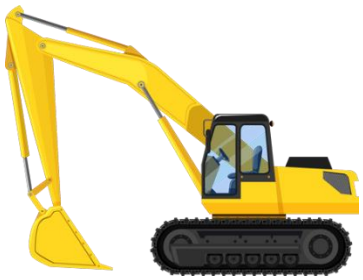
SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257



Projet PAF : Lemmings

BOUCHENAKI Habib 3522424
MEYNIER Florent 3700051
DEMANGEON Romain

Les différents types

Nous avons un type *Coord* avec un constructeur *C int int* qui permet de gérer les coordonnées x, y de nos personnages, ainsi que de tout élément de notre jeu. Nous avons aussi un type *Deplacement* qui permet de savoir dans quelle direction un personnage regarde.

```
data Deplacement = N
                 | G
                 | D
                 | H
                 | B
                 | GH
                 | GB
                 | DH
                 | DB
    deriving(Show, Eq)
```

```
data Coord = C Int Int
    deriving(Show, Eq)
```

Afin de gérer facilement les différentes classes et statuts d'un Lemming, nous avons implémenté un type *Character*, où chaque classe a un constructeur et possède un S*tatus* qui est un type et qui permet de définir le statut d'un Lemming (s'il est Marcheur, Tombeur ou Mort). Tous les *Status* sont représentés par un *State* qui possède une coordonnée de type *Coord*, une direction de type *Deplacement* et une vitesse qui est un entier. Un *Tombeur*, en plus d'un *State*, possède un entier représentant sa distance de chute et un autre *State* représentant son état avant d'être Tombeur. Ainsi on peut ajouter/supprimer des classes très simplement et permet de faciliter les fonctions qui vont gérer nos différents Lemmings.

```
data Character = Lemming Status | Flotteur Status | Grimpeur Status | Pelleteur Status | Creuseur Status
    deriving (Eq, Show)

data Status = Marcheur State | Tombeur State Int State | Mort State
    deriving (Eq, Show)

data State = State { coord :: Coord ,
                     direction :: Deplacement  ,
                     speed :: Int
                   }
                   deriving(Eq, Show)
```

Un niveau est défini par le type *Niveau* qui possède 3 entiers (la hauteur, la largeur, et la taille d'une case), une map avec pour clé une *Coordonnée* et pour valeur *Case* et qui représente la carte du niveau, une map avec pour clé une *Coordonnée* et pour valeur un entier et qui représente les murs qui sont en train d'être cassés, et un type *InfoNiveau* qui permet de stocker les paramètres d'un niveau tel que le délai de spawn entre 2 lemmings, le nombre maximum de lemmings qui peuvent apparaître, le nombre minimal de lemming qui doivent atteindre la sortie pour terminer le niveau, la distance de chute mortelle, la durabilité des murs en terre et pour finir le nom du niveau. Une case peut être vide, en terre, en métal, une entrée, ou une sortie. Le type *Case* permet de différencier chacun de ces états.

```haskell
data Niveau = Niveau { hNiveau :: Int,
                       lNiveau :: Int,
                       sizeCase :: Int,
                       caseNiveau :: Map Coord Case,
                       cassable :: Map Coord Int,
                       infoN :: InfoNiveau
                     }
    deriving (Eq)
```

```haskell
data InfoNiveau = InfoNiveau { spawnDelay :: Int,
                               maxSpawn :: Int,
                               nbExit :: Int,
                               distMortel :: Int,
                               durabilite :: Int,
                               name :: [Char]
                             }
    deriving (Eq)
```

Le type *Game* permet de stocker toutes les informations relatives à une partie. Un attribut *Niveau* y est stocké, une liste des lemmings, un type *InfoGame* et un entier qui permet de savoir quelle classe est séléctionnée. Le type *InfoGame* permet de conserver l'avancement de la partie en y stockant le nombre de lemming qui ont déjà spawn, le nombre qui ont atteint la sortie, ainsi que le *GameStatus* qui permet de définir le statut de la partie.

```haskell
data GameStatus = Pending | Playing | Win | Loose
    deriving (Eq, Show)
```

```haskell
data InfoGame = InfoGame { nbSpawn :: Int,
                           nbExit :: Int,
                           game_status :: GameStatus
                         }
    deriving (Eq, Show)
```

```haskell
data Game = Game { niveau :: Niveau,
                   lemmings :: [Character],
                   infoG :: InfoGame,
                   clique :: Int
                 }
    deriving (Eq, Show)
```

Les lemmings

Les différentes classes de Lemming qui existent sont stockés dans le type *Character*. Chaque classe possède un *Status* qui peut être soit *Marcheur*, soit *Tombeur*, soit *Mort*. Chaque statut possède un état *State* qui permet d'avoir certaines informations concernant un personnage tel que sa position, sa direction et sa vitesse. De plus, un *Tombeur* a un entier représentant la distance de chute et un autre état représentant l'état du Lemming avant sa chute. Ces différents types permettent de simplifier l'ajout d'une classe, mais aussi la gestion des personnages lors de leurs tours de jeu.

Ainsi, on peut faire des actions différentes en fonction de la classe du Lemming, mais aussi en fonction de son statut. Divers getteur permettent de récupérer certaines informations d'un Lemming tel que ses coordonnées où sa vitesse et permet, à certains endroits dans le code, de faire des actions sans se soucier du type de Lemming que l'on traite. La méthode *stillAlive* prend une liste de *Character* et renvoie *True* si au moins un Lemming est vivant, et *False* s'ils sont tous morts.

Les Niveaux

Toutes les informations relatives à un niveau sont stockées dans le type *Niveau*. Une map ayant pour clé une *Coordonnée* et pour valeur une *case* permet de stocker toutes les cases d'un niveau et leur type (si elle est vide, en métal ou terre, si c'est une entrée ou sortie). Tous les niveaux doivent répondre à certaines contraintes :

- Il ne peut y avoir qu'une seule entrée et une seule sortie par niveau et une entrée doit être au dessus d'une case vide et une sortie au dessus d'une case en métal
- Le niveau doit être entièrement fermé par des cases de métal
- Toutes les coordonnées comprises entre 0 et hauteur/largeur sont associées à une case et tout case de notre niveau doit avoir une coordonnée comprise entre 0 et hauteur/largeur.

Pour tester ces contraintes, un invariant a été créé *invNiveau*.

Une case *Terre* peut être cassé par certaines classes de Lemming. Cette case a une durabilité qui est précisé dans le type *InfoNiveau* et un Lemming doit effectuer autant de fois l'action que la durabilté précisée dans le niveau afin de la casser. La map *cassable* ayant pour clé une coordonnée et pour valeur un entier permet de stocker la durabilité restante d'une case. Lorsque la durabilité d'une case tombe à 0 elle est retirée du Niveau (elle devient Vide). Ainsi plusieurs Lemming peuvent travailler ensemble afin d'accélérer la destruction d'une case.

La fonction est *Dure* prend en paramètre une coordonnée et une map de coordonnée et de case et renvoie *True* si la case se trouvant à cette coordonnée est dure (c'est-à-dire qu'un lemming ne peut pas la traverser). Une case est dite "dure" si elle est en *Metal* ou en *Terre*.

Un niveau doit pouvoir être chargé depuis un fichier texte. Pour cela, le type case instancie les classes *Show* et *Read* et permet de convertir un caractère en une *Case* (et inversement). Le type *Niveau* instancie donc aussi les classes *Show* et *Read* et permet de convertir un *Niveau* en chaîne de caractère (et inversement). Un fichier de niveau doit être sous cette forme :

- Première ligne 3 entiers : la hauteur, la largeur et la taille d'une case du niveau
- Deuxième ligne 5 entiers et une chaîne de caractère : le délai d'apparition de chaque Lemming, la quantité maximale de Lemming qui peut apparaître, la quantité nécessaire pour valider le niveau, le durabilité d'un bloc de Terre et pour finir, une chaîne de caractère permettant de conserver le nom du niveau
- Autant de lignes que la hauteur du niveau avec une case représentée par un caractère (un espace représente une case vide, un X du métal, 0 de la terre, E l'entrée et S la sortie, tout autre caractère provoquera une erreur lors du chargement du niveau)

Gestion des coordonnées

Le jeu est composé de carrés d'une taille définie par le niveau lors de son chargement et cette taille est modifiable à tout moment en modifiant le fichier de sauvegarde d'un niveau. Si nous avons une taille de case de 20 et un niveau qui fait 15 cases de longueurs et 10 de hauteurs, notre surface de jeu sera de 300 * 200 pixels. Par exemple, la case aux coordonnées (*C 3 4*) débutera au pixel x = 60 et y = 80 et se terminera au pixel x = 79 et y = 99. Notre personnage quant à lui, se déplace d'un pixel par 1 pixel, il peut donc être sur 4 cases (maximum) simultanément. Les coordonnées des cases du niveau sont donc comprises entre 0 et *h* et 0 et *l* (avec *h* la hauteur et *l* la largeur) et notre personnage peut être compris entre les coordonnées 0 et *taille * h* et entre 0 et *taille * l* (entre 0 et 299 et entre 0 et 199 en reprenant l'exemple précédent). Cette implémentation offre davantage de libertés sur les déplacements et les choses que peuvent faire nos lemmings, mais implique davantage de rigueur quant à la gestion des coordonnées et des collisions.

La détection des murs (sols et plafonds) demande de rajouter des tests et de gérer plusieurs cas de figure. En effet, la plupart du temps, lorsqu'un lemming se déplace, il est sur 2 cases. Pour savoir s'il touche le sol, il faut donc vérifier si l'une des deux cases sous lui est du sol alors que s'il n'est que sur une seule case, il n'y a que la case sous lui à tester. Pour détecter si il y a un obstacle face à lui, il faut faire attention là où regarde le lemming. En effet, s'il regarde à gauche, il suffit de détecter la case qui est juste à côté de lui alors que s'il regarde à droite, il faut peut-être (s'il est sur 2 cases) détecter la case qui n'est pas juste à côté, mais faire un décalage de un (les coordonnées du personnage sont en haut à gauche de sa hitbox, il faut donc que ce soit le côté droit de sa hitbox qui détecte un mur). La fonction *modifyCoord* prend une coordonnée et une direction et renvoie une coordonnée qui a été décalée de un vers la droite si besoin. Pour tester la collision, il faut convertir les coordonnées d'un lemming en coordonnée du niveau. Pour cela, une méthode *persoCoordToMapCoord* qui prend en pramètre une coordonnée et une taille et renvoie une coordonnée qui a été transformée (la fonction *mapCoordToPersoCoord* fait la transformation inverse).

Gestion des évènements

Lors de la partie il est possible de faire plusieurs actions :

- Assigner une classe à un Lemming grâce à la souris
- Redémarrer un niveau
- Quitter le jeu

Les events sont collectés dans la boucle principale du jeu et sont ensuite envoyés à deux modules, le premier va gérer les évènements liés à la souris et le second ceux liés au clavier.

Lorsqu'un événement souris est détecté, on ne conserve que ceux qui résultent d'un clique (gauche ou droit). Puis on récupère ses coordonnées. Si la coordonnée se situe dans le niveau, on appelle la méthode *cliqueMap* et on regarde si ce clique a été effectué sur un ou plusieurs lemmings, si c'est le cas, on assigne la classe qui est sélectionnée à ce/ces lemming(s), sinon, on ne fait rien. Si la coordonnée se situe sous le niveau, on appelle la méthode *CliqueClasse* et la classe correspondant à l'endroit où le clique a été effectué est sélectionnée et peut ensuite être assignée à un Lemming.

Lorsque la touche R est pressée, le niveau actuel est rechargé. Tous les blocs cassés réapparaissent et le niveau recommence au début.

Lorsque la touche "Escape" est pressée, le jeu est quitté.

La gestion de ces deux événements a lieu dans la boucle principale du jeu afin de faciliter leurs actions.

Début de la partie

Lors du lancement du jeu, il y a la possibilité de choisir le mode de jeu. Soit la campagne peut être lancée, auquel cas le joueur choisit par quel niveau il souhaite commencer, soit le joueur peut choisir le niveau qu'il souhaite charger. Lors du lancement du premier niveau, des informations telles que la hauteur et la largeur du jeu ainsi que la taille d'une case sont récupérées et permettent d'initialiser la fenêtre de jeu. Toutes les textures du jeu sont chargées (le fond d'écran, les différents blocs, les lemmings et leurs classes), et la résolution de chaque texture est définie afin qu'elles soient conformes au dimension et taille du jeu. Ainsi, un lemming fait la taille d'une case tout comme les blocs de métal et terre, ainsi que les portes d'entrée et de sortie. Une fois toutes les informations chargées, l'appel à la boucle principale du jeu (*gameLoop*) est fait avec le nombre d'affichage par seconde qui sera fait (ce nombre correspond aussi au nombre de tour de jeu qui sera fait par seconde).

Tour de jeu

A chaque tour de jeu, les événements clavier et souris sont collectés et on traite les événements liés à la souris afin d'appliquer ce à quoi ils se rapportent. La fenêtre de jeu est "nettoyée", c'est-à-dire qu'elle est remise à zéro, et toutes les textures sont affichées une par une. Dans un premier temps, le fond est affiché, puis tous les lemmings (avec leurs classes), puis les murs et entrée et sortie et pour finir le menu de sélection de classe. Une fois l'affichage effectué, on appelle la fonction *gameStepGameManager* afin d'effectuer un tour de jeu à tous nos lemmings mais aussi à en faire apparaître un nouveau si besoin. Une fois le tour des lemmings effectué, on vérifie si la partie est terminée. Pour cela on appelle la méthode *isFinishGame* qui renvoie un tuple de booléen et de statut de jeu. Si la fonction renvoie *True* cela signifie que la partie est terminée. Si la fonction renvoie *False*, cela signifie que la partie doit continuer et on vérifie les entrées claviers, si la touche *R* a été pressée, le niveau actuel redémarre, si la touche *Escape* a été pressée, le jeu est quitté.

Si la partie est terminée, soit elle est gagnée, soit perdue. Pour qu'elle soit gagnée, il faut que le minimum de lemming requis ait atteint la sortie. Pour qu'elle soit perdue, il faut que tous les lemmings soient apparus, et qu'il n'y en ait plus aucun de vivant. Si le niveau est gagné et que c'est un niveau de la campagne, le niveau suivant se lance.

Tour de jeu des lemmings

La méthode *gameStepLemmings* va, pour chaque lemming, effectué son tour de jeu. Pour cela, la méthode *tourCharacter* est appelée. Cette méthode va dans un premier temps regardé si le lemming se situe sur la *Sortie*, si c'est le cas, il est retiré de la liste des lemmings, et le compteur de lemming ayant atteint la sortie est incrémenté de un, sinon on doit effectuer son tour de jeu. On récupère la classe du Lemming, puis on appelle la fonction relative à la classe de ce Lemming. Si c'est un Lemming classique, la méthode *tourLemming* est appelée, si c'est un *Flotteur*, la méthode *tourFlotteur* est appelée, etc … Ces méthodes sont récursives et effectuent *n* tours (avec *n* la vitesse du lemming).

Méthode pour chacune des classes :

- Lemming Classique : aucun comportement spécifique a effectué, la méthode *tourStatus* est appelé
- *Flotteur* : on appelle la méthode *tourStatus*, si elle renvoie un statut *Tombeur*, on réinitialise la distance de chute à 0 et on met sa vitesse de chute à 1, sinon on appelle la méthode *tourStatus*
- *Grimpeur* : si le lemming est un *Marcheur*, on regarde si c'est vide au dessus de sa tête puis on regarde si il y a un mur en face de lui, si c'est le cas, il le monte, s'il arrive au sommet, il monte directement dessus. Si il y a un plafond au dessus de lui, il arrête de monter et arrête de bouger jusqu'à ce qu'il ne soit plus *Grimpeur*. Sinon, dans tous les autres cas, on appelle la méthode *tourStatus*
- *Pelleteur* : si le lemming est un *Marcheur*, on récupère le mur qui est face à lui, si c'est ddee la terre, il met un coup pour tenter de le casser, sinon il appelle la méthode *tourStatus*
- *Creuseur* : si le lemming est un *Marcheur*, on récupère le bloc qui est sous ses pieds, si c'est de la terre, il met un coup pour tenter de le casser, sinon il appelle la méthode *tourStatus*

La méthode *tourStatus* est une méthode qui prend un *Status* et qui effectue un tour de jeu, sans se soucier de la classe du Lemming. La méthode récupère le *Status*, si c'est un *Marcheur* elle appelle la méthode *tourMarcheur*, si c'est un *Tombeur*, elle appelle la méthode *tourTombeur*, sinon c'est un *Mort*, elle renvoie le statut.

La méthode *tourMarcheur* prend un *Status* et un *Niveau* en paramètre et effectue le tour classique d'un *Marcheur* (si le statut est un *Marcheur*). Tout d'abord, on regarde si le lemming a un sol sous ses pieds, si ce n'est pas le cas, le Lemming devient un *Tombeur*, la méthode renvoie donc un *Tombeur*, avec les coordonnées du début de chute, une vitesse de chute à 3, une distance de chute initialisée à 0 et un autre *State* qui est les informations du *Marcheur*. Si il y a du sol sous ses pieds, on regarde si il peut avancer dans la direction où il regarde, si c'est le cas, il avance du case, sinon, on regarde si c'est vide au-dessus du mur auquel il fait face, si c'est ce le cas, on le fait monter sur ce mur, sinon, il se retourne.

La méthode *tourTombeur* prend un *Status* et un *Niveau* en paramètre et effectue le tour classique d'un *Tombeur* (si le statut est un *Tombeur*). Tout d'abord on regarde si il y a du sol sous ses pieds, si c'est le cas, on regarde la distance de chute. Si cette distance est supérieure à celle définit dans *InfoNiveau* du *Niveau*, on renvoie un status *Mort*, sinon le lemming *Status* devient *Marcheur*, on récupère la direction dans laquelle regardait le lemming et sa vitesse avant de devenir *Tombeur*. S'il y a du vide sous ses pieds, il continue de tomber, et on augmente sa distance de chute de 1.

Pour tester si il y a du sol sous les pieds d'un Lemming, la méthode *hasGround* qui prend en paramètre une coordonnée *Coord*, un *Deplacement*, une map ayant pour clé une coordonnée et pour valeur une *Case* (cette map représente le niveau de jeu) et un entier, qui est la taille d'une case est appelée. La coordonnée est une coordonnée de type Personnage, c'est-à-dire qu'elles peuvent aller de de 0 à hauteur/largeur fois la taille d'une case.

- On regarde si cette coordonnée correspond au début d'une case en *x* (*x* modulo la taille = 0), si c'est le cas, cela signifie que le lemming se situe sur une seule case, il n'y a donc qu'une seule case sous ses pieds à vérifier
- Si le lemming regarde à droite et que sa coordonnée *x* se situe sur la dernière coordonnée d'une case (*x* modulo la taille = taille - 1), on doit le faire avancer d'une case, et la fonction renvoie si la nouvelle case sur laquelle il se trouve, à du vide en-dessous. On le fait avancer d'une case sinon, lorsqu'il est sur l'avant dernière coordonnée d'une case, il peut se bloquer ou ne pas détecter du vide s'il n'y a qu'une seule case d'écart entre deux sols
- Sinon dans tous les autres cas, le lemming se situe sur 2 cases. Pour qu'il y ait du vide, il faut que sous l'une des deux cases, il y ait du sol

Propriétés

invNiveau : Invariant du niveau décrit précédemment.

prop_attaqueCase_pre : Pré-condition pour qu'un lemming puisse attaquer une case, il faut que cette case soit en Terre.

prop_spawnCharacter_pre : Pré-condition pour qu'un lemming puisse apparaître, il faut que le nombre de lemming déjà spawn soit inférieure à la quantité max possible.

prop_tour<type>_pre : Pré-condition pour les tours de jeu des différents types de Lemming et Status. *<type>* peut être remplacé par : *Flotteur*, *Grimpeur*, *Pelleteur, Creuseur*, *Marcheur*, *Tombeur*.

Description des tests

Movement :

bougeCoord : Vérification de la modification d'une coordonnée.

Lemming :

stillAliveSpec : Vérification s'il reste des Lemmings vivant.

Map :

initNiveauSpec : Vérifie si un niveau est bien initialisé.

invNiveauSpec : Vérifié l'invariant du Niveau.

invEstDure : Vérifie quels types de blocs sont dures ou non.

attaqueCaseSpec : Vérifie quels types de blocs sont cassables ou non.

GameManager :

spawnCharacterSpec : Vérifie l'invariant sur l'apparition des Lemmings ainsi que si l'apparition de Lemming se passe bien.

hasGroundSpec : Vérifie si il y a du sol à ces coordonnées.

tourTombeurSpec : Vérifie la pré-condition de *tourTombeur* et effectue plusieurs scénarios d'un tour d'un *Tombeur*.

tourMarcheurSpec : Vérifie la pré-condition de *tourMarcheur* et effectue plusieurs scénarios d'un tour d'un *Marcheur*.

tourCreuseurSpec : Vérifie la pré-condition de *tourCreuseur* et effectue plusieurs scénarios d'un tour d'un *Creuseur*.

tourPelleteurSpec : Vérifie la pré-condition de *tourPelleteur* et effectue plusieurs scénarios d'un tour d'un *Pelleteur*.

tourGrimpeurSpec : Vérifie la pré-condition de *tourGrimpeur* et effectue plusieurs scénarios d'un tour d'un *Grimpeur*.

tourFlotteurSpec : Vérifie la pré-condition de *tourFlotteur* et effectue plusieurs scénarios d'un tour d'un *Flotteur*.

tourLemmingSpec : Effectue plusieurs scénarios d'un tour d'un Lemming classique.

gameStepLemmingsSpec : Effectue un tour de jeu des personnages d'une partie.