Projet PAF 2021: *Lemmings*

PAF 2020 - Master STL S2

Version du 22/03/2021

Ce document présente le sujet du projet de l'UE PAF 2021 et contient les attendus, la description du projet, une liste des extensions possibles et des consignes pour le rapport et la soutenance. Ce document complète le sujet de partiel.

1 Objectifs et Rendus

1.1 Objectifs

L'objectif principal de ce projet est le développement d'un **jeu** vidéo **sûr**, dans le langage **Haskell**. L'évaluation portera sur ces deux points:

1. programmation **sûre**: les types et les fonctions associées du projet doivent toutes être accompagnées de propositions d'**invariants**, de **pré-conditions**, et de **post-conditions**. Les opérations des entités doivent toutes être testées en utilisant, de façon raisonnablement complète, les **tests unitaires** (*HSpec*) et les **tests basés sur les propriétés** (*Quickcheck*).
2. programmation **fonctionnelle**: le langage Haskell est imposé, l'application doit être écrite dans un style fonctionnel, une attention doit être donnée à la séparation de la logique (évolution de l'état du jeu) et les effets de bords (boucle de jeu, entrée/sortie, affichage). L'utilisation de spécificités du langage Haskell est encouragée: types définis comme sommes d'états remarquables, instanciation de classes de types, utilisation des structures algébriques (`Functor`, `Applicative`, `Monad`).

Le projet sera évalué sur ces deux critères, ce qui veut dire, entre autres;

- que si elles sont appréciées par les correcteurs, les extensions/fonctionnalités superficielles (rendu graphique évolué, gestion du son, interface utilisateur) qui ne contribuent pas aux deux objectifs, ne seront pas prises en compte dans la notation.
- que le projet peut s'éloigner des caractéristiques proposées dans ce document, tant qu'il satisfait les deux critères.

1.2 Rendu

Le projet est à faire en binôme (monôme possible avec autorisation de l'équipe enseignante, trinôme non-autorisé). Le rendu consiste en un projet GitLab contenant un rapport. Les enseignants doivent y être ajoutés comme *maintainer*.

La deadline est le **???** (initialeent mi-mai 2021, en discussion).

2 Description du Jeu

(repris du partiel 2021:)

L'objectif du projet est la modélisation et l'implémentation sûre d'une version (discrète, c'est-à-dire "case par case") du jeu *Lemmings* développé par *DMA Design* et édité par *Psygnosis* en 1991.

Lemmings est un jeu de réflexion en temps réel se déroulant dans des niveaux en deux dimensions vus de côté. Ces niveaux sont composés de cases pleines (terre ou métal) ou vides.

A intervalle régulier, des personnages (appelés **lemmings**) apparaissent d'un point du niveau (l'entrée). Le joueur n'a pas de contrôle direct sur les lemmings, qui marchent dans la même direction jusqu'à rencontrer un obstacle (ils se mettent alors à marcher dans l'autre sens) ou une falaise/un trou (ils tombent et, si la chute n'est pas mortelle, continuent à marcher dans la même direction une fois arrivés sur le sol).

Les niveaux contiennent tous une case de sortie. Si un lemming marche sur la sortie, il disparait (et est considéré comme "sauvé"). Le but du jeu est de sauver le plus de lemmings possible.

Les actions du joueurs prennent une unique forme: le joueur choisit (en temps réel) un lemming et lui assigne une **classe**, qui modifie son comportement: par exemple en le faisant creuser à travers des obstacles ou en lui faisant construire un pont au dessus d'un trou. Le jeu se termine quand un nombre fixé à l'avance de lemmings a été créé et quand il n'y a plus de lemmings dans le niveau, le score du joueur correspond au pourcentage de lemmings sauvés par rapport aux lemmings apparus.

Le projet ne sera pas évalué vis-à-vis de sa similitude avec le jeu de 91 (les critères d'évaluation sont donnés plus haut) ; l'originalité est encouragée. Toutefois, on pourra consulter des documents en ligne décrivant le jeu originel pour s'en inspirer:

- page Wikipédia `https://fr.wikipedia.org/wiki/Lemmings_(jeu_vid%C3%A9o,_1991)`
- preuve de PSPACE-complètude `https://www.semanticscholar.org/paper/Lemmings-is-PSPACE-complete-Viglietta/a42deb4f1c4ee788267865daf72ff867e8ea5f29` (la preuve de NP-complétude existe mais est plus difficile à trouver en ligne)
- une recréation du jeu original dans un navigateur: `https://www.elizium.nu/scripts/lemmings/`
- Lemmings 2 en entier en vidéo `https://www.youtube.com/watch?v=RSByEMmPJw8`

2.1 Fonctionnalités

- le jeu doit permettre le chargement de niveaux importés sous forme de fichiers texte (mais on n'impose pas un standard pour ces fichiers),
- l'état du jeu (niveau, lemmings, monstres éventuels) doit être affiché à l'aide d'une interface graphique (en SDL2), qui représente (entre autres) le niveau en "vue de côté",
- le joueur doit pouvoir sélectionner un lemming et lui affecter une classe à la souris,

- le jeu doit avoir des conditions de victoire et de défaite atteignables, et permettre l'enchainement des niveaux.
- la *boucle de gameplay* du jeu peut être construite à partir de celle du TME6 (en utilisant SDL2).
- le jeu contient au moins plusieurs niveaux simples et implémente des classes différentes, par exemple inspirées par les huit classes du jeu original:
  - le *grimpeur* est capable de monter une paroi verticale,
  - le *flotteur* déploit un parapluie quand il tombe, et ainsi ne meurt pas d'une chute mortelle.
  - le *bloqueur* arrête les autres lemmings comme s'il était un mur,
  - l'*exploseur* s'autodétruit au bout d'un certain temps, vide les cases de terre autour de lui et tue les lemmings pris dans le rayon de l'explosion.
  - le *pelleteur* creuse un tunnel horizontal,
  - le *creuseur* creuse un puit vertical,
  - le *poseur* pose un escalier de 12 marches,
  - le *mineur* creuse en diagonale.

3 Contenu et Extensions

3.1 Consignes

Pour obtenir une **note satisfaisante** à l'évaluation, il faut que le projet respecte les contraintes suivantes:

- implémentation du **jeu de base** (partie 2.1) en Haskell + SDL2 (ou une modification du jeu de contenu équivalent),
- rédaction de **propriétés** pre/post/inv pour les types et les opérations du jeu,
- test systématique des propriétés pre/post/inv incluant **une utilisation pertinente de tests basés sur les propriétés** (*Quickcheck*).
- quelques extensions basiques, afin de proposer un contenu ludique minimal.

Pour obtenir **la note maximale**, le projet doit inclure de multiples extensions parmi celles proposées ou non (c'est encore mieux si elles sont originales), une utilisation pertinente **du test basés sur les propriétés** et **des classes algébriques**.

La motivation des extensions est importante: il faudra montrer ce qu'elles apportent au jeu, mais surtout ce qu'elles apportent au projet: complexité de la rédaction de propositions, élégance du code fonctionnel utilisé, défi pour obtenir des tests complets, ... Une extension qui ne contiendrait pas de contenu "logiciel sûr" (propositions et tests) ou de contenu "programmation fonctionnelle" (utilisation de construction propres à la programmation fonctionnelle ou à Haskell pour la mettre en oeuvre) n'ajouterait rien à l'évaluation.

3.2 Tests Basés sur la propriété

Le projet doit contenir une utilisation pertinente de *QuickCheck*, conforme à celles proposées en TME. Un soin particulier sera apporté à l'efficacité des générateurs (cf. le sujet de TME sur l'utilisation de `suchThat`).

La rédaction et l'utilisation de générateurs complexes pour des types non-triviaux est fortement encouragée. Dans tous les cas, les tests doivent être décrits dans le rapport et la méthode de rédaction des générateurs doit y être explicitée.

3.3 Classes de types

Les projets doivent rendre explicite la maîtrise des classes de types :

- manipulation correcte, élégante et efficace de `Maybe`, `Either`, `IO`, ...,
- instanciation de classes algébriques par des types définis dans le projet (par exemple `instance Monad EtatDuJeu where ...`) et utilisation des bénéfices apportés par l'instanciation (primitives, *do-notations*, compositions, ...)

La programmation "avancée" avec les algébriques (rédaction de *transformers*, ...) est attendue pour une évaluation maximale.

3.4 Extension d'intérêt ludique

Voici quelques extensions possibles renforçant l'intéret du projet en tant que "jeu vidéo". Les extensions originales sont encouragées:

Classes supplémentaires. On implémente de nouvelles classes pour les lemmings, complétement originales ou inspirées des nombreuses classes de *Lemmings 2*. On essaiera de cibler des classes dont l'implémentation est intéressante : soit parce qu'elle demande l'utilisation d'élémentes de programmation fonctionnelle avancés, soit parce qu'elle donne lieu à l'écriture de propriétés complexes.

Cases supplémentaires. On implémente de nouveaux types de case aux niveaux, qui ajoutent de nouvelles possibilité de déplacement, par exemple des échelles, des trampolines, de la roche qui ne peut être creusée que dans une direction, ...

Monstres. On ajoute au jeu des ennemis indépendants qui se déplacent dans les niveaux et qui jouent le rôle d'opposant aux Lemmings (ils les ralentissent ou les tuent, détruisent leur construction, ...) ainsi que des nouvelles classes qui permettent au joueur de gérer ces difficultés. Les monstres peuvent avoir un comportement totalement scripté, aléatoire ou (mieux) réactif.

Continu. Le jeu sort de la contrainte case-par-case, pour ressembler au jeu originel de 91 et nécessite donc la prise en charge de *hitbox* pour détecter les collisions et agir sur l'environnement.

Multijoueur. Le jeu autorise l'intervention simultanée de plusieurs joueurs, sur la même machine ou en ligne.

4 Rapport

Le rapport de projet se compose de quatre parties:

- un **manuel** d'utilisation succint expliquant comment tester le jeu depuis le projet *stack*.
- une liste exhaustive des **propositions** (invariants/pré-conditions/post-conditions) implémentées dans le projet, classées par types / opérations et contenant une brève description de ce que vérifie la propriété.
- une description des **tests** implémentés par le `Spec.hs` du projet.
- un **rapport** proprement dit, qui décrit le projet, les extensions choisies et leur implémentation et qui, en plus, explicite les points importants de l'implémentation, en mettant en évidence du code, des propositions, des tests pertinents (ou des bugs mis découvertes grâce à la méthode de développement sûre).

De manière générale, il est fortement conseillé de consigner dans le rapport (sous la forme d'un paragraphe) toutes les difficultés rencontrées dans le développement du projet et tous les *points forts* (tests basés sur la propriété, utilisation d'algébriques, invariants de type complexe, ...) du projet afin de les mettre en valeur pour l'évaluation.

5 Barême Prévisionnel

Cette suggestion de barême n'est pas opposable, il s'agit d'un exemple possible pour la notation du projet, destiné à faire comprendre les priorités d'évaluation:

- 4 points pour **le rapport**: le rapport doit être clair, lisible, et contenir les informations nécessaires (manuel, propositions, tests). Deplus, le rapport doit présenter suffisamment de points intéressants du développement du projet.
- 4 points pour l'implémentation correcte d'un **jeu minimum** en SDL2.

- 2 points pour **les propositions**: la rédaction des invariants/pre-conditions/post-conditions est systématique, pertinente, et commentée.
- 2 points pour **les tests**: les tests sont complets, correctement structurés, utilisent la puissance du *property based testing* et commentés.
- 1 point pour l'utilisation pertinente **du test basé sur la propriété**.
- 1 points pour l'utilisation pertinente **des classes algébriques**.
- 6 points pour **les extensions**: un nombre suffisant d'extensions ont été choisies, implémentées dans un style fonctionnel et (autant que possible) "haskellien", augmentées de propositions et de tests adéquats (on rappelle qu'une extension qui n'est pas accompagnée de inv/pre/post et de tests n'est pas évaluée). La pertinence, la difficulté et l'orginalité des extensions choisies sont prises en compte.