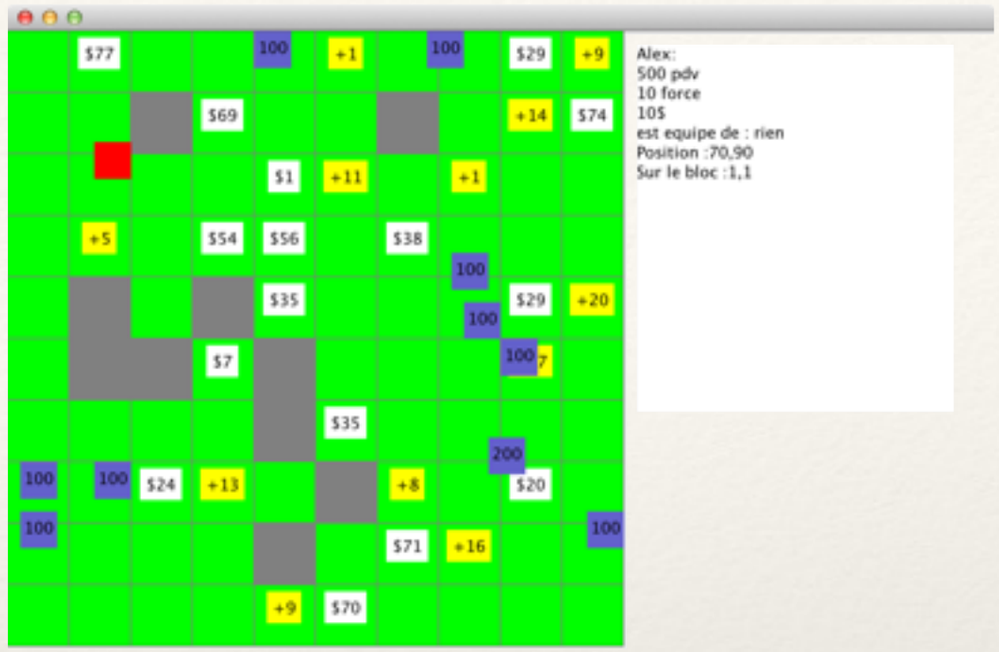


River City Ransom

Steven Varoumas
Béatrice Carré

Problématique

- Spécification : reconnaître les éléments logiques d'un programme à partir d'une définition du jeu River City Ransom.
- Tests : Décrire et implémenter les tests pour le respect du contrat.
- Implémentation par contrat : développer les services et conditions soulevés, en respectant le squelette étudié.

Spécification (1)

- problèmes d'interprétation personnelle
- combler le manque d'éléments
- rester cohérents
- importante notion de référentiel pour chaque service
- représentation d'une liste difficile
- représentation de la notion de temps difficile

Spécification (2)

voir spec.pdf

Tests (1)

- Pour n conditions sur chaque opération => 2^n tests
- Théorie : tous les tests réussissent => une application sans bug !
- Réalité : il faut aussi que l'implémentation respecte ce qui est demandé.

Tests (2)

Service : Personnage

Cas de test : Personnage::RetraitVieWorking
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
Opération : P0 =def retraitPdV(personnage, 3)
Oracle :
argent(P0) = 7 = (10-3)

Cas de test : Personnage::RetraitVieFailing
CI :personnage = init("Alex", 10, 10, 10, 10, 100, 100)
Opération : P0 =def retraitPdV(personnage, -5);
Oracle :
-5 < 0
Une exception est levée

Cas de test : Personnage::RetraitArgentWorking
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
Opération : P0 =def retraitArgent(personnage, 3)
Oracle :
pointsDeVie(P0) = 7 = (10-3)

Cas de test : Personnage::RetraitArgentFailing
CI :personnage = init("Alex", 10, 10, 10, 10, 100, 100)
Opération : P0 =def retraitArgent(personnage, -5);
Oracle :
-5 < 0
Une exception est levée

Cas de test : Personnage::RetraitArgentFailing
CI :personnage = init("Alex", 10, 10, 10, 10, 100, 100)
Opération : P0 =def retraitArgent(personnage, 108);
Oracle :
108 > 100
Une exception est levée

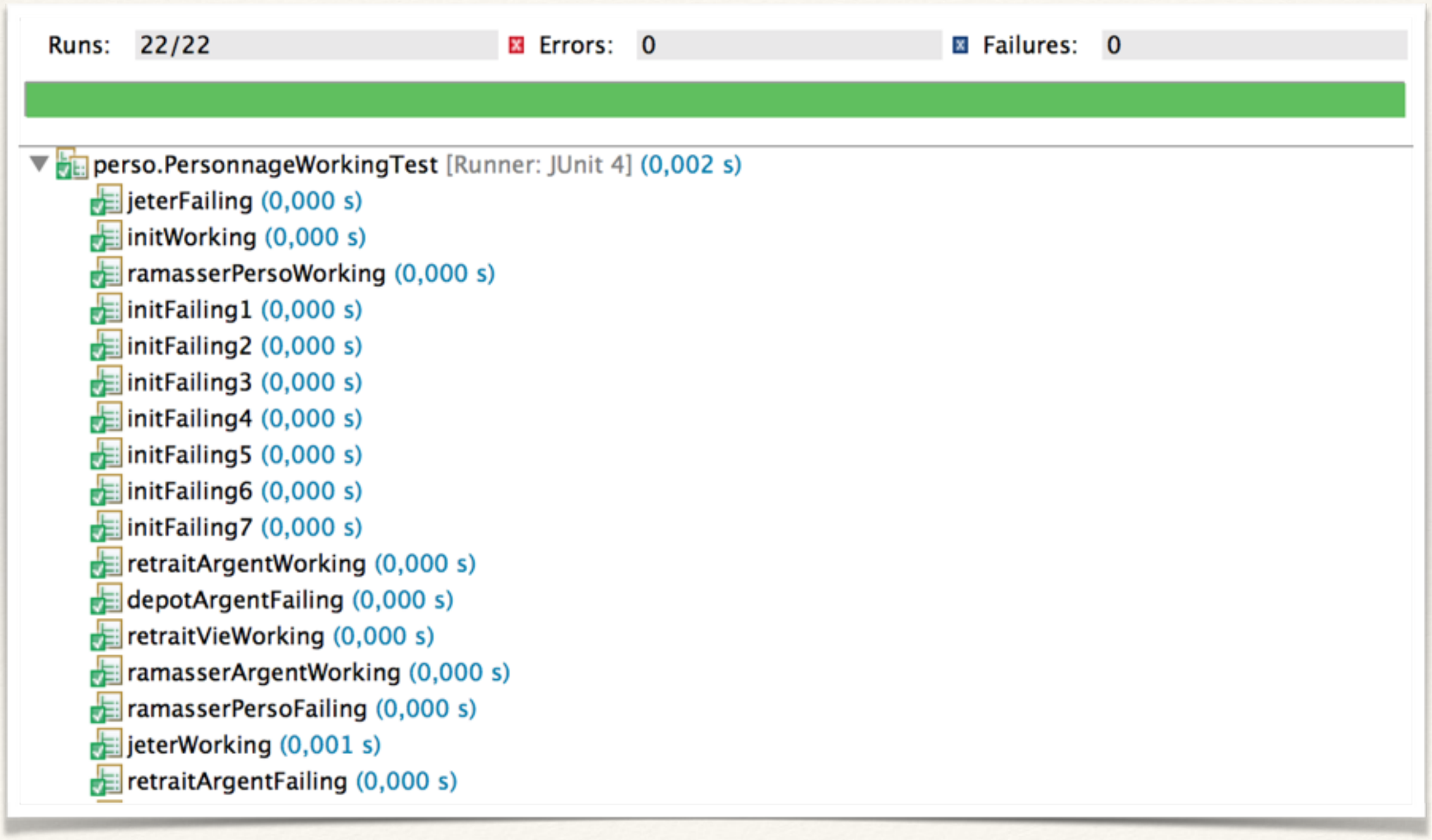
Cas de test : Personnage::ramasserPersoWorking
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
perso2 = init("Ryan", 10, 10, 10, 10, 100, 100)
Opération : P0 =def ramasserPerso(personnage, perso2)
Oracle :
persoEquipe(P0) = perso2

Cas de test : Personnage::ramasserPersoFailing1
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
perso2 = init("Ryan", 10, 10, 10, 10, 100, 100)
personnage = personnage.retraitPdV(10000);
Opération : P0 =def ramasserPerso(personnage, perso2)
Oracle :
estVaincu(P0) = true
Une exception est levée

Cas de test : Personnage::jeterWorking
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
obj = Objet::init("arme",10,0)
personnage = ramasserObjet(personnage,obj)
Opération : P0 =def jeter(personnage)
Oracle :
persoEquipe(P0) = null
force(P0) = 100
objetEquipe(P0) = false

Cas de test : Personnage::jeterFailing1
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
Opération : P0 =def jeter(personnage)
Oracle :
estEquipeObjet(personnage) = false
estEquipePerso(personnage) = false
Une exception est levée

Tests (3)


Runs: 22/22
Errors: 0
Failures: 0
perso.PersonnageWorkingTest [Runner: JUnit 4] (0,002 s)
jeterFailing (0,000 s)
initWorking (0,000 s)
ramasserPersoWorking (0,000 s)
initFailing1 (0,000 s)
initFailing2 (0,000 s)
initFailing3 (0,000 s)
initFailing4 (0,000 s)
initFailing5 (0,000 s)
initFailing6 (0,000 s)
initFailing7 (0,000 s)
retraitArgentWorking (0,000 s)
depotArgentFailing (0,000 s)
retraitVieWorking (0,000 s)
ramasserArgentWorking (0,000 s)
ramasserPersoFailing (0,000 s)
jeterWorking (0,001 s)
retraitArgentFailing (0,000 s)

Choix d'implémentation

- ramasser => 3 méthodes différentes
- 1 objet par bloc => jeter un objet que sur bloc vide
- action gangster => commande aléatoire
- ramasser un perso = équipé comme un objet donc invisible
- argent n'est pas un objet équipé ! => peut se récolter autant que l'on veut, même si équipé de quelque chose
- Les blocs sont en 2D au sol, mais les personnages peuvent sauter sur le troisième axe.
- observateur estVisible() pour afficher ou non des gangsters/persos

Démo

Suite et fin

- Gestion du saut
- Le gel
- Le recul dans la frappe
- Le rapport