Service : Joueur
use : GameEng
types : ClasseType{ TOMBEUR, MARCHEUR,GRIMPEUR,CREUSEUR }

Observators :

const nbJetons : [Joueur] -> int
const gameEng : [Joueur] --> GameEng
getJetons : [Joueur] * ClasseType -> int
classeTypes : [Joueur] -> set<ClasseType, int>

Constructors :

init : GameEng * int -> Joueur
pre init(G,nbJtn) require nbJtn > 0 ^ G != NULL

Operators :

assignerClasse : [Joueur] * ActivityLemming * Lemming -> Joueur
pre: assignerClasse(J, cl, l) require
getJetons(J, ActivityLemming::classeType(cl)) > 0
^ GameEng::lemmingExiste(gameEng(J), Lemming::getId(l))
^ ¬GameEng::gameOver

reset : [Joueur] -> Joueur
pre: reset(J) require ¬GameEng::gameOver

startGame : [Joueur] -> Joueur
pre: startGame(J) require ¬GameEng::gameOver

annihilation : [Joueur] -> Joueur
pre: annihilation(J) require ¬GameEng::gameOver

Observations:

[invariants]
nbJetons > 0
classeTypes != {}

[init]
gameEng(init(G, n)) = G
nbJetons(init(G, n)) = n
∀ *c∈ classeTypes(); getJetons(init(G, n), ct) = n*
classeTypes(init(G, n)) = {<MARCHEUR,n>,<GRIMPEUR,n>,...}

[assignerClasse]
getJetons(assignerClasse(J, c, l), c) = getJetons(J,c)@pre -1
∀ *c1,c2∈ classeTypes(); c1 != c2; getJetons(assignerClasse(J, c1, l), c2) = getJetons(J,c2)*
classeTypes((assignerClasse(J, c, l)) = {<MARCHEUR,n1>,<GRIMPEUR,n2>,...}

[startGame]
getJetons(assignerClasse(J, c, l), c) = nbJetons
∀ *c1,c2∈ classeTypes(); c1 != c2; getJetons(assignerClasse(J, c1, l), c2) = getJetons(J,c2)@pre*
classeTypes((assignerClasse(J, c, l)) = {<MARCHEUR,n1>,<GRIMPEUR,n2>,...}

[reset]
nbJetons(reset(J)) = nbJetons(J)
∀ *c∈ classeTypes(); getJetons(reset(J), ct) = nbJetons(J)*
classeTypes(reset(J)) = {<MARCHEUR,n1>,<GRIMPEUR,n2>,...}

[annihilation]
getJetons(annihilation(J), c) = getJetons(J, c)@pre
∀ *c1,c2∈ classeTypes(); c1 != c2; getJetons(annihilation(J), c2) = getJetons(J,c2)@pre*

classeTypes((annihilation(J)) = {<MARCHEUR,n1>,<GRIMPEUR,n2>,...}