Service : GameEng
use : Level
types :

Observators :

const level : [GameEng] -> [Level]
const sizeColony :[GameEng] -> int
const spawnSpeed : [GameEng] -> int
tour : [GameEng] -> int
obstacle : [GameEng] * int * int -> bool
pre obstacle(G,i,j) require Level::caseExiste(level(G), i, j)
gameOver : [GameEng] -> bool
pre gameOver(G) require ¬Level::editing(level(G))
score : [GameEng] -> int
pre score(G) require gameOver(G)
nbSauves : [GameEng] -> int
nbCrees : [GameEng] -> int
lemmings : [GameEng] -> set<Lemming>
getLemming : [GameEng] * int -> Lemming
lemmingExiste : [GameEng] * int -> boolean

Constructors :

init : Level * int * int -> [GameEng]
pre : init(L,size,speed) require size > 0 ^ speed > 0 ^ L != null

---

Operators :

runTour : [GameEng] -> GameEng
pre runTour(G) require ¬gameOver(G)

supprimeLemming : [GameEng] * int -> GameEng
pre supprimeLemming(G, id) require lemmingExiste(G, id)

creeLemming : [GameEng] * int * int * int -> GameEng
pre creeLemming(G,id, x, y) require
card(lemmings(G)) < sizeColony(G)
^ ¬lemmingExiste(G, id)
^ ¬gameOver(G)

saveLemming : [GameEng] * int -> GameEng
pre saveLemming(G, id) require
lemmingExiste(G, id)
^ ¬gameOver(G)

stopCreation : [GameEng] -> GameEng
pre stopCreation(G) require
^ ¬gameOver(G)

Observations :

[Invariant]
card(lemmings(G)) <= sizeColony(G)
0 <= nbSauves(G) < sizeColony(G)
nbCrees(G) <= sizeColony(G)
nbCrees(G) min= tour(G)/spawnSpeed(G)
gameOver(G) min= (nbCrees(G) == sizeColony(G)) && (card(lemmings(G)) == 0)
score(G) min= nbSauves(G)/sizeColony(G) * 100
obstacle(G,x,y) min= ¬(Level::nature(level(G), x,y) = EMPTY)
LemmingExiste(G, id) min= (getLemming(G,id) != null)

[init]
level(init(L, size, speed)) = L
sizeColony(init(L, size, speed)) = size
spawnSpeed(init(L, size, speed)) = speed

tour(init(L, size, speed)) = 0
nbSauves(init(L, size, speed)) = 0
nbCrees(init(L, size, speed)) = 0
lemmings(init(L, size, speed)) = {}

[supprimeLemming]
tour(supprimeLemming(G, id)) = tour(G)@pre
nbSauves(supprimeLemming(G, id)) = nbSauves(G)@pre
lemmings(supprimeLemming(G, id)) = lemmings(G)@pre / getLemming(G,id)
getLemming(supprimeLemming(G,id),id) = null
∀ n∈ *Lemming::id(lemmings()) / {id},getLemming(supprimeLemming(G,id),n) = getLemming(G,n)*

[creeLemming]
tour(creeLemming(G, id,x,y)) = tour(G)@pre
nbSauves(creeLemming(G, id)) = nbSauves(G)@pre
lemmings(creeLemming(G, id,x,y)) = lemmings(G)@pre ∪ *{Lemming::init(G, id,x,y)}*
if id = n then getLemming(creeLemming(G,id,x,y),n) = Lemming::init(G, id,x,y)
else getLemming(creeLemming(G,id,x,y),n) = getLemming(G,n)

[saveLemming]
tour(saveLemming(G, id)) = tour(G)@pre
nbSauves(saveLemming(G, id)) = nbSauves(G)@pre +1
lemmings(saveLemming(G, id)) = lemmings(G)@pre / getLemming(G,id)
getLemming(saveLemming(G,id),id) = null
∀ n∈ *Lemming::id(lemmings()) / {num}, getLemming(saveLemming(G,num),n) = getLemming(G,n)*

[runTour]
tour(runTour(G)) = tour(G)@pre + 1
nbSauves(runTour(G)) = n ∈ *{nbSauves(G)@pre, nbSauves(G)@pre +1}*
lemmings(runTour(G)) = {lemmings(G)@pre || ... || {}}
∀ n∈ *Lemming@pre::id(lemmings()), getLemming(runTour(G),n) = getLemming(G,n) si LemmingExiste(G,n)*
getLemming(supprimeLemming(G),n) = null sinon

[stopCreation]
tour(stopCreation(G)) = tour(G)@pre
nbSauves(stopCreation(G)) = nbSauves(G)@pre
lemmings(stopCreation(G)) = lemmings(G)@pre
getLemming(stopCreation(G), id) = getLemming(G,id)