comprendre. Cette utilisation nous a néanmoins paru indispensable afin de laisser du temps entre chaque tâche, puisque les méthodes wait() et sleep() bloquaient l'affichage du plateau. Cela a entraîné de nombreuses réécritures de la méthode Control.stepByStep() afin d'obtenir un résultat correct.

L'affichage de la fenêtre de victoire ou de défaite a, au début, été défaillant lorsque plusieurs actions (c'est-à-dire plusieurs suppressions de cases) avaient lieu en même temps (ou du moins à intervalle trop court) : il y avait plusieurs appels aux méthodes Control.win() ou Control.fail() et donc plusieurs déblocages de niveaux, pertes de cœurs et fenêtres qui apparaissent. Ce problème a entraîné l'obligation de vérifier la correspondance du niveau et du nombre de cœurs pour effectuer ces appels.

Afin d'obtenir effectivement l'affichage attendu, différents layouts ont dû être choisis et combinés, malgré une certaine méconnaissance de ceux-ci.

Il nous a été impossible d'introduire un affichage d'images aléatoire pour les blocs d'animaux avec des liens : lorsque nous tentions cela (avec un tableau de liens et un nombre aléatoire qui sélectionne un lien par exemple), les images changeaient tout le temps dans le plateau de jeu, sans raison identifiée. Cela nous a conduit à choisir définitivement les images par niveau.

Après avoir introduit les bonus (coups spéciaux), il était difficile pour nous (et encore plus pour un joueur lambda) de savoir quand est-ce qu'un bonus était activé ou non. Nous avons donc fait en sorte que les bonus sélectionnés changent de couleur, au niveau du bouton. Le bouton reprendra ensuite sa couleur initiale, après avoir été désélectionné (par utilisation ou annulation).

• Problèmes avec l'affichage textuel :

Lors d'une partie en affichage textuel, l'ordre dans le choix des coordonnées a paru confus : était-ce l'abscisse ou l'ordonnée qui était symbolisée par le premier chiffre entré ? Finalement, nous avons fait le choix de représenter les abscisses par des lettres de l'alphabet et les ordonnées par des nombres.

Nous avons cherché à produire un affichage aussi simple que possible, afin de ne pas envahir le joueur de plateaux de jeu et le perdre dans ses actions possibles, alors que les possibilités d'affichages sont particulièrement réduites. Ainsi, certaines actions n'ont pas été implémentées alors que celles-ci sont possibles dans l'interface graphique (pas d'animation du changement des cases du tableau de jeu après un tour, impossibilité de choisir ses niveaux, ...)

• Problèmes généraux :

Dès le commencement du projet, le problème des coordonnées dans un tableau bidimensionnel s'est posé. Nous avons par la suite convenu de fixer, pour un tableau t, t[colonne][ligne].

L'écriture des niveaux ne peut se faire qu'« à la main », sans insertion utile de boucle, ce qui a entraîné (et entraînerait lors de chaque ajout de niveau) une perte de temps dommageable.

Un décalage d'indice entre le numéro effectif attribué au niveau et sa place dans le tableau

Levels.levels existe : le niveau n est positionné à l'indice n-1. Or, l'oubli de cela peut être très rapide et, bien entendu, occasionner des pertes de temps facilement évitables. Le décalage des indices aurait été possible en laissant la case d'indice 0 vide, mais cela semblait contre-intuitif.

L'ajout d'une fonctionnalité (telle que les bonus) dans une classe implique beaucoup d'ajouts et de changements dans plusieurs classes, et donc de possibles oublis nuisant au bon déroulement de la production.

Divers autres problèmes liés à une implémentation incomplète (tous les cas ne sont pas traités) ou défaillante de certaines méthodes ont été rencontrés et repérés par différents affichages, pour mettre en lumière l'origine du problème et y remédier.

IV - Pistes d'extensions

• Editeur de niveau (créer ses propres niveaux)

Dans ce mode, le joueur pourrait créer son propre niveau puis y jouer. Pour cela, en mode graphique, nous pourrions proposer des icônes cliquables que le joueur pourrait disposer dans une grille. En mode textuel, cela pourrait prendre la forme d'une demande au joueur du type de case (case de couleur, animal ou case interdite) et, le cas échéant, de la couleur qu'il souhaite introduire à certaines coordonnées. Cela créerait néanmoins une lourdeur dans l'interface textuelle que nous avons jusqu'ici souhaité éviter.

Comment procéder toutefois lorsque le joueur souhaite créer un niveau qui dépasserait la taille de la fenêtre sur l'interface graphique ? Il faudra ensuite bien gérer la création d'un tel tableau (récupérer le nombre de lignes, les coordonnées de chaque couleur, ...).

• Personnalisation (et amélioration) de l'interface graphique

L'interface graphique, telle qu'actuellement présentée, est plutôt déplaisante et peu attirante. Nous pourrions commencer par modifier le fond et le style des boutons (d'abord de manière simple, sans apport de ressources, puis de façon plus élaborée ensuite). Si plusieurs styles sont rendus disponibles, le joueur pourrait choisir celui qu'il souhaite. Les styles pourraient concerner aussi bien les fonds ou les boutons que les cases ou, si cette fonctionnalité était implémentée, l'animation sur le plateau de jeu des bonus.

• Sauvegarde de la partie depuis le terminal, à n'importe quel moment

Cette fonctionnalité non implémentée ici pour garder l'interface légère pourrait l'être plutôt aisément en ajoutant une condition à chaque scanner : une méthode permettant la serialization serait utilisée lorsqu'un scanner recevrait, par exemple, « SAVE ».

• Possibilité d'avoir plusieurs sauvegardes sur la même machine

Plusieurs personnes pouvant utiliser la même machine, il pourrait être très utile d'introduire la présence de plusieurs sauvegardes. Nous pourrions, pour cela, utiliser simplement des numéros de sauvegardes différents ou, dans une version plus élaborée, associer un pseudonyme à chaque joueur.

• Annuler le dernier coup

L'annulation du dernier coup passerait par un champ conservant l'état du tableau, et un autre retenant le score lui étant associé, avant le dernier coup joué. Si l'on souhaite pouvoir annuler plusieurs coups, il faudrait implémenter des piles (de tableaux et d'entiers pour les scores), ce qui est faisable à l'aide de la classe *Stack*.

Pour effectuer l'annulation, nous pourrions simplement ajouter un bouton ou implémenter un raccourci clavier tel que l'habituel Ctrl-Z grâce à la classe *KeyStroke* de Swing. Il semble néanmoins plus évident de n'utiliser la seconde implémentation qu'en complément de la première, afin que la fonctionnalité ajoutée reste facilement accessible.

• Aide

Il serait possible de montrer la zone (en changeant la couleur des cases ou de leurs bordures en mode graphique, ou en affichant les coordonnées d'une case appartenant à cette zone en mode textuel) où le plus de blocs peuvent être supprimés, ce qui est déjà fait avec le mode de jeu automatique. Toutefois, ceci ne garantit pas une aide effectivement utile pour terminer le niveau : comment garantir que le plateau sera toujours terminable après le coup suggéré par l'aide ? Par ailleurs, la même remarque pourrait être faite concernant le jeu automatique ; dans tous les cas, une solution serait de proposer le coup qui serait fait lors du jeu automatique.

Il peut être envisagé de faire apparaître cette aide lors d'une demande expresse du joueur ou après une certaine période d'inactivité, c'est-à-dire sans coup joué ou sans mouvement de la souris.

• Restauration automatique des cœurs

Dans notre version du jeu, il est impossible de gagner des cœurs, seulement possible d'en perdre (en perdant un niveau). Dans la version graphique, pour le permettre, nous aurions pu rajouter un timer dans la classe Control, qui, au bout d'une durée décidée, ajoute un cœur au joueur du Model. Dans ce cas, nous pouvions aussi créer un affichage pour le temps d'attente restant, dans le sommaire des niveaux (dans le timer, chaque seconde écoulée va rafraîchir la page du sommaire et changer l'affichage du temps restant). Pour l'affichage textuel, nous n'afficherons peut-être pas ce temps restant, mais nous pouvions déjà ajouter un affichage du nombre de cœurs, et ensuite ajouter un timer (dans la classe Player) qui sera seulement appelé quand le jeu textuel sera lancé.