Université de Paris

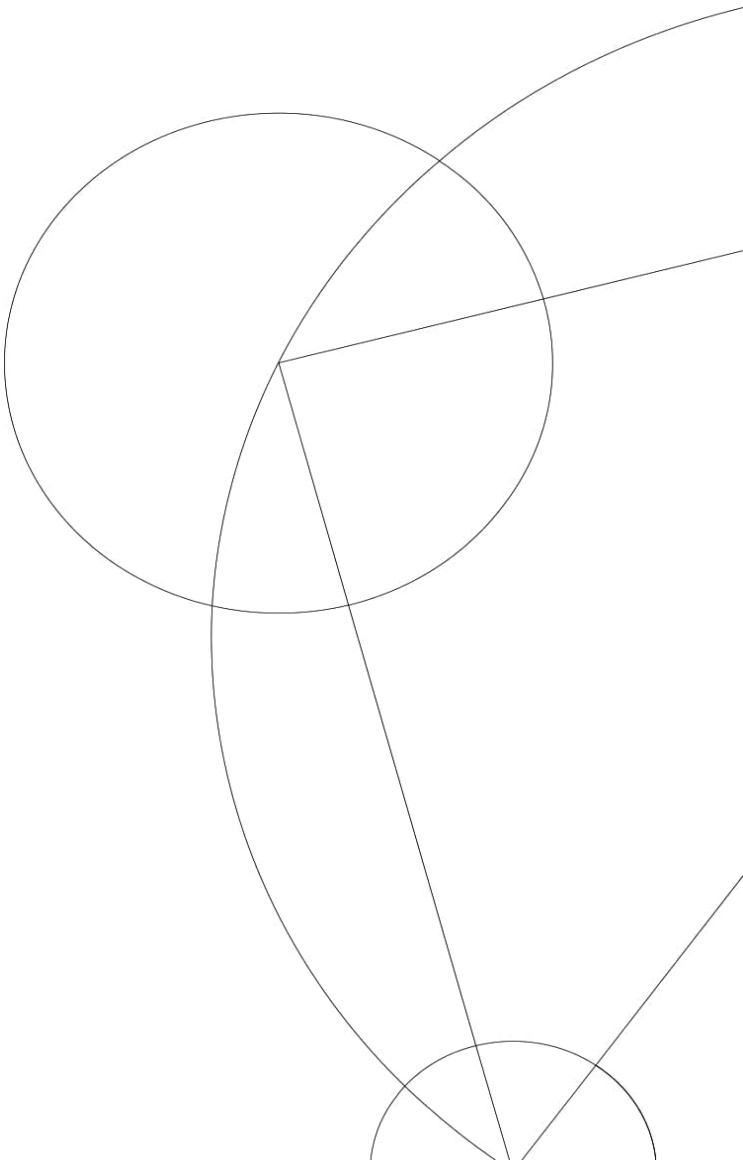
Projet de Programmation orientée objet et Interfaces graphiques 2020 (POOIG)

Rapport: "Pet" Rescue Saga

GROUPE 3 (Math-Info 2)

Damya BOUIZEGARENE **21961216**
bouizegarene.damya@gmail.com

Thuy Vi Émilie NGUYEN **21953967**
emilie.ngn10@gmail.com



Sommaire

1 Introduction

Le projet réalisé est basé sur le jeu Pet Rescue Saga.

Notre version, **Dino Rescue Saga**, a été achevée en deux mois à l'aide de la plateforme collaborative GitLab de l'Université Paris Diderot et de nos connaissances tirées des cours de POO. Elle reprend les mêmes principes de base que le jeu originel : il faut ramener sains et saufs les dinosaures jusqu'à l'arche. Pour cela le joueur devra détruire les blocs adjacents de même couleur, user de bonus généreusement proposés par les développeurs et déjouer les obstacles du level design.

Le programme comprend un affichage textuel et une interface graphique dont le choix de la vue est laissé au joueur lors du lancement. Ils sont tous deux équivalents en terme de contenu : ils accompagnent le joueur dans sa progression à travers une suite de quatre niveaux à difficulté croissante - chaque niveau ayant pour volonté de montrer des options différentes du gameplay implémenté. Le programme est également muni d'un système de sauvegarde qui stocke les niveaux sur le disque et permet au joueur d'enregistrer ces meilleurs scores ou encore de s'arrêter au milieu d'une partie et de reprendre plus tard.

2 Parties traitées

2.1 Contenu

Dino Rescue Saga implémente les fonctionnalités des vingt-deux premiers niveaux du jeu Pet Rescue Saga sur mobile (voire des fonctionnalités en plus).

2.1.1 Partie

Nous avons implémentés en premier les bases du jeu :

- la gestion et réorganisation du plateau (cas d'un plateau d'une grande hauteur ou avec plein de blocs fixes);
- les différents blocs intervenant dans les niveaux (bloc coloré, plateforme, trou et animal) et leur gestion à travers des cases;
- la destruction des blocs colorés;
- l'implémentation de la victoire et de la défaite : les différentes conditions de victoire (tous les animaux sauvés, nombre minimal d'animaux à sauver, score minimal à atteindre), le nombre limité ou illimité de déplacement;
- la vérification de la possibilité d'un mouvement et son application.

Le premier niveau est alors fonctionnel.

Par la suite nous avons ajouté des options plus avancées :

- les blocs spéciaux (grille, combo, bombe, ballon);
- les bonus (marteau, fusée, bombe et tenailles);
- les indices permettant d'avoir le meilleur score;

- la génération infinie de bloc;
- un bot pour résoudre le niveau.

Ces options permettent la création des niveaux suivants, bien plus complexes.

2.1.2 Sauvegarde

L'interface `Serializable` a été utilisée pour implémenter le système de sauvegarde. Toutes les méthodes appelées pour enregistrer les parties et les niveaux sur le disque sont implémentées dans la classe `Niveau`.

Le répertoire `sauvegarde` est divisé en deux sous-répertoires, `initial` et `progression`, dans le but de distinguer les niveaux vierges et ceux entamés ou finis par le joueur. Cette organisation permet la sélection d'un niveau, la sauvegarde du meilleur score pour chaque niveau et l'option d'abandonner et donc de réinitialiser le niveau en cours, ou de le quitter pour y revenir plus tard.

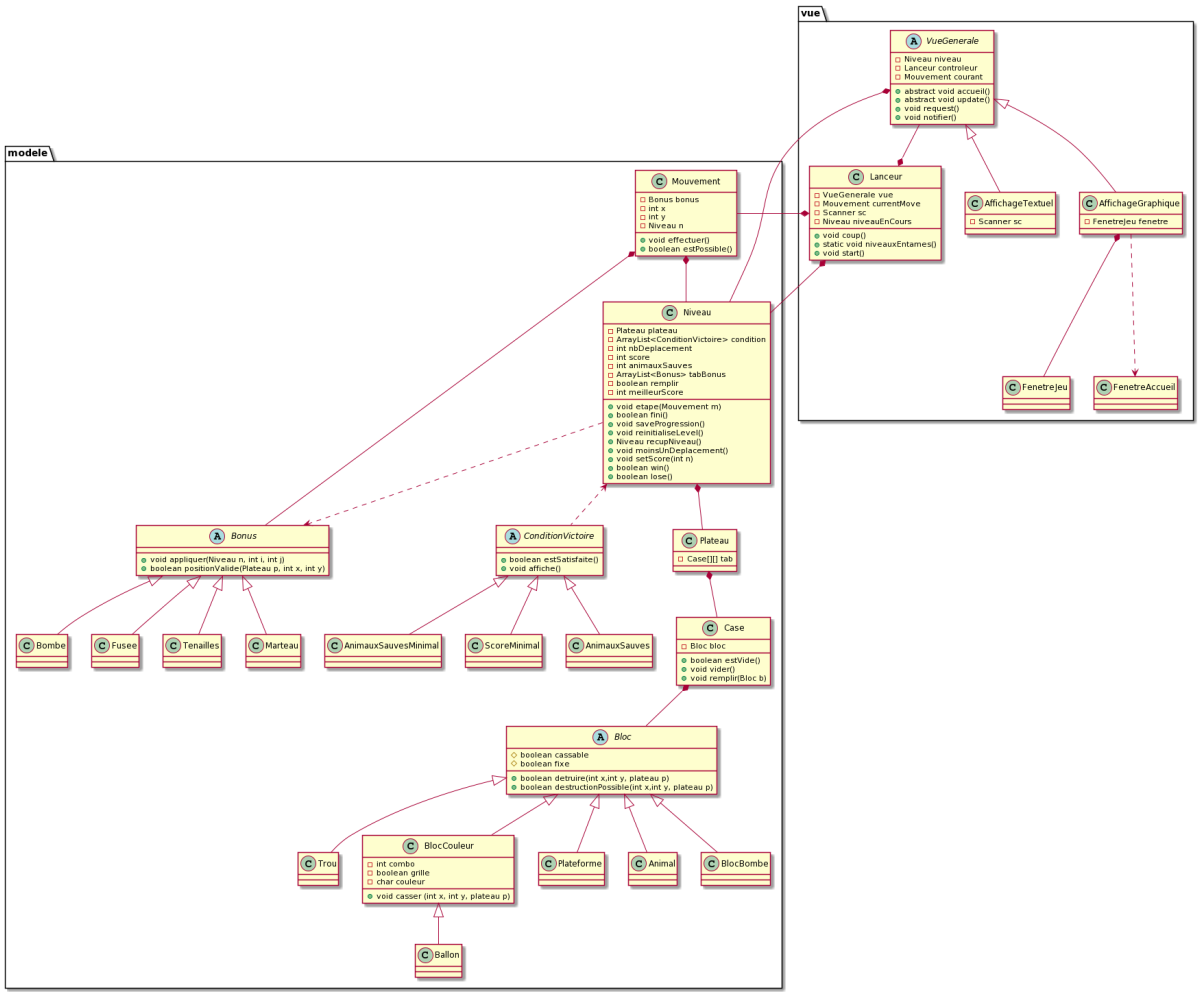
2.1.3 Vue

Nous avons intégrés une interface textuelle ainsi que graphique. Le répertoire `images` complète l'interface graphique.

En premier lieu, elles présentent chacune au joueur l'accueil : c'est ici qu'il prend connaissance du but du jeu, voit le meilleur score atteint dans chaque niveau et sélectionne le niveau qu'il souhaite jouer. Il y a ensuite une transition avec un autre affichage (textuel sur console ou nouvelle fenêtre) qui est celui du niveau en cours : l'objectif menant à la victoire y est inscrit ainsi que le score, le nombre de déplacement et d'animaux, les bonus disponibles et le plateau de jeu avec lequel interagira le joueur à travers le terminal en répondant aux questions, si affichage textuel, où les cliques de la souris, si affichage graphique.

2.2 Architecture

2.2.1 Représentation graphique de l'architecture



2.2.2 Fonctionnement

Nous avons choisi une architecture qui exploite au mieux l'héritage de manière à obtenir un code simple et qui permette d'ajouter des options sans modifier le code de base. En voici les exemples les plus importants.

- Séparation modèle-vue
  Le "fond" du logiciel (principalement les niveaux et leurs évolutions) sont dans le paquet `modele`. Les éléments liés à la vue sont dans le paquet `vue`.
  Aucune des classes du modèle ne fait appel aux classes de la vue.
  Le paquet `vue` contient la classe `Lanceur` dont le rôle est de faire le lien entre l'utilisateur et le modèle, par le biais de la vue. On peut ainsi dire que c'est un contrôleur.
  Un objet de type `Lanceur` "possède" donc impérativement des éléments du modèle sur lesquels il agit (un `Niveau` qui est le niveau en cours et un `Mouvement` qui est le mouvement à effectuer) ainsi qu'un objet de type `VueGenerale`.

Au cours d'une partie, il actualisera cette vue via la fonction `Vue.update()` et lui demandera éventuellement les choix de l'utilisateur à travers la fonction `Vue.request()`. En retour, la vue notifiera le lanceur en utilisant la méthode `Vue.notifier()` pour qu'il effectue le mouvement suivant, qui dans notre cas consiste à jouer le mouvement sélectionné par exemple. On peut donc également parler ici d'un design pattern "Observer".
Vous remarquerez qu'à aucun moment nous spécifions le type de la vue: en effet `VueGenerale` est une classe abstraite étendue par deux classes représentant les deux vue proposées par notre logiciel. Les spécificités de chaque vue sont implémentées dans ces classes: par exemple, la fonction `request()`, lorsqu'elle est appelée sur un `AffichageGraphique` ne fait rien car l'affichage muni de boutons est lui-même une demande d'action.

- La destruction des blocs
  La destruction des blocs qui composent le plateau d'un niveau donné est un exemple de l'exploitation du dynamic binding. La classe `Bloc` est une classe abstraite contenant la méthode `boolean detruire(int x, int y, Plateau p)` qui permet d'éventuellement détruire l'objet selon la sous-classe de `Bloc` sur laquelle elle est appelée. Par exemple, la méthode telle qu'elle est redéfinie dans `Plateforme` ne fait rien et renvoie false car ce type de bloc n'est pas cassable.
  Dans la classe `BlocCouleur` elle détruit tous les blocs adjacents de même couleur ou casse leur grille et dans la classe `Ballon`, elle détruit tous les blocs de la même couleur que le bloc courant. Cela permet d'appeler une unique méthode pour tous les blocs dans la classe `Plateau` au moment de la destruction d'un bloc.

- Bonus
  Les bonus utilisent un motif similaire. Les classes étendant `Bonus` contiennent toutes une redéfinition de la méthode `void appliquer(Niveau n, int i, int j)` qui est utilisée au moment d'exécuter un `Mouvement` muni de ce bonus.

- Conditions de victoire
  Il en est de même pour les conditions de victoire qui étendent la classe `ConditionVictoire` et redéfinissent `boolean estSatisfaite(Niveau n)` (de la même manière qu'on utiliserait l'interface `Predicate`)
  Ce motif, s'apparentant à un strategy pattern, permet d'ajouter à l'avenir d'autres blocs spéciaux, conditions de victoire et bonus en étendant à nouveau les classes `Bloc`, `ConditionVictoire` et `Bonus`.

3 Problèmes connus

Un certains nombre de problèmes ont été rencontrés lors de l'élaboration du projet. En effet la nature du logiciel demandé implique une grande liberté lors de la création de l'architecture, une tâche qui a donc impliqué plusieurs décisions importantes:

- Interaction avec le joueur
  Devrait-il y avoir une classe `Partie` qui fait le lien entre l'utilisateur et le niveau? Une option que nous avons rejetée au profit de la classe `Mouvement` associée à la classe `Lanceur` par soucis d'une séparation nette entre la vue et le modèle.

- Robot aléatoire
  Bien qu'il soit maintenant fonctionnel (il génère bien une partie entière jouée de manière aléatoire), il ne laisse cependant pas à la vue "graphique" le temps d'en afficher les étapes, chose qui est très bien faite dans la version textuelle du logiciel. Le code de la méthode `randomBot()` du lanceur contient en commentaire une méthode infructueuse pour "retarder" l'exécution de chaque coup du bot en utilisant `Thread.sleep()`. Une autre solution aurait consisté à ne réaliser qu'une seule étape aléatoire, de la manière suivante:

```
public void randomBot(){
  if (niveauEnCours!=null){
        Mouvement rand= Mouvement.randomMove(niveauEnCours);
        currentMove=rand;
        niveauEnCours.etape(currentMove);
        vue.update();
        if (!niveau.fini()){
            vue.request();
        }else {
              vue.jeuFini();
        }
  }
}
```

4 Pistes d'extensions

- Niveaux déverrouillés
  Nous aurions pu implémenter le déverrouillage progressif des niveaux. Il aurait ainsi fallu gagner au moins une fois le niveau numéro n pour pouvoir jouer au niveau n+1. Cela aurait cependant pu prendre du temps à démontrer lors de la soutenance -les différentes fonctionnalités du projet étant étendues sur les 4 niveaux.
  Voici comment nous pourrions implémenter cela:
  - Ajouter un attribut `private boolean reussi` à la classe niveau qui indiquerait si le niveau a été gagné au moins une fois (il serait donc initialement `false`)
  - Actualiser cet attribut lors de la sauvegarde de la progression en modifiant la méthode `void reinitialiseLevel()` de la classe `Niveau` appelée à la fin d'une partie comme suit:

```
public void reinitialiseLevel() throws IOException,
    ClassNotFoundException{
   int n= Math.max(score, meilleurScore);
   boolean w= (this.reussi || this.win());
   Niveau sauv= new Niveau(numero);
   sauv=
       sauv.recupNiveau("sauvegarde/initial/niveau"+numero+".ser");
   sauv.meilleurScore=n;
   sauv.reussi=w;
   sauv.save("sauvegarde/progression/niveau"+numero+".ser");
}
```

  - Lors de l'affichage de l'accueil dans les classes étendant `VueGenerale`, n'afficher que les niveau dont l'attribut `reussi` est vrai et le premier niveau dont l'attribut `reussi` est faux

- Éditeur de niveau
  Nous pourrions envisager la création d'un créateur simple de niveau que le joueur pourrait ouvrir à l'accueil du jeu.