MI047 Composants – Rapport de projet – Bomberman

# Service Bloc

// [const]

TYPE getType();

TRESOR getTresor();

// [invariants]

// \pre init(m)

// \post getType() = type

// \post getTresor() = tresor

**void** init(TYPE type, TRESOR tresor);

Classe de Test – BlocAbstractTest

POST-CONDITION

Objectif 1 : init

Condition initiale : aucune

Opération : L1.1 = init(TYPE.MURBRIQUE, TRESOR.DETONATOR)

Oracle : invariants vrais et getType(L1 .1) = TYPE.MURBRIQUE et getTresor(L1.1)=TRESOR.DETONATOR

# Service Terrain

// const

**public** **int** getNombreColonnes();

// const

**public** **int** getNombreLignes();

// \pre 1 <= x <= getNombreColonnes()

// \pre 1 <= y <= getNombreLignes()

**public** BlocService getBloc(**int** x, **int** y);

// [invariants]

// \pre x >= 1

// \pre y >= 1

// \post getNombreColonne() = x

// \post getNombreLigne() = y

// \post getBloc(x, y) != NULL

// \post Bloc:getType(getBloc(x, y)) = TYPE.VIDE

// \post Bloc:getTresor(getBloc(x, y)) = TRESOR.RIEN

**public** **void** init(**int** x, **int** y);

Classe de Test – TerrainAbstractTest

PRECONDITIONS

1. Objectif 1 : init
   1. cas positif

condition initiale : aucune

opération : L1.1=init(3, 5)

oracle : pas de message d’erreur

* 1. cas négatif

condition initiale : aucune

opération : L1.1=init(-2, 5)

oracle : nombre de colonnes négatif

1. Objectif 2 : getBloc

2.1 cas positif

Condition initiale : I2.1 = init(3, 5)

Opération : L2.1 = getBloc(I2.1, 2, 2)

Oracle : pas de message d’erreur

2.2 cas négatif

Condition initiale : I2.1 = init(3,5)

Opération : getBloc(I2.2, 4, 5)

Oracle : bloc hors terrain

POST-CONDITION

1. Objectif 3 : init

Condition initiale : aucune

opération : L3.1 = init(3, 5)

oracle : invariants vrais et getNombreColonnes(L3.1) = 3 et getNombreLignes(L3.1) = 5 et getBloc(L3.1, 3, 5) != NULL et Bloc :getType(getBloc(L3.1), 3, 5) = TYPE.VIDE et Bloc :getTresor(getBloc(L3.1), 3, 5) = TRESOR.RIEN

# Service MoteurJeu

// const

**int** getMaxPasJeu();

**int** getPasJeuCourant();

// [invariants]

// 0 <= getPasjeuCourrant(M) <= getMaxPasJeu(M)

// \pre init(m) require m>=0

// \post getMaxPasJeu(init(m)) = m

// \post getPasJeuCourrant(init(m)) = 0

**void** init(**int** m);

// \pre pasJeu(M, cmd) require getPasJeuCourant(M) <= getMaxPasJeu(M)

// \post getPasjeuCourrant(pasJeu(M, cmd)) = getPasjeuCourrant(M)+1

**void** pasJeu(Commande cmd);

Classe de Test – MoteurJeuAbstractTest

PRECONDITIONS

1. Objectif 1 : init
   1. cas positif

condition initiale : aucune

opération : L1.1=init(5)

oracle : pas de message d’erreur

* 1. cas négatif

condition initiale : aucune

opération : L1.1=init(-1)

oracle : nombre de pas de jeu négatif

// on ne peut pas tester pasJeu

POST-CONDITION

1. Objectif 2 : init

Condition initiale : aucune

opération : L2.1 = init(5)

oracle : invariants vrais et getMaxPasJeu(L2.1) = 5 et getPasJeuCourrant(L2.1) = 0

1. Objectif 3 : pasJeu

Condition initiale : I3.1 = init(5)

Opération : L3.1 = pasJeu(init(5), COMMANDE.RIEN)

Oracle : invariants vrais et getPasJeuCourrant(L3.1) = getPasJeuCourrant(I3.1) + 1

INVARIANTS

Objectif 4 : getPasJeuCourrant

Condition initiale : I4.1 = pasjeu(init(5), COMMANDE.RIEN)

Opération : L4.1 = pasjeu(I4.1, COMMANDE.RIEN)

Oracle : invariants vrais

# Service Bombe

// [const]

**int** getNumero();

// [const]

**int** getX();

// [const]

**int** getY();

// [const]

**int** getAmplitude();

**int** getCompteARebours();

**boolean** vaExploser();

// [invariants]

// 0 <= getCompteARebours(B) <= 10

// vaExploser(B) = (getCompteARebours(B) = 0)

// \pre 3 <= amplitude <= 11

// \post getNumero() = num

// \post getX() = x

// \post getY() = y

// \post getAmplitude() = amplitude

// \post getCompteARebours() = 10

**void** init(**int** num, **int** x, **int** y, **int** amplitude);

// \pre getCompteARebours() != 0

// \post getNumero() = getNumero()@pre

// \post getX() = getX()@pre

// \post getY() = getY()@pre

// \post getAmplitude() = getAmplitude()@pre

// \post getCompteARebours() = getCompteARebours()@pre - 1

**void** diminuerCompteARebours();

Classe de Test – BombeAbstractTest

PRECONDITIONS

1. Objectif 1 : init
   1. cas positif

condition initiale : aucune

opération : L1.1=init(1, 3, 5, 6)

oracle : pas de message d’erreur

* 1. cas négatif

condition initiale : aucune

opération : L1.1=init(1, 3, 5, 12)

oracle : mauvaise amplitude

1. Objectif 2 : diminuerCompteARebours

2.1 cas positif

Condition initiale : I2.1 = init(1, 3, 5, 6)

Opération : L2.1 = diminuerCompteARebours (I2.1)

Oracle : pas de message d’erreur

2.2 cas négatif

Condition initiale : I2.1 = init(1, 3, 5, 0)

Opération : getBloc(I2.2)

Oracle : compte à rebours nul

POST-CONDITION

1. Objectif 3 : init

Condition initiale : aucune

opération : L3.1 = init(1, 3, 5, 6)

oracle : invariants vrais et getNumero(L3.1) = 1 et getX(L3.1) = 3 et getY(L3.1) = 5 et getAmplitude(L3.1) = 6 et getCompteARebours() = 10

1. Objectif 4 : diminuerCompteARebours

Condition initiale : I4.1 = init(1, 3, 5, 6)

Opération : L4.1 = diminuerCompteARebours(I4.1)

Oracle : invariants vrais et t getNumero(L4.1) = 1 et getX(L4.1) = 3 et getY(L4.1) = 5 et getAmplitude(L4.1) = 6 et getCompteARebours() = 9

INVARIANTS

1. Objectif 5 : invariant 1 - 0 <= getCompteARebours(B) <= 10

Condition initiale : I5.1 diminuerCompteARebours(init(1, 3, 5, 6))

Opération : L5.1 = diminuerCompteARebours(I5.1)

Oracle : invariants vrais

1. Objectif 6 : invariant 2 - vaExploser(B) = (getCompteARebours(B) = 0)

6.1 cas A

Condition initiale : I6.1 = diminuerCompteARebours(diminuerCompteARebours(diminuerCompteARebours(diminuerCompteARebours(diminuerCompteARebours(diminuerCompteARebours(diminuerCompteARebours(diminuerCompteARebours(diminuerCompteARebours init(4, 3, 5, 6))))))))))

Opération : L6.1 = diminuerCompteARebours(I6.1)

Oracle : vaExploser = true

6.2 cas B

Condition initiale : I6.1 = init(4, 3, 5, 6)

Opération : L6.1 = diminuerCompteARebours(I6.1)

Oracle : vaExploser = false

# Service Vilain

// const

**public** TYPE\_VILAIN getType();

**public** **int** getX();

**public** **int** getY();

// invariants

// \post getType() = type

// \post getX() = x

// \post getY() = y

**public** **void** init(TYPE\_VILAIN type, **int** x, **int** y);

// déplace aléatoirement le vilain en fonction des murs qui l'entourent

// \pre Il existe m app {murNord, murSud, murOuest, murEst} tq m = VIDE

// \post if getX() = getX()@pre then getY() = getY()@pre +/- 1

// \post if getY() = getY()@pre then getX() = getX()@pre +/- 1

**public** **void** deplacer(TYPE murNord, TYPE murSud, TYPE murOuest, TYPE murEst);

Classe de test : VilainAbstractTest.java

# Service Bomberman

// [Observateurs]

**public** **int** getX();

**public** **int** getY();

**public** SANTE getSante();

**public** **int** getForceVitale(); // Fire-Up

**public** **int** getNbrBombe();

**public** **int** getNbrBombeMax(); // Bomb-Up

**public** **boolean** canOverPassWall(); // Wall-Pass

**public** **boolean** canOverPassBomb(); // Bomb-Pass

**public** **boolean** isInvicible(); // FireSuit

// [invariants]

// 3 <= getForceVitale() <= 11

// 1 <= getX()

// 1 <= getY()

// [init]

// \pre x > 0

// \pre y > 0

// \post getX() = x

// \post getY() = y

// \post getSante() = SANTE.VIVANT

// \post getForceVitale() = 3

// \post getNbrBombe() = 1

// \post getNbrBombeMax() = 1

// \post canOverPassWall() = false

// \post canOverPassBomb() = false

// \post isInvicible() = false

**public** **void** init(**int** x, **int** y);

// [operateurs]

// \pre getSante() = SANTE.VIVANT

// \post getX() = getX()@pre

// \post getY() = getY()@pre + 1

// \post getSante() = getSante()@pre

// \post getForceVitale() = getForceVitale()@pre

// \post getNbrBombe() = getNbrBombe()@pre

// \post getNbrBombeMax() = getNbrBombeMax()@pre

// \post canOverPassWall() = canOverPassWall()@pre

// \post canOverPassBomb() = canOverPassBomb()@pre

// \post isInvicible() = isInvicible()@pre

**public** **void** monter();

**public** **void** descendre();

**public** **void** gauche();

**public** **void** droite();

// Traite les Power-Up rencontrés

**public** **void** addPowerUp(TRESOR powerUp);

// \pre getSante() = SANTE.VIVANT

**public** **void** meurt();

// le bomberman vient de créer une bombe avec un numéro

**public** **void** bombeCreee(**int** num);

// la bombe numéro num vient d'exploser. S'il la contient, il la retire de se liste

**public** **void** bombeExplosee(**int** num);

INVARIANTS

Objectif 1 : 3 <= getForceVitale() <= 11

Condition initiale : aucune

Opérations : L1.1 = init(1,1)

Oracle : L1.1.getForceVitale() >=3 et L1.1.getForceVitale() <= 11

Objectif 2 : getX()

Condition initiale : aucune

Opérations : L2.1 = init(1,1)

Oracle : L2.1.getX() >= 1

Objectif 3 : getY()

Condition initiale : aucune

Opérations : L3.1 = init(1,1)

Oracle : L3.1.getY() >=1

PRE-CONDITIONS

Objectif 4 : init

Cas positif 4.1

Condition initiale : aucune

Opérations : L4.1 = init(1, 1)

Oracle : pas de message d’erreur

Cas négatif 4.2

Condition initiale : aucune

Opérations : L4.2 = init(1,1)

Oracle : En dehors du terrain

POST-CONDITIONS

Objectif 5 : getSante() = SANTE.VIVANT

Condition initiale : aucune

Opération : L5.1 = init(1,1).getSante()

Oracle : invariant vrai et L5.1 = getSante()@pre

Objectif 6 : getForceVitale() = 3

Condition initiale : aucune

Opérations : L6.1 = init(1,1).getForceVitale()

Oracle : invariant vrai et L6.1 = 3

Objectif 7 : getNbrBombe() = 1

Condition initiale : aucune

Opérations : L7.1 = init(1,1).getNbrBombe()

Oracle : invariant vrai et L7.1 = 1

Objectif 8 : getNbrBombeMax() = 1

Condition initiale : aucune

Opérations : L8.1 = init(1,1).getNbrBombeMax()

Oracle : invariant vrai et L8.1 = 1

Objectif 9 : canOverPassWall()

Condition initiale : aucune

Opérations : L9.1 = init(1,1).canOverPassWall()

Oracle : invariant vrai et L5.1 = false

Objectif 10 : canOverPassBomb()

Condition initiale : aucune

Opérations : L10.1 = init(1,1).canOverPassWall()

Oracle : invariant vrai et L10.1 = false

Objectif 11 : isInvicible()

Condition initiale : aucune

Opérations : L7.1 = init(1,1).isInvicible

Oracle : invariant vrai et L11.1isInvincible() = false