Examen réparti 1 – Warcraft ^{TM} !
Une solution

Binh-Minh Bui-Xuan

17 mars 2015

Une solution à l'énoncé de l'examen réparti 1 pour le cours CPS 2015.

Villageois

service : Villageois
observators :
const race : [Villageois] → String
const largeur : [Villageois] → int
const hauteur : [Villageois] → int
const force : [Villageois] → int
const vitesse : [Villageois] → double
pointsDeVie : [Villageois] → int
quantiteOr : [Villageois] → int
estMort : [Villageois] → boolean
Constructors :
init : String × int × int × int × double × int → [Villageois]
pre init(race,largeur,hauteur,force,vitesse,pointsVie) **require** race ≠ "" ∧ largeur%2=1 ∧ hauteur%2=1
Operators :
retrait : [Villageois] × int → [Villageois]
pre retrait(V,s) **require** ¬estMort(V) ∧ s>0
Observations :
[invariants]
estMort(V) $\stackrel{\min}{=}$ pointsDeVie(V) ≤ 0
[init]
race(init(s,l,h,f,v,p))=r
largeur(init(s,l,h,f,v,p))=l
hauteur(init(s,l,h,f,v,p))=h
force(init(s,l,h,f,v,p))=f
vitesse(init(s,l,h,f,v,p))=v
pointsDeVie(init(s,l,h,f,v,p))=p
quantiteOr(init(s,l,h,f,v,p))=0
[retrait]
pointsDeVie(retrait(V,s))=pointsDeVie(V) -s
quantiteOr(retrait(V,s))=quantiteOr(V)

Mine

service : Mine
observators :
const largeur : [Mine] → int
const hauteur : [Mine] → int
orRestant : [Mine] → int
estAbandonnee : [Mine] → boolean
estLaminee : [Mine] → boolean
abandonCompteur : [Mine] → int
Constructors :
init : int × int → [Mine]
pre init(largeur,hauteur) **require** largeur%2=1 ∧ hauteur%2=1
Operators :
retrait : [Mine] × int → [Mine]
pre retrait(M,s) **require** ¬estLaminee(M) ∧ s>0
acceuil : [Mine] → [Mine]
pre acceuil(M) **require** ¬abandoned(M)
abandoned : [Mine] → [Mine]
pre abandoned(M) **require** ¬acceuil(M)
Observations :
[invariants]
estLaminee(M) $\stackrel{\min}{=}$ orRestant(M) ≤ 0
estAbandonnee(M) $\stackrel{\min}{=}$ abandonCompteur $= 51$
$0 \leq$abandonCompteur(M)≤ 51
[init]
largeur(init(l,h))=l
hauteur(init(l,h))=h
orRestant(init(l,h))=51
abandonCompteur(init(l,h))=51
[retrait]
orRestant(retrait(M,s))=orRestant(M)-s
abandonCompteur(retrait(M,s))=abandonCompteur(M)
[acceuil]
orRestant(acceuil(M))=orRestant(M)
abandonCompteur(accueil(M))=0
[abandoned]
orRestant(abandoned(M))=orRestant(M)
abandonCompteur(abandoned(M))=abandonCompteur()+1

HotelVille

service : HotelVille
observators :
const largeur : [HotelVille] → int
const hauteur : [HotelVille] → int
orRestant : [HotelVille] → int
estAbandonnee : [HotelVille] → boolean
estLaminee : [HotelVille] → boolean
abandonCompteur : [HotelVille] → int
Constructors :
init : int × int → [HotelVille]
pre init(largeur,hauteur) **require** largeur%2=1 ∧ hauteur%2=1
Operators :
retrait : [HotelVille] × int → [HotelVille]
pre retrait(H,s) **require** ¬estLaminee(H) ∧ s>0
Observations :
[invariants]
estLaminee(H) $\stackrel{\min}{=}$ orRestant(H) ≤ 0
[init]
largeur(init(l,h))=l
hauteur(init(l,h))=h
orRestant(init(l,h))=51
estAbandonnee(init(l,h))=true
[retrait]
orRestant(retrait(H,s))=orRestant(H)-s
estAbandonnee(retrait(H,s))=false

service : MoteurJeu
types : enum RESULTAT{GAGNE, PERDU},
enum COMMANDE{RIEN, DEPLACER, ENTRERMINE, ENTRERHOTELVILLE}
observators :
const largeurTerrain : [MoteurJeu] → int
const hauteurTerrain : [MoteurJeu] → int
const maxPasJeu : [MoteurJeu] → int
pasJeuCourant : [MoteurJeu] → int
estFini : [MoteurJeu] → boolean
resultatFinal : [MoteurJeu] → RESULTAT
pre resultatFinal(M) **require** estFini(M)
const numeroesVillageois : [MoteurJeu] → Set<int>
getVillageois : [MoteurJeu] × int → Villageois
pre getVillageois(M,num) **require** num ∈ numeroesVillageois(M,num)
positionVillageoisX : [MoteurJeu] × int → int
pre positionVillageoisX(M,num) **require** num ∈ numeroesVillageois(M,num)
positionVillageoisY : [MoteurJeu] × int → int
pre positionVillageoisY(M,num) **require** num ∈ numeroesVillageois(M,num)
const numeroesMine : [MoteurJeu] → Set<int>
getMine : [MoteurJeu] × int → Mine
pre getMine(M,num) **require** num ∈ numeroesMine(M,num)
const positionMineX : [MoteurJeu] × int → int
pre positionMineX(M,num) **require** num ∈ numeroesMine(M,num)
const positionMineY : [MoteurJeu] × int → int
pre positionMineY(M,num) **require** num ∈ numeroesMine(M,num)
hotelDeVille : [MoteurJeu] → HôtelVille
const positionHotelVilleX : [MoteurJeu] → int
const positionHotelVilleY : [MoteurJeu] → int
peutEntrer : [MoteurJeu] × int × int → boolean
pre peutEntrer(M,numVillageois,numMine) **require** numVillageois ∈ numeroesVillageois(M,numVillageois)
∧ numMine ∈ numeroesMine(M,numMine)
peutEntrerHotelVille : [MoteurJeu] × int → boolean
pre peutEntrerHotelVille(M,numVillageois) **require** numVillageois ∈ numeroesVillageois(M,numVillageois)
Constructors :
init : int × int × int → [MoteurJeu]
pre init(largeur,hauteur,maxPas) **require** largeur≥ 600 ∧ hauteur≥ 400 ∧ maxPas≥ 0
Operators :
pasJeu : [MoteurJeu] × COMMANDE × int × int → [MoteurJeu]
pre pasJeu(M,commmand,numVillgeois,argument) **require**
$$\begin{cases} \neg\text{estFini(M)} \\ \text{command=DEPLACER} \Rightarrow 0 \leq \text{argument} \leq 360 \\ \text{command=ENTRERMINE} \Rightarrow \begin{cases} \text{argument} \in \text{numeroesMines(M)} \\ \text{peutEntrer(M,numVillageois,argument)} \end{cases} \\ \text{command=ENTRERHOTELVILLE} \Rightarrow \text{peutEntrerHotelVille(M,numVillageois)} \end{cases}$$
Observations :
[invariants]
$0 \leq$ pasJeuCourant(M) $\leq$ maxPasJeu(M)
estFini(M) $\stackrel{\min}{=}$ HotelVille::orRestant(hotelDeVille(M)) $\geq 1664 \vee$ pasJeuCourant(M)=maxPasJeu(M))
resultatFinal(M)=GAGNE ⇔ HotelVille::orRestant(hotelDeVille(M)) ≥ 1664
peutEntrer(M,numVillageois,numMine) $\stackrel{\min}{=}$ distance(positionVillageoisX(M,numVillageois),positionVillageoisY(M,numVillageois),
positionMineX(M,numMine),positionMineY(M,numMine)) ≤ 51
peutEntrerHotelVille(M,numVillageois) $\stackrel{\min}{=}$ distance(positionVillageoisX(M,numVillageois),positionVillageoisY(M,numVillageois),
positionHotelVilleX(M),positionHotelVilleY(M)) ≤ 51
[init]
maxPasJeu(init(l,h,m))=m
pasJeuCourant(init(l,h,m))=0
etc...
[pasJeu]
pasJeuCourant(pasJeu(M,c,numVillageois,arg))=pasJeuCourant(M) $+1$
getMine(pasJeu(M,c,numVillageois,arg),numMine) $= \begin{cases} \text{Mine::abandoned(getMine(M,numMine))} & \text{si c}\neq\text{ENTRERMINE} \vee \text{arg}\neq\text{numMine} \\ \text{Mine::acceuil(getMine(M,numMine))} & \text{sinon} \end{cases}$
// il reste à faire pour getVillageois quelque chose de similaire, puis des implications entre getMine et getVillageois