Antoine de Roquemaurel (3.1)

Rédigé avec LaTeX
Version du 13 avril 2014

Simulation d'une campagne de tests

Tests et Maintenance Logiciel

L3 Informatique
Semestre 6

Pour Ileana Ober
Rédigé par Antoine de Roquemaurel

2014

Avant-propos

Ce dossier a été rédigé par Antoine de Roquemaurel dans le cadre de l'enseignement Tests et Maintenance Logiciel délivré en L3 Informatique parcours Ingénierie des Systèmes Informatiques à l'université Toulouse III – Paul Sabatier.

Ce projet à pour but de simuler une campagne de tests sur un logiciel de taille moyenne, logiciel développé par d'autres étudiants au semestre précédent : un jeu en réseau.

Ce rapport à été fournis dans une archive zip, contenant plusieurs fichiers :

Document de test Ce présent document

Répertoire contenant les jeux de tests

Répertoire contenant les scripts de tests

Lecture de ce rapport

Ce rapport, contient plusieurs fonctions typographiques détaillées ci-dessous :

> Ceci est un message afin d'attirer l'attention du lecteur

> R Ceci est une remarque afin d'apporter des précisions sur quelque chose

Ce message contient une note de bas de page [1]

1. Ceci est une note de bas de page

Table des matières

1 Spécifications

Le logiciel est un jeu en réseau, chaque joueur est téléporté dans une arène. Chaque personnage présent dans l'arène se battent, le gagnant est le joueur ayant survécu. Il est également possible d'envoyer des équipements (bonus ou malus) dans l'arène.

1.1 Spécifications générales

1.1.1 Spécifications fonctionnelles

Le jeu doit posséder les fonctionnalités suivantes :
- Téléporter un joueur ou un équipement dans l'arène
- Déplacer automatiquement un joueur
- Ramasser un objet
- Combattre un autre joueur
- Afficher les positions des différents joueurs
- Gagner/perdre

1.1.2 Spécifications non fonctionnelles

Afin de pouvoir installer le jeu, le serveur doit posséder les fonctionnalités suivantes :
- JVM[1] 1.5
- Suffisamment de RAM[2](> 1Gio) alloué à la JVM afin de lancer suffisamment de joueurs simultanément
- Accepter le protocole RMI

Les clients doivent posséder les fonctionnalités ci-dessous :
- JVM 1.5
- Une interface Graphique
- Être connecté en réseau au serveur
- Accepter le protocole RMI

1. Java Virtual Machine
2. Random Access Memory

1.2 Partie de la spécifications détaillées

1.2.1 Le déplacement dans l'arène

Le déplacement d'un joueur peut en prendre plusieurs critères, notamment la *stratégie*, cependant nous partons du principe que le déplacement d'un joueur est indissociée de sa stratégie.

Le déplacement d'un joueur s'effectue donc dans la classe `Console`, dans cette classe, deux méthodes sont concernées par le déplacement : la méthode `run()` et la méthode `seDirigerVers(int)`.

`run()` Cette méthode est appelé à chaque tour, c'est-à-dire à chaque fin du timer, cette méthode appel la fonction de déplacement avec les coordonnées corrects. C'est-à-dire un personnage ou un objet si il y en a un proche, ou alors signale à cette méthode que le personnage devra errer.

`seDirigerVers(int)` Cette méthode déplace le personnage en fonction de son paramètre : soit le personnage se dirige vers un objet ou un autre personnage, soit celui-ci « erre », c'est-à-dire qu'il se déplacera vers un point calculé aléatoirement.

1.2.2 L'interaction entre objets

Les interactions se font dans la classe `Arene`, avec soit une interaction entre deux personnages, soit un ramassage d'objet, ces deux cas sont gérés de façon différentes.

Interaction entre deux personnages

Une interaction entre deux personnages va générer un duel entre ceux-ci. Cette fonctionnalité est gérer dans deux parties du logiciel :

Interface `IArene` La classe `Arene` via l'implémentation de la méthode `interaction(int, int)` qui gère le combat entre deux références d'objets. Cette méthode appel la fonction `DuelBasic` qui va se charger d'exécuter un duel afin de connaître le vainqueur de ce combat.

Interface `IDuel` La classe `DuelBasic` possède la méthode `realiserCombat()` permettant de réaliser le combat entre deux personnages, c'est cette méthode qui permettra de connaître le vainqueur de ce duel.

Ramassage d'un objet

Le ramassage d'un objet se fait dans les classes `Arene` et `Console`, la gestion est différente que celle des combats.

Interface `IArene` La classe `Arene` possède l'implémentation de la méthode `ramasser(int, int)` qui permet un personnage de ramasser un objet. Cette méthode vérifie que le

personnage à le droit de ramasser l'objet, et si c'est le cas, elle fait ensuite appel à la méthode qui va ramasser l'objet proprement dit sur la map.

Interface `IConsole` La classe `Console` possède l'implémentation du ramassage d'un objet, cette méthode affectera les nouvelles caractéristiques au personnage ayant bien ramassé l'objet.

Dans la suite de cette campagne de tests, nous allons nous intéresser aux fonctions d'interactions et de déplacement énumérés dans les spécifications détaillées ci-dessus.

Nous allons donc nous intéresser aux interfaces et aux implémentations associés suivantes : `IConsole`, `IDuel`, `IArene`.

2 Tests

2.1 Tests unitaires

Afin de pouvoir lancer les tests unitaires, plusieurs dépendances doivent être respectées.

Tout d'abord, le serveur doit être lancé, afin que la connexion RMI s'effectue sans problème. D'autres parts, les méthodes de déplacement ou d'interactions dépendent d'autres classes, ainsi les tests unitaires de chacune des méthodes dépendent d'autres méthodes.

> Afin de simplifier notre travail, nous considérerons que les autres modules autres que ceux testés sont corrects.

2.2 Tests d'intégration

Afin de faire un test d'intégration, la méthode la plus simple vérifiant que tout se passe correctement consiste à faire apparaître deux personnages assez proche pour qu'ils se rencontrent : un combat va avoir lieu, leurs caractéristiques doivent être de sorte que nous pouvons savoir à l'avance qui va gagner. Une fois le vainqueur déclaré, un objet apparaîtra proche de lui, il doit le ramasser.

Si notre scénario se déroule comme prévu : les deux personnages se rencontrent, le vainqueur est celui prévu, et le vainqueur ramasse un objet lui prodiguant les nouvelles caractéristiques prévues alors nous pourrons dire que les modules fonctionnent correctement entre eux.

2.3 Tests de validation et test fonctionnel

Il faut créer une map avec un objet et deux personnages, les deux personnages doivent combattre ensemble, et un des personnage doit ramasser un objet.

Afin de faire un test de robustesse, il faudrait appeler la méthode `interaction` ou la méthode `ramasser` avec des valeurs négatives, ou des références n'existant pas, afin de vérifier si le programme gères bien ce cas.