

# Projet CPS : Bomberman

Alexis Deluze & Clément Barbier

12 avril 2013

## Sommaire

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Spécification</b>	<b>2</b>
<b>3</b>	<b>Tests MBT</b>	<b>3</b>
<b>4</b>	<b>Scénarios utilisateurs</b>	<b>3</b>
4.1	Scénario 1 . . . . .	3
4.2	Scénario 2 . . . . .	3
4.3	Scénario 3 . . . . .	4
<b>5</b>	<b>Traces de l'implémentation buggée</b>	<b>4</b>
<b>6</b>	<b>Conclusion</b>	<b>4</b>

# 1 Introduction

Le but du projet est de réaliser un clone simplifié du jeu **Bomberman** (de 1992) en s'appuyant sur le langage de spécification de services vu en cours. Le jeu sera ensuite soumis à une série de tests pour s'assurer de la robustesse et de la justesse du code de l'implémentation. Ces tests seront écrits en *JUnit*. Pour montrer l'efficacité des tests, le jeu sera implémenté deux fois, avec d'un côté une implémentation respectant les contrats des services (et passant les tests), et de l'autre une implémentation dans laquelle des bugs ont été volontairement insérés. Cette seconde implémentation ne devra pas passer les tests et affichera des erreurs lorsque les contrats seront violés.

## Cibles du fichier build.xml :

Compilation du projet : **ant compile**

Lancement du jeu : **ant run**

Lancement du jeu buggée : **ant runbug**

Lancement des tests sur le jeu : **ant test**

Lancement des tests sur le jeu buggée : **ant testbug**

Création de la distribution : **ant dist**

Nettoyage du projet : **ant clean**

*Remarque 1* : le lancement du jeu via **run** et **runbug** se fait sans l'IHM. Les actions du joueur sont alors aléatoires. Pour tester le jeu avec l'IHM, il faut l'importer dans *eclipse*.

*Remarque 2* : il n'est pas possible de lancer le jeu ou les tests sans passer par les contrats. Certains objets sont créés dynamiquement dans notre programme (par exemple les Bombes), il aurait donc fallu faire 4 implémentations (normale, normale + contrats, buggée, buggée + contrats), ce que nous n'avons pas fait par souci de clarté du projet.

# 2 Spécification

Pour des raisons de lisibilité des spécifications, nous avons préféré laisser ces dernières dans les fichiers texte séparés, dans le dossier **Spécifications** à la racine du projet.

Afin de définir au mieux nos services, nous avons choisi de réaliser un prototype du jeu avant d'écrire les spécifications. Cela nous a permis d'identifier rapidement les fonctions principales du programme et surtout d'avoir des services auxquels nous n'avons jamais eu besoin de retoucher au fur et à mesure que le projet avançait. Le prototype a ensuite évolué avec l'écriture des spécifications, pour finir en programme complet avec une interface homme/machine illustrant le terrain de jeu.

Par souci de clarté du code, notamment dans le service *Moteur*, nous avons eu recours à des fonctions privées (et donc non visibles dans les spécifications). Ces fonctions ne sont appelées qu'une fois par tour de jeu, et uniquement dans la fonction principale du moteur de jeu : **step**.

### 3 Tests MBT

Les tests ont été directement implémentés en JUnit et se lancent via la commande `ant test` à la racine du projet. Au vue du nombre de tests écrits, il n'était pas raisonnable de les recopier dans le rapport. Tous les tests sont dans le package `tests` du projet (classes `AbstractServiceTest.java`). Les tests sont lancés sur les deux implémentations contractualisées du jeu (avec et sans bugs) et le résultat est enregistré au format textuel dans des fichiers nommés `test_Service<_bug>.txt` dans le dossier `bomberman`.

### 4 Scénarios utilisateurs

Plusieurs données ne sont pas testables dans nos scénarios utilisateur du fait des données aléatoires générées par le jeu (comme le placement des murs ou les actions du Kidnappeur et des Vilains).

#### 4.1 Scénario 1

**Nom :** scénario1

**Objectif :** Une partie est lancée le joueur ne peut aller ni à gauche, ni en haut, mais il peut aller à droite et en bas. Le joueur reviens à sa position initiale.

**Préambule :** Moteur de jeu initialisé avec 500 pas, puis lancé.

M1 = init(500)

**Contenu :** Parcours = gauche,haut,droite,gauche,bas,haut

Sc1 = step(step(step(step(step(M1, Action.Gauche), Action.Haut), Action.Droite), Action.Gauche), Action.Bas), Action.Haut)

**Oracle :**

inv 1 : getStepRestants() = 494  
inv 2 : Joueur : :getLigne(getJoueur()) = 2  
inv 3 : Joueur : :getColumnne(getJoueur()) = 2  
inv 4 : Terrain : :getNbLignes(getTerrain()) = 13  
inv 5 : Terrain : :getNbColonnes(getTerrain()) = 15  
inv 6 : getNbBombes() >= 0

#### 4.2 Scénario 2

**Nom :** scénario2

**Objectif :** Une partie est lancée le joueur pose une bombe, la bombe est sur le terrain.

**Préambule :** Moteur de jeu initialisé avec 500 pas, puis lancé.

M2 = init(500)

**Contenu :** Parcours = bombe

Sc2 = step(M2, Action.Bombe)

**Oracle :**

inv 1 : getStepRestants() = 499  
inv 2 : Joueur : :getLigne(getJoueur()) = 2  
inv 3 : Joueur : :getColumnne(getJoueur()) = 2  
inv 4 : Terrain : :getNbLignes(getTerrain()) = 13  
inv 5 : Terrain : :getNbColonnes(getTerrain()) = 15  
inv 6 : getNbBombes() >= 1  
inv 7 : bombeExiste(500) = true

### 4.3 Scénario 3

**Nom :** scénario3

**Objectif :** Une partie est lancée le joueur pose une bombe qui explose au bout de 10 pas.

**Préambule :** Moteur de jeu initialisé avec 500 pas, puis lancé.

M3 = init(500)

**Contenu :** Parcours = bombe,rien,rien,rien,rien,rien,rien,rien,rien,rien,rien

Sc3 = step(step(step(step(step(step(step(step(step(step(M3, Action.Bombe), Action.Rien), Action.Rien), Action.Rien), Action.Rien), Action.Rien), Action.Rien), Action.Rien), Action.Rien), Action.Rien)

**Oracle :**

inv 1 : getStepRestants() = 489

inv 2 : Joueur : :getLigne(getJoueur()) = 2

inv 3 : Joueur : :getColumnne(getJoueur()) = 2

inv 4 : Terrain : :getNbLignes(getTerrain()) = 13

inv 5 : Terrain : :getNbColonnes(getTerrain()) = 15

inv 6 : getNbBombes() >= 0

inv 7 : bombeExiste(500) = false

## 5 Traces de l'implémentation buggée

Afin de prouver que nos tests sont pertinents et qu'ils couvrent une majorité de problèmes potentiels, nous avons écrit une implémentation du jeu en y incluant des bugs. Ces bugs doivent empêcher les tests de passer.

Obtention des traces : commande `ant runbug`

Un exemple d'exécution est disponible dans le fichier `trace_implem_bug.txt` à la racine du projet.

## 6 Conclusion

Ce projet nous a permis d'apprendre à construire une application découpée en plusieurs composants en se basant sur une série de contrats (voir section 2) et de tests (voir sections 3 et 4) qui s'avèrent longs et fastidieux à mettre en place mais qui garantissent une certaine robustesse du code.