NODIN Aurélie 21960201
MIKIA Benidy 21958639
Groupe 52

2020/2021
L2 informatique
POO

Animal Crossing Rescue

Nous avons choisi d'expliquer notre projet par rapport à la chronologie de notre codage. Nous allons donc parler des idées survenues, de la conception de ces idées, des problèmes rencontrés et puis des solutions apportés.

L'idée

Pour commencer, nous avions fait un fichier écrit résumant toutes les classes requises :

• Une classe animaux qui fabrique des objets (chien, chat, cochon, oiseau etc...):

-chaque animal est associé à un nombre de point que le joueur obtient si l'animal a atteint le bas du plateau.
-un animal est associé à une case (dans laquelle il est).

• Une classe cube :

-les cubes ont une couleur et un nombre de point associés.
-les cubes peuvent se déplacer vers le bas (mais également vers la gauche lorsqu'il y a un espace).

• Une classe case:

-chaque case aura un attribut booléen qui vérifie si la case est vide ou non.
-une méthode qui permet d'enlever un cube d'une case.

• Une classe plateau (chaque niveau a un plateau différent avec un forme différente):

-on pourra modéliser chaque niveau par un plateau quadrillé (tableau de cubes)
-toutes les cases sont remplies aléatoirement, les animaux se trouvent un peu partout dans le plateau.
-une méthode qui vérifie pour une case donnée qui contient un cube (on fera une boucle pour vérifier chaque case) que la case qui se trouve en dessous n'est pas vide .Si c'est le cas elle abaisse le cube jusqu'à ce qu'il y ait une case contenant un cube ou le bas du plateau.
cette fonction doit aussi déplacer le cube vers la gauche si la gauche est vide.
REMARQUE !!! cette méthode s'applique aussi aux animaux.
-une méthode qui permet de vérifier s'il y a un cube de la même couleur à coté du cube sélectionné(et si c'est le cas il faut vérifier récursivement ces cases là aussi) , dans le jeu actuel il faut au minimum 2 cases de même couleur pour les faire disparaître.
- un attribut compteur et un attribut nbAnimaux qui compte le nb d'animaux présent dans le plateau actuel
-une méthode qui vérifie si un animal est en bas du plateau. Si c'est le cas ont vide la

case et on fait compteur++.
- une méthode booléen qui renvoie true si compteur = nbAnimaux (tout les animaux ont atteint le bas du plateau).

Contenu , Cube , Animal, Case

Ceci était notre idée pour une base non graphique uniquement terminale. Au début de la conception du code, nous avons commencé une nouvelle classe dont nous n'avions pas pensé au premier abord, la classe **Contenu**, nous l'avons incluse dans un package p*lateauLogistique* . Ce package allait inclure tout ce qui est indépendant du joueur, toute la logistique pour faire fonctionner le programme correctement. Pour en revenir à Contenu, nous avions étudié le vrai jeu Pet Rescue saga et avions remarqué que les points n'étaient pas dépendant de la couleur des cubes ou de la sorte d'animal sauvé, mais seulement du nombre de cube sélectionnés (le calcul étant le suivant : (NombreCubeSélectionnné*NombreCubeSélectionné)*10) et si un animal était sur la dernière ligne (+10 000 points). De ce fait, la classe Contenu contient un String *name*, qui correspond au nom du cube/animal et un booléen *VaEtreSupprimé* pour savoir par la suite quelles cases vider. Cette classe contient également des méthodes getter et setter et une méthode ToString() pour l'affichage sur le terminal. C'est après cette classe que l'on a créer deux autres classes, filles de Contenu, **Cube** et **Animal.** Cube et Animal hérite donc de Contenu. Elles sont très simple, le constructeur prend en paramètre un nom seulement.

Ensuite viens la conception du plateau, pour cela nous avons créé une classe **Case**. Comme prévu, nous avons un booléen *estVide* pour des tests plus tard. Nous avons aussi ajouté un attribut Contenu *content*, qui n'était pas prévu à la base puisque la classe Contenu ne l'était pas non plus. Des getters et des setters et des fonctions basiques comme remplirCase() et viderCase() ont également été ajouté. A la base, viderCase retournait un Contenu null, nous expliquerons par la suite pourquoi nous avons changé cela.

Plateau (et Test_Projet)

Nous sommes ensuite rentrées dans le vif du sujet avec une classe très importante, **Plateau.** Au départ, la classe contenait ces attributs là :

Case[][] *niveau*; private int *longueur*,*largeur*; protected int *animaux*;
private int *point* = 0; private int *compteurAnimal* = 0;

Nous avions fait un constructeur, celui que l'on voit dans le projet, public Plateau(int longueur, int largeur,int ani), ce constructeur construit un plateau de Case vide d'une longueur+2 et d'une largeur+2 pour avoir des bords et éviter l'erreur OutOfBounds. Le plateau visible commence au indice (1,1). Puis nous avions fait une méthode qui remplissait les cases avec des cubes, de façon aléatoire (grâce à Random) Elle avait en paramètre un nombre d'animaux à placer. Ceux-ci était placé en faisant attention de ne pas les mettre aux bords et sur la dernière ligne. Déjà, ici nous avions un premier problème que nous n'avions pas remarqué au départ, les animaux étaient placé n'importe où et non pas sur la première ligne du plateau visible. Un autre problème c'est posé ensuite, des niveaux totalement insolubles. Mais pour nous en rendre compte, nous avons du faire des méthodes pour jouer sous forme d'affichage terminal.

Nous avons commencé par affichePlateau(), cette méthode appelle la méthode ToString() de Case pour afficher le contenu de chaque case correctement. Puis, une fonction qui nous a donné du fil à retordre, selectionCubeCouleur(int x, int y), nous savions qu'elle devait être récursive. Au début, nous vérifiions seulement les noms des cases adjacentes, également si elles n'étaient pas vide et si le booléen *vaEtreSupprimé* n'était pas déjà à « true ».Malheureusement, cela nous mettait un OutOfBoundsException , puis nous avons réglé cela en ajoutant les conditions de x et y inférieur à

la longueur du tableau -2. Nous avons aussi fait des vérifications pour ne pas supprimer un animal (nous n'avions pas encore les Obstacles) .

Puis nous utilisons le booléen *vaEtreSupprimé* pour compter les points à ajouter (avec le système spécifié plus haut) avec la fonction comptePoints() , c'était une fonction plutôt simple a réaliser, nous avons décidé que cette fonction servirait à ne pas supprimer le cube si il est sélectionné seul.

SupprimerCubeCouleur() était encore plus facile, il suffit d'appeler viderCase() de la classe Case pour chaque case qui avait *vaEtreSupprimé* à true, de ne rien faire sinon.

Nous avions des fonctions selectionnerCaseX() et selectionnerCaseY() contenant chacune un Scanner pour demander au joueur sur le terminal quelles coordonnées il veut jouer.

AbaisseContenu() fut également plutôt simple à faire, on évalue si la case d'en dessous est vide et on appelle récursivement la fonction. Par contre déplaceGauche() était très difficile à faire et ne fonctionnait que partiellement.Nous avons réussi à l'arranger qu'à la toute fin du projet.Nous avons pris beaucoup de temps à la faire, voyant qu'elle ne fonctionnait pas, nous sommes passé à la suite, c'était une bonne idée de ne pas resté coincé dessus indéfiniment. Nous l'expliquerons donc plus tard, à la fin du projet.(Avec firstRempli()).

AnimalSauve() a été faite rapidement, nous évaluons si la dernière ligne contient un animal, si oui, on incrémente le compteur d'animal, on ajoute 10 000 points et on modifie le booléen *vaEtreSuprimé* à true.
PartieGagne() , comme son nom l'indique, vérifie qu'une partie est gagné.Celle-ci sera très très utile pour la déserialisation plus tard.
DonneEtoile() et afficheEtoile() n'était pas présente au début, nous les expliquerons donc plus tard.
Des getters et setters viennent compléter la liste des fonctions.

La classe Plateau était finie pour le moment, nous avions toutes les fonctions pour avoir un début de jeu, alors nous avons créé une classe éphémère **Test_Projet.java** qui contenait le main, puis nous avions créé chaque objet à la main et rempli le tableau à la main pour pouvoir interagir avec les Cubes et animaux. C'est grâce à ces premiers tests qu'on a pu corrigé l'erreur OutOfBounds de selectionCubeCouleur(). Puis nous avons généré un plateau aléatoire et là, nous avions remarqué que certains niveaux auraient été irrésolvables. Pour pallier à cela, nous avons repris les quatre premiers niveaux de Pet Rescue Saga.

Niveau1 , Niveau2, Niveau3, Niveau4

Nous avons donc supprimé la fonction de remplissage automatique du plateau et nous avons créé 4 nouvelles classes, **Niveau1**, **Niveau2, Niveau3**, **Niveau4**. Ces classes héritent de Plateau et nous utilisons le constructeur de Plateau pour définir leur largeur, longueur et nombre d'animal à sauver. A partir de là, nous décidons de créer une autre classe, Obstacle, qui hérite de Contenu (comme Cube et Animal) puis nous effectuons toutes les modifications requises. C'est à dire que l'on ne peut pas sélectionner un obstacle, on ne peut pas non plus faire tomber un obstacle ni le supprimer. Une fois cela fait, nous sommes revenues sur nos Niveaux, et nous avons commencés à remplir les cases manuellement. Nous avons fait une erreur ici car nous commencions à faire un remplissage du type :

```
Obstacle mur = new Obstacle();
this.remplirCase(1,1,mur);
this.remplirCase(1,3,mur);
```

```
Animal chien = new Animal("Dog");
this.remplirCase(1,2,chien);
this.remplirCase(1,6,chien);


Cube rose = new Cube("pink");
this.remplirCase(2,1,rose) ;
this.remplirCase(3,1,rose);
```

C'est-à-dire qu'on construisait un seul objet et qu'on le plaçait dans les cases appropriés. Quand nous avons essayé le programme, le tableau s'affichait correctement dans le terminal, mais dès que l'on sélectionnait des coordonnées pour jouer, si on sélectionnait un Cube rose par exemple, le programme supprimait tous les Cubes roses du plateau. Nous ne comprenions pas pourquoi, nous avons cherché longtemps du coté de notre fonction selectionCubeCouleur() puis nous avons trouvé le problème ! L'objet que l'on créait était seulement dupliqué dans les Cases, c'était le même objet, le même Cube donc si un avait *vaEtreSupprimé()* à true, alors les autres aussi. Nous avons donc modifié pour créer autant d'objet qu'il y a de Cases sauf pour les Obstacles qui ne sont finalement que de la décoration qui n'interagit pas. Ce gros problème étant réglé, nous avons décidé d'inclure le système d'étoile également du jeu.

Interface Etoile

Pour cela, une interface **Etoile** a été créé, contenant 2 méthodes, donneEtoile(int x) et afficheEtoile(). Plateau implémente cette dernière pour qu'elle soit également commune aux Niveaux par les fonctions. Le système est simple, on ajoute 3 booléens etoile1, etoile2, etoile3 pour savoir combien d'étoile on a à chaque fin de partie, et on associe un nombre de points à avoir pour débloquer une étoile. On override donneEtoile() dans les Niveaux par rapport aux points à avoir et puis on implémente afficheEtoile() dans le plateau qui affiche une étoile si le booléen est à true.

Joueur , Jeu , Lanceur

Les niveaux sont finis, le jeu fonctionne très bien sur le terminal. On décide de créer une classe **Joueur** que l'on place dans un package *joueur* car c'est pour le contrôle du jeu et non pas la création du jeu. Dans cette, classe on ajoute des attributs String *nom*, Scanner *scanReponse*, un tableau de int p*ointJoueur* et un int *dernierNiveau*. Puis on créé des méthodes , surtout des setters et des getters et on déplace selectionnerCaseX() et selectionnerCaseY() de Plateau pour le mettre dans cette classe, Joueur.

Une fois tout cela fait, notre jeu fonctionnait très bien avec Test_Projet qui contenait le main et la succession de méthodes pour créer le jeu version Terminal. Il fallait que l'on commence la partie graphique. Mais avant on s'est dite que l'on allait supprimer la classe Test_Projet et la séparer en deux nouvelles classes, **Jeu** et **Lanceur.** La Partie Terminale de la classe Jeu est exactement ce qu'il y avait dans Test_Projet sauf par rapport à l'id, nous avons ajouté cet attribut dans Plateau pour donner un id à chaque Niveau avec un getter. Dans Lanceur, nous avons créer un nouveau jeu dans le main et lancer le programme puis par la suite, un nouvel Affichage, une classe que nous allons voir dès maintenant.

Affichage (classe interne : Cube, JpanelWithBackground)

Cette nouvelle classe **Affichage** qui contient des champs *jeu*, *plateau* et *joueur* au est la Vue de notre jeu . Grâce à Window Builder, nous créons ensuite un premier accueil en plaçant des boutons pour lancer le niveau 1, 2 , 3 et 4 et créant leur design. C'est la seule utilisation que nous ferons de cet outil. Ensuite nous assignons des *Actions listeners* aux boutons, pour lancer les niveaux, nous écrivons la méthode mais elle est vide pour le moment. Notre priorité est de créer le plateau ; on créer un fonction qui retourne un panel getNiveau() qui aurait une configuration en GridLayout de dimension plateau.longueur* plateau.largeur. Pour placer les cubes, on pense d'abord à créer une fonction qui créé des boutons, ces boutons seraient des cubes auxquels on aurait assigné des actions listener, mais cette idée fut finalement avortée car nous n'arrivions pas à utiliser mouseClicked() pour avoir les coordonnées de la Case (donc du bouton).

C'est là qu'est venu l'idée de créer une classe interne **Cube,** qui hérite de *Jpanel*, pour représenter les cases qui seraient des panels et non plus des boutons. Cette classe interne Cube implémente *MouseInputListener* donc même si nous ne les utilisions pas, nous avons du mettre toutes les méthode de *MouseInputListener*. On fait le nécessaire pour que le panel ait bien l'image qu'on souhaite avec paintComponent(Graphic g) puis nous redéfinissons la méthode qui nous intéresse, mouseClicked() qui récupère les coordonnés de la case.

Par la suite, on créé une nouvelle méthode placementCube() qui place correctement les cubes, obstacles et animaux, par rapport aux noms, nous donnons la bonne image et la bonne coordonnée dans le tableau. Et là un problème survient, nous avions fait en sorte que viderCube() place le Contenu de la case à null une fois vide, donc pour recréer le tableau à chaque clique, cela génère des NullPointerException. Pour régler ça, nous avons dit que le Contenu changerait de nom pour « vide » (au lieu de pink, blue etc) et l'image du nouveau cube serait blanc, comme si il avait disparu vu que le fond est blanc.

Ceci fait, on ajoute la fonction à getNiveau(). Nous avons donc créé un panel ayant le plateau qui s'affiche sans trop de difficulté. Par contre, nous avions un problème, on arrivait pas à passer d'un panel à l'autre avec le bouton; nous avons tenter d'avoir un CardLayout mais ce fut un échec car malgré la documentation, et les recherches, nous ne sommes pas arrivés à le maîtriser. Alors tout simplement nous avons fait en sorte que le bouton affiche le nouveau panel avec miseAJour(Jpanel p) qui supprime l'ancien panel et place le nouveau sur le ContentPane. Nous pouvons donc changer de panel comme nous le souhaitons.

Nous avons aussi fait une fonction mettreAjour() pour que le panel de la page du Niveau se mettre à jour à chaque clique. Puis nous avons ajouté des instructions dans miseAjour() pour créer un panel général ou il y a, le score d'affiché, le nombre d'animaux à sauvé et le plateau. Ce panel étant fonctionnel, nous avons ensuite ajouté un panel FindeNiveau pour afficher le score final et le nombre d'étoile, grâce à des conditions de validation, et un bouton pour retourner à la sélection de niveau.

La plupart des fonctions que les Actionlistener des boutons appellent sont dans Jeu pour séparer la partie **Vue** de la partie **Controleur.** Nous avons ensuite décidé de faire un accueil avec des boutons Nouvelle Partie, Continuer la partie (pour la sérialisation) et Quitter. Pour Quitter, nous avons créer un JOptionPane pour confirmer le fait de quitter le jeu. Et pour Nouvelle Partie, nous l'avons juste rattaché à SelectionNiveau.

Par la suite, nous avons décidé d'ajouter une image de fond aux Jpanels, alors nous avons crée une autre classe interne de Affichage, **JpanelWithBackground** qui hérite aussi de Jpanel. Le constructeur prend en paramètre un fichier qui est une image. Si il ne trouve pas le fichier, le fond

est juste blanc. Nous l'avons ajouté partout sauf sur le panel du plateau car le fond doit être obligatoirement blanc pour que les cases « vides » paraissent vraiment vide.

Sérialisation , Désérialisation

Nous avons ensuite attaqué la sérialisation puisque nous avions presque fini le projet. Avant toute choses, nous avons implémenté l'interface Serializable sur toute nos classes. Nous l'avons écrit dans la classe Jeu puisqu'il centralise toutes les classes requises pour jouer à un niveau et à une partie en général. Nous avons commencé par sérialiser les niveaux, écrire la fonction Serialisation(Plateau niv, String name) était plutôt facile et est maintenant inutile car les niveaux sérialisés ne changent pas. Nous avons aussi fait une fonction désérialisation(String chemin) pour déserialiser les niveaux au bon moment.Pour le niveau 1, il est désérialisé dès le début. Nous avons grisés les boutons 2-3-4 sur sélection niveau, ils ne sont dégrisés que si le nouveau précédent est gagné. Quand on clique dessus, on désérialise le niveau correspondant, donc on peut refaire les niveaux autant de fois que l'on veut.

Pour sérialiser la partie, le jeu, pour sauvegarder finalement la progression du joueur et rendre le bouton Continuer la partie utilisable. Nous sérialisons donc le jeu au bouton Quitter, et on ajout un autre bouton quitter dans le panel de sélection de niveau. L'idée de départ était de désérialiser le jeu quand on clique sur le bouton, utiliser setJeu(Jeu j) pour remplacer le nouveau Jeu par le Jeu sérialisé puis renvoyer le joueur à selectionNiveau(), il n'y avait pas les assignations du type *niv1 = j.niv1* et la fonction retournait un Jeu.

Malheureusement, cette technique n'a pas fonctionné car une fois le jeu désérialisé, les boutons pour lancer les niveaux fonctionnaient (vérification avec des System.out.print()) mais ne s'affichait pas à l'écran . Nous sommes restés coincés sur ce problème pendant deux jours jusqu'à ce qu'on trouve le problème. Les boutons du panel de la sélection de niveau était actif sur l'ancien jeu malgré le setter. Alors nous avons rendu déserialisationJeu() de type void et nous avons instancier les nouvelles valeurs des niveaux, joueur et plateau aux anciennes. Ceci a réglé correctement le problème.

Les problèmes que l'on a réglé, nouvelles fonctionnalités, échecs

Une fois cela fait, il restait deux grosses choses à faire, deplaceGauche() et implémenter un joueur bot qui aide le joueur à faire le meilleur coup possible. Pour le premier, après beaucoup de recherche sur internet, beaucoup d'échecs, nous avons réussi à le coder. En effet, la fonction se déclenche si la colonne à gauche est vide (Obstacle non compté grace à firstRempli()).

Pour le bot, ce fut assez simple, on lui demande d'évaluer quel cube se multiplie le plus de fois et on appelle selectionCubeCouleur ensuite . Donc on a implémenté un nouveau bouton dans le panel où s'affiche le niveau avec le score etc, « aide » qui appelle le bot à la rescousse du joueur.

Le projet était enfin fini mais il nous restait un peu de temps car nous avions travaillé jour et nuit (même les soirs de fêtes), alors nous avons ajouté des fonctionnalités. La première étant un bouton quitter dans le panel ou s'affiche le plateau pour retourner à la sélection de niveaux et recommencer le niveau si l'on est bloqué. La seconde fonctionnalité est pour arrondir les boutons, leur donner un meilleur aspect avec

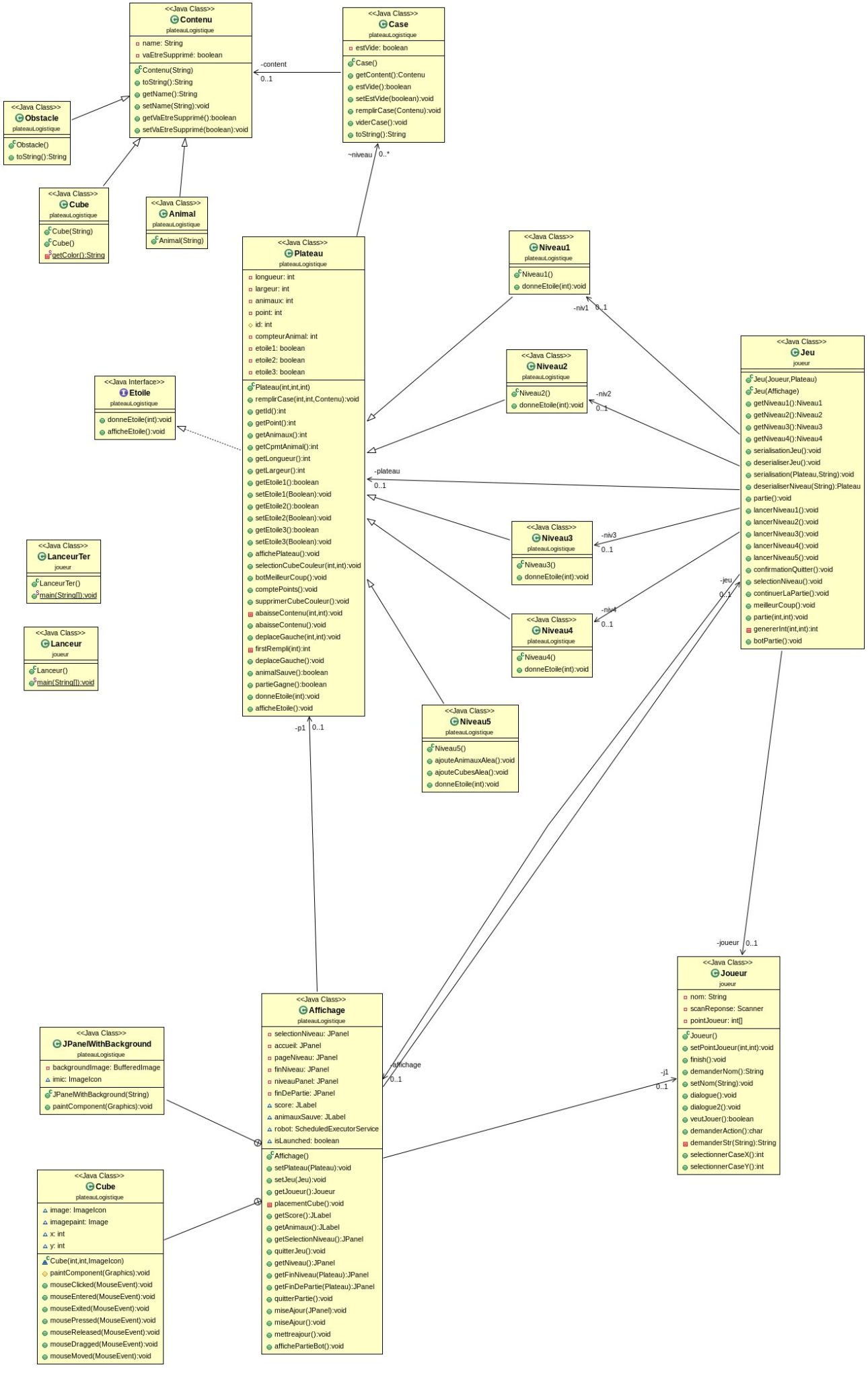
```
UIManager.setLookAndFeel(new NimbusLookAndFeel());
```

Malheureusement cette ligne nous causa une erreur par rapport au bouton Quitter, il était fonctionnel mais nous avions un Null pointer exception qu'on arrivait pas du tout à retirer alors nous avons supprimé le LookAndFeel.

Ensuite, ce sont rajoutés deux gros ajouts que nous avions abandonnés, un bot et un niveau aléatoire. Pour ce dernier, nous avons repris nos fonctions du départ qui générait un plateau aléatoire et les avons arrangé pour qu'il y ait des animaux seulement sur la première ligne. Pour le bot qui joue seul à un niveau, ce fut un peu plus compliqué, mais en lisant beaucoup de la JAVADOC nous avons pu trouver les classes et fonctions adéquats. Le bot a une chance sur deux de faire ne faire que des « coups » parfaits pour terminer un niveau, c'est un plus.

Nous aurions pu ajouter plus de niveaux pour bien voir le niveau de difficulté augmenter et également trouver une solution pour le Look and Feel qui est dû à un problème de sérialisation de la classe NimbusLookAndFeel().

Notre projet étant maintenant terminé, je vous invite à le tester.


<<Java Class>>
Contenu
plateauLogistique
name: String
vaEtreSupprimé: boolean
Contenu(String)
toString():String
getName():String
setName(String):void
getVaEtreSupprimé():boolean
setVaEtreSupprimé(boolean):void
-content
0..1
<<Java Class>>
Case
plateauLogistique
estVide: boolean
Case()
getContent():Contenu
estVide():boolean
setEstVide(boolean):void
remplirCase(Contenu):void
viderCase():void
toString():String
~niveau
0..*
<<Java Class>>
Obstacle
plateauLogistique
Obstacle()
toString():String
<<Java Class>>
Cube
plateauLogistique
Cube(String)
Cube()
getColor():String
<<Java Class>>
Animal
plateauLogistique
Animal(String)
<<Java Class>>
Plateau
plateauLogistique
longueur: int
largeur: int
animaux: int
point: int
id: int
compteurAnimal: int
etoile1: boolean
etoile2: boolean
etoile3: boolean
Plateau(int,int,int)
remplirCase(int,int,Contenu):void
getId():int
getPoint():int
getAnimaux():int
getCpmtAnimal():int
getLongueur():int
getLargeur():int
getEtoile1():boolean
setEtoile1(Boolean):void
getEtoile2():boolean
setEtoile2(Boolean):void
getEtoile3():boolean
setEtoile3(Boolean):void
affichePlateau():void
selectionCubeCouleur(int,int):void
botMeilleurCoup():void
comptePoints():void
supprimerCubeCouleur():void
abaisseContenu(int,int):void
abaisseContenu():void
deplaceGauche(int,int):void
firstRempli(int):int
deplaceGauche():void
animalSauve():boolean
partieGagne():boolean
donneEtoile(int):void
afficheEtoile():void
-p1
0..1
<<Java Class>>
Niveau1
plateauLogistique
Niveau1()
donneEtoile(int):void
-niv1
0..1
<<Java Class>>
Niveau2
plateauLogistique
Niveau2()
donneEtoile(int):void
-niv2
0..1
<<Java Class>>
Niveau3
plateauLogistique
Niveau3()
donneEtoile(int):void
-niv3
0..1
<<Java Class>>
Niveau4
plateauLogistique
Niveau4()
donneEtoile(int):void
-niv4
0..1
<<Java Class>>
Niveau5
plateauLogistique
Niveau5()
ajouteAnimauxAlea():void
ajouteCubesAlea():void
donneEtoile(int):void
<<Java Class>>
Jeu
joueur
Jeu(Joueur,Plateau)
Jeu(Affichage)
getNiveau1():Niveau1
getNiveau2():Niveau2
getNiveau3():Niveau3
getNiveau4():Niveau4
serialisationJeu():void
deserialiserJeu():void
serialisation(Plateau,String):void
deserialiserNiveau(String):Plateau
partie():void
lancerNiveau1():void
lancerNiveau2():void
lancerNiveau3():void
lancerNiveau4():void
lancerNiveau5():void
confirmationQuitter():void
selectionNiveau():void
continuerLaPartie():void
meilleurCoup():void
partie(int,int):void
genererInt(int,int):int
botPartie():void
-plateau
0..1
-jeu
0..1
<<Java Interface>>
Etoile
plateauLogistique
donneEtoile(int):void
afficheEtoile():void
<<Java Class>>
LanceurTer
joueur
LanceurTer()
main(String[]):void
<<Java Class>>
Lanceur
joueur
Lanceur()
main(String[]):void
<<Java Class>>
Affichage
plateauLogistique
selectionNiveau: JPanel
accueil: JPanel
pageNiveau: JPanel
finNiveau: JPanel
niveauPanel: JPanel
finDePartie: JPanel
score: JLabel
animauxSauve: JLabel
robot: ScheduledExecutorService
isLaunched: boolean
Affichage()
setPlateau(Plateau):void
setJeu(Jeu):void
getJoueur():Joueur
placementCube():void
getScore():JLabel
getAnimaux():JLabel
getSelectionNiveau():JPanel
quitterJeu():void
getNiveau():JPanel
getFinNiveau(Plateau):JPanel
getFinDePartie(Plateau):JPanel
quitterPartie():void
miseAjour(JPanel):void
miseAjour():void
mettreajour():void
afficheFinBot():void
-affichage
0..1
-joueur
0..1
<<Java Class>>
Joueur
joueur
nom: String
scanReponse: Scanner
pointJoueur: int[]
Joueur()
setPointJoueur(int,int):void
finish():void
demanderNom():String
setNom(String):void
dialogue():void
dialogue2():void
veutJouer():boolean
demanderAction():char
demanderStr(String):String
selectionnerCaseX():int
selectionnerCaseY():int
-j1
0..1
<<Java Class>>
JPanelWithBackground
plateauLogistique
backgroundImage: BufferedImage
imic: ImageIcon
JPanelWithBackground(String)
paintComponent(Graphics):void
<<Java Class>>
Cube
plateauLogistique
image: ImageIcon
imagepaint: Image
x: int
y: int
Cube(int,int,ImageIcon)
paintComponent(Graphics):void
mouseClicked(MouseEvent):void
mouseEntered(MouseEvent):void
mouseExited(MouseEvent):void
mousePressed(MouseEvent):void
mouseReleased(MouseEvent):void
mouseDragged(MouseEvent):void
mouseMoved(MouseEvent):void