Projet CPS - Lemmings

Patrick Tran, Ladislas Halifa

4 mai 2016

1 Introduction

Lemmings est un jeu vidéo de réflexion sorti en 1991 dont le principe est de guider tous les lemmings, de petites créatures écervelées, vers une sortie prédéfinie dans le terrain de jeu. Pour y arriver, le joueur a le choix de donner différentes capacités aux lemmings, qui incluent par exemple la capacité de grimper des obstacles ou de creuser la terre. Contrairement à la version officielle du jeu, le jeu développé dans le cadre du projet ne contient pas toutes les fonctionnalités existantes (type de lemmings, types de case, modes). Dans ce rapport, seront détaillés les règles du jeu ainsi que le mode d'emploi, l'implémentation et les difficultés rencontrées.

2 Règles du jeu

Comme décrit dans l'introduction, le but du jeu est de guider tous les lemmings générés vers la sortie du terrain. Avant de jouer, le terrain doit être paramétré par le joueur (hauteur, largeur, taille de la colonie, vitesse de création des lemmings). Le jeu fonctionne d'après un système de tours, où à chaque tour, chaque lemming effectue une action propre à son type.

Le terrain contient des cases de type EMPTY, DIRT ou METAL. Les cases EMPTY sont les cases dans lesquelles les lemmings peuvent se déplacer librement. Les cases DIRT peuvent être détruites avec un lemming creuseur ou exploseur par exemple. Les cases METAL ne peuvent pas être détruites.

Le joueur peut donner une capacité spéciale à un lemmings en utilisant un jeton de la capacité en question. Pour chaque type (capacité), le joueur a au départ 10 jetons. Une fois ce nombre tombé à 0, il ne pourra plus donner le type à un lemming.

Le score obtenu à la fin en pourcentage correspond à la proportion de lemmings sauvés par rapport au nombre de lemmings crées. L'objectif est donc d'obtenir le score le plus proche de 100% possible.

3 Mode d'emploi

3.1 Configuration du terrain

3.1.1 Dimensions

Au lancement du jeu, le joueur doit au préalable configurer le terrain. Il doit spécifier la hauteur, la largeur, la taille de la colonie (nombre de lemmings à créer) et la vitesse de création des lemmings où 5 par exemple, correspond à un lemming créé tous les 5 tours. Une fois tous les paramètres remplis, appuyer sur le bouton "Valider" permettra de généré un nouveau terrain dont toutes les cases des bords sont en métal.

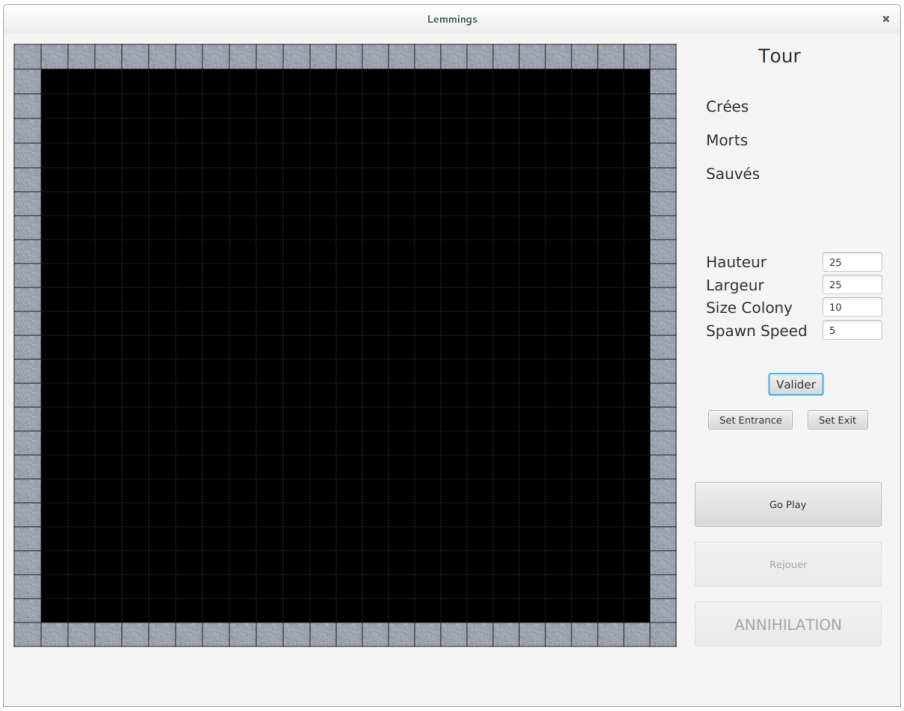

FIGURE 1 – Etat de l'interface après une première configuration du plateau

3.1.2 Modification des cases

Un terrain doit au minimum contenir une case d'entrée et une case de sortie pour les lemmings. Les boutons "Set Entrance" et "Set Exit" permettent de les placer. Une fois un des boutons cliqué, le joueur peut décider ou les placer en cliquant directement sur le plateau. Des conditions existent sur le placement d'une entrée ou d'une sortie, par exemple, une entrée doit être placée de telle

sorte que la case au dessus et en dessous soient vides. La sortie elle, doit être placée de telle sorte que la case au dessus soit vide et la case en dessous soit en métal.

Pour changer le type d'une case, le joueur peut utiliser les 3 boutons de la souris. En cliquant sur une case du terrain avec le bouton gauche, elle devient une case DIRT, avec le bouton droit, elle devient une case METAL et avec le bouton du milieu, une case EMPTY. Il est possible de faire glisser la souris sur plusieurs cases avec le bouton enfoncé pour changer plus facilement le type de plusieurs cases. Une fois la construction du terrain finie, le joueur peut lancer la partie en appuyant sur le bouton "Go Play" en vérifiant bien que tous les bords du terrain sont en METAL et que les conditions pour la case entrée et sortie sont bien respectées.

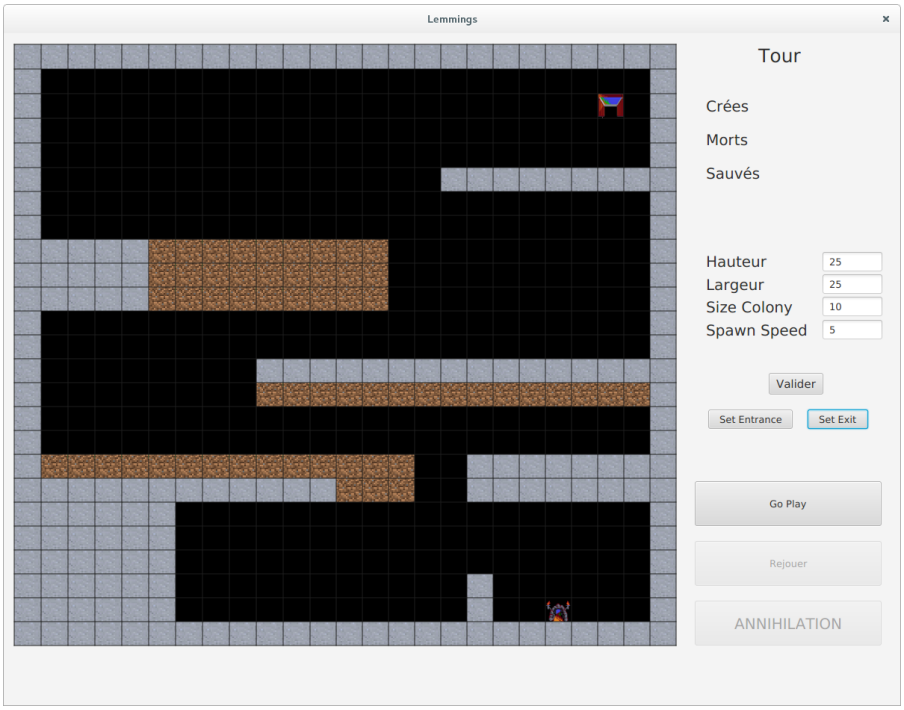

FIGURE 2 – Etat de l'interface après la construction du terrain

3.2 En jeu

Pendant la partie, le joueur peut attribuer une capacité à un lemming. Pour cela, différents boutons apparaissent au début de la partie, chaque bouton représentant une capacité. Cliquer sur un des boutons puis cliquer sur un lemming ou plusieurs lemmings permet de lui ou leur attribuer la capacité correspondante. Si cela réussi, le nombre de jetons restant pour la capacité décrémente.

Il n'y a aucun moyen de récupérer des jetons et attribuer une capacité est une action irréversible.

Attention : certaines capacités ne sont pas compatibles. Un basher ne peut par exemple pas devenir stopper ou un lemming qui tombe (ou flotte) ne peut pas devenir stopper tant qu'il n'est pas sur la terre ferme.

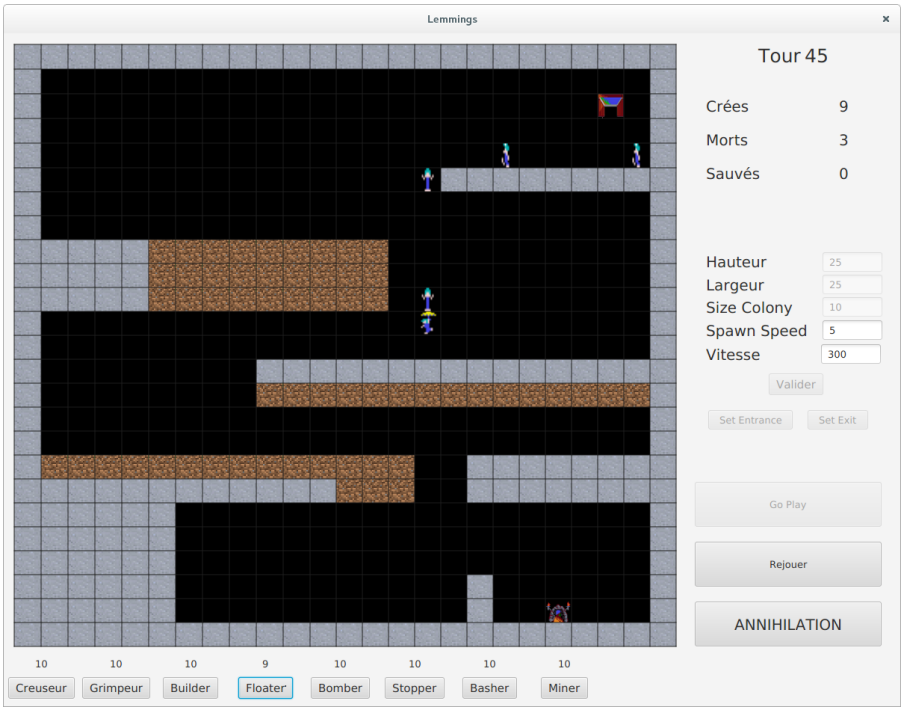

FIGURE 3 – Etat de l'interface une fois que le joueur a lancé la partie

Le nombre de lemmings créés, morts et sauvés est indiqué sur le coin supérieur droit de la fenêtre.

La valeur du champ "Spawn Speed" est modulable grâce à la molette de la souris : il suffit de placer le pointeur de la souris sur le champ et de faire glisser la molette vers le haut pour incrémenter la valeur (ralentir la création des lemmings) et vers le bas pour décrémenter la valeur (accélérer la création des lemmings). Il en est de même pour la valeur du champ "Vitesse" qui correspond à la vitesse de déroulement de jeu.

Le bouton "Rejouer" permet de recommencer la partie tandis que le bouton "ANNIHILATION" permet d'arrêter la partie en arrêtant la création des lemmings, transformant tous les lemmings actifs en exploseur.

4 Implémentation

4.1 Structure

Notre version du jeu est composée de 4 composants :

— Lemming, dans lequel est géré tout ce qui est en rapport avec le comportement d'un lemming ;
— Level, qui permet de gérer le terrain (dimensions, cases, entrée/sortie) ;
— GameEng, qui gère le déroulement du jeu (liste des lemmings, création des lemmings, tours) ;
— Joueur, qui permet au joueur d'interagir avec les lemmings et de leur attribuer des capacités ;

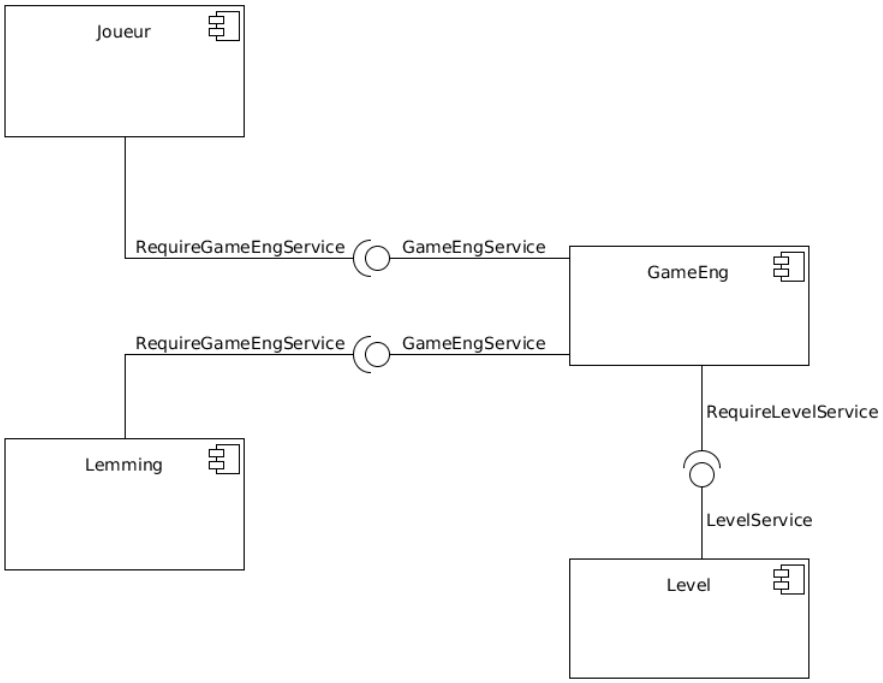

FIGURE 4 – Diagramme de composant

Le composant GameEng requiert un LevelService pour accéder aux natures des cases du terrain. Le composant Lemming requiert un GameEngService pour pouvoir lui aussi accéder aux natures des cases et aux dimensions du terrain. De même, le composant Joueur requiert un GameEngService pour accéder aux lemmings actifs.

4.2 Choix d'implémentation des composants

4.2.1 Level

Nous utilisons une énumération simple, "Nature", contenant les différents types de case possibles. Les seuls types représentés sont EMPTY (case vide), DIRT (case en terre destructible), METAL (case en métal indestructible). Le terrain est donc stocké sous forme de tableau à 2 dimensions de type "Nature". Un second terrain "backup" est stocké et utilisé lorsque le joueur souhaite relancer la partie. Les positions de l'entrée et la sortie sont représentées par un objet de type Point.

4.2.2 Joueur

Pour la gestion des jetons de chaque capacité de lemming, nous avons décidé d'utiliser une HashMap de couples String-Integer. Les chaines de caractères correspondent aux types et les entiers aux nombres de jetons restants pour le type.

4.2.3 GameEng

Le moteur de jeu est composé de différents compteurs correspondants aux nombres de tours écoulés, de lemmings créés, actifs, morts et sauvés. D'autres variables permettent de stocker les informations de jeu, notamment le rythme de création de lemmings, et la taille de la colonie à créer.

Il nous a semblé judicieux de n'utiliser qu'une Map de couple Integer-Lemming liant les id (correspondant au numéro de création d'un lemming) aux lemmings actifs. Stocker les lemmings morts ou sauvés n'était pas nécessaire, aucune action n'étant permise sur ces lemmings, et leur présence n'ayant pas d'impact sur le déroulement du jeu. La structure de Map étant très simple à manipuler, nous l'avons préféré à la structure de tableau pouvant laisser des cases vides à la mort ou au sauvetage d'un lemming et ainsi favoriser les NullPointerException, ou bien une List qui ne permet pas de récupérer facilement un lemming à partir de son identifiant.

Un problème s'est posé lors de l'implémentation : tous les types de lemming n'appréhendent pas de la même façon les obstacles ou les autres lemmings. Nous avons donc implémenté 3 fonctions différentes permettant de détecter un obstacle. Ces 3 fonctions ont de légères différences mais sont surtout utilisées dans différentes situations.

- isObstacle permet de détecter si une case est de nature METAL ou DIRT ou si elle est occupée par un lemming stoppeur. Cela permet à un lemming grimpeur de savoir s'il peut grimper ;
- isObstacle2 permet de détecter si une case est occupée par un lemming actif. Cela permet d'arrêter un lemming creuseur ou basher lorsqu'il rencontre un lemming sur son passage ;
- isLibre permet de détecter si une case n'est ni de nature METAL ou DIRT et ni occupée par un lemming actif. Cela permet à un lemming

builder d'arrêter de construire son escalier lorsqu'il rencontre un obstacle ou un lemming ;

4.2.4 Lemming

Chaque lemming a des variables contenant son id, ses coordonnées (deux entiers), son type et sa direction. Deux autres énumérations ont dues être mises en place pour le type et la direction d'un lemming. L'énumération Direction contient DROITIER et GAUCHER. L'énumération Type contient elle, les types MARCHEUR, TOMBEUR, CREUSEUR, BASHER, MINER et STOPPEUR. Tous les types n'apparaissent pas dans cette énumération car certains sont cumulables avec d'autres types. Par exemple, un flotteur peut être cumulable avec n'importe quel autre type. C'est pourquoi, les types restants à savoir GRIMPEUR, FLOTTEUR, BUILDER et EXPLOSEUR, sont indiqués via des booléens (respectivement isGrimpeur, isFlotteur, isBuilder, isExploseur). D'autres variables ont été mises en place :

- tombeDepuis permet de savoir depuis combien de tours un lemming tombe (pour pouvoir le faire mourir s'il n'est pas un flotteur et que cette valeur est supérieure ou égale 8) ;
- nombreTourBuilder permet de savoir depuis combien de tours un lemming attend pour poser une dalle ;
- nombreDallesPosees permet de savoir combien de dalles un lemming builder a posé ;
- nbCreuseTunnel permet de savoir combien de fois un lemming basher a creusé ;
- exploseurDepuis permet de savoir depuis combien de tours un lemming est exploseur ;

Ces booléens sont utiles lorsqu'un type a des conditions à respecter, par exemple un exploseur explose 5 tours après être devenu exploseur (se référer à la consigne pour les autres conditions).

4.3 Choix d'implémentation pour les types de lemmings

Les lemmings creuseur, pelleteur et mineur se comportent comme décrit et s'arrêtent lorsqu'ils rencontrent un autre lemming sur leur chemin. Le lemming mineur ne peut cependant pas "miner" vers le haut.

Le lemming constructeur se comporte comme décrit dans l'énoncé mais en plus de s'arrêter quand il a posé 12 dalles ou quand les trois cases qu'il rencontre (à gauche ou à droite en fonction de sa direction), il s'arrête lorsqu'il atteint un "plafond", c'est-à-dire avant que sa tête n'entre en collision avec des obstacles au-dessus de lui.

Le lemming exploseur, quand il explose, doit détruire 14 cases autour de lui à condition qu'elles soient de nature DIRT. Nous avons choisi de détruire les cases décrites sur la figure 5.

En ce qui concerne les lemmings stoppeurs, ils ne peuvent pas stopper quand ils sont dans les airs.

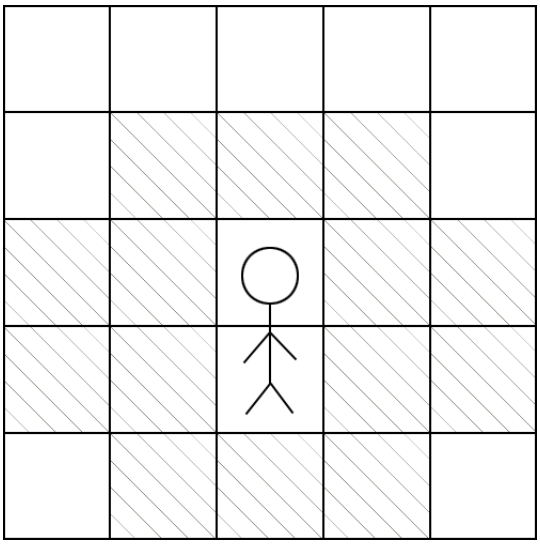

FIGURE 5 – Cases détruites autour d'un lemming exploseur

5 Difficultés rencontrées

Une des difficultés majeures que nous avons rencontré durant ce projet, a été l'ajout d'extensions. En effet lors de l'ajout du mode "Annihilation" par exemple, il a fallut changer la spécification et le contrat de plusieurs fonctions comme par exemple le step du service GameEng qui doit arrêter de créer des lemmings si le mode "Annihilation" a été lancé. L'invariant de minimisation qui indiquait que GameOver() signifiait que le nombre de lemmings créés est égal à la taille de la colonie et que le nombre de Lemmings morts additionné au nombre de lemmings sauvés est égal à la taille de la colonie a aussi été enlevé.

Un autre problème a été rencontré pour la méthode step du service Lemming, au niveau de l'implémentation du contrat (et de l'implémentation du comportement). En effet, certains types de lemming étant cumulables il se pouvait que lors de la vérification des contrats, certains tests non nécessaires étaient quand même effectués (ce qui engendrait des erreurs de post-conditions ou des exceptions). Un exemple est celui du Lemming exploseur qui peut avoir au même moment n'importe quel autre type. Pour un Lemming marcheur exploseur qui doit exploser à l'appel du step courant, si le contrat n'a pas été bien implémenté, les tests de post-conditions pour un Lemming exploseur ET les tests de post-conditions pour un Lemming marcheur vont être effectués. Cela est faux car si le Lemming explose, il ne doit pas avoir effectué le comportement du marcheur. De même, si les tests des post-conditions de marcheur sont avant des ceux de l'exploseur, cele lèvera aussi une erreur car le Lemming doit exploser au tour courant et non pas marcher puis exploser. Ces problèmes ont été résolus grâce à un meilleur ordonnancement des tests à effectuer (dans la spécification et les

contrats).

6 Tests MBT

Nos différents tests basés sur le modèle et effectués grâce à JUnit 4, se déclinent en deux versions : une version utilisant l'implémentation normale et version utilisant l'implémentation buggée. Un exemple de test :

```
/**
 * Objectif 1: init
 *
 * Cas 1_0: init: positif
 *
 * Condition initial:
 * aucune
 *
 * Operation:
 *  init(25,20)
 *
 *  Oracle:
 *  Pas de ContractError &&
 *  getWidth() == 20 &&
 *  getHeight() == 25 &&
 *  isEditing() == true &&
 *  \forall (x,y) getNature(x,y) == Nature.EMPTY
 */
@Test
public void testInit1_0() {
    String test = "Level Test Objectif 1.0";
    try {
        final int width = 25;
        final int height = 20;
        // operation
        level.init(width, height);

        // oracle
        assertion(test+" : La hauteur est incorrecte", level.getHeight() == height);
        assertion(test+" : La largeur est incorrecte", level.getWidth()  == width);
        assertion(test+" : Le mode d'edition n'est pas actif", level.isEditing() == true);
        for (int x = 0; x < level.getWidth() ; x++)
            for (int y = 0; y < level.getHeight() ; y++)
                assertion(test+" : Une des cases n'est pas vide", level.getNature(x, y) == Nature.EMPTY);
    } catch (ContractError e) {
        assertion(test+" : "+e.getMessage(), false);
    }
}
```

FIGURE 6 – Test de la fonction init du service Level, qui doit respecter les contrats

Pour le test de init, il n'y a aucune condition initiale. Les tests Oracles ne doivent pas renvoyer de ContractError et les assertions sur les post-conditions doivent renvoyer vrai.

Seuls les tests MBT du service Level (43 tests en tout) ont été fait entièrement. Pour les autres services, les tests redondants ont été évités. Tous nos tests sont disponibles dans le répertoire "tests" à la racine du projet.

7 Conclusion

Ce projet nous a permis d'avoir un aperçu de la programmation par contrat et du pattern Require-Provide qui se sont avérés très pratiques pour le débuggage et notamment pour la mise en place d'une interface utilisateur où il a seulement fallut lier les différentes fonctions des services aux éléments graphiques. Malgré tout, les tests MBT et les contrats sont assez redondants et une grosse faiblesse de ce modèle se trouve dans l'ajout de nouvelles fonctionnalités qui nécessite une régression. De plus, implémenter tous les tests est une tache fastidieuse car il est difficile de tester tous les cas pouvant se présenter au cours d'une partie, en particulier pour la fonction step du service Lemming qui dépend de beaucoup de paramètres. De notre point de vue, en plus des tests MBT, l'implémentation d'une interface graphique était nécessaire pour pouvoir tester visuellement les choix de spécification et déterminer s'ils sont corrects.