Laurea Magistrale in informatica - Università di Salerno
Corso di *Ingegneria Gestione ed Evoluzione del Software*

Strategic Robots
Unit Test Design

Versione	2.0
Data	14/10/2021
Destinatario	Prof. Andrea De Lucia Dott. Fabiano Pecorelli Dott. Emanuele Iannone Dott. Manuel De Stefano
Presentato da	Antonio Martella

Sommario

1. Introduzione

Lo scopo di questo documento è quello di pianificare il test di unità del sistema Strategic Robots. Verranno riportate le strategie adottate per il testing, gli strumenti utilizzati e le funzionalità testate.

In questa fase saranno effettuati dei test sulle unità che non interagiscono con altre unità del sistema, tenendo conto anche del fatto che i possibili dati di input e output sono stati opportunamente suddivisi in categorie, rappresentanti stati validi o non validi per gli stessi; di conseguenza, durante il test ci si è accertati anche della correttezza delle variabili usate.
Successivamente, sarà possibile procedere al test di integrazione.

2. Dettagli per il testing di unità

2.1 Features da testare

Durante questa fase verranno ricercate le condizioni di fallimento, dove possibile, nei singoli componenti. Le classi che verranno sottoposte al testing di unità sono le seguenti:

- Robot.java
- RobotLavoratore.java
- RobotCombattente.java
- Ostacolo.java
- BarraEnergia.java
- BancoRifornimenti.java

2.2 Features da non testare

Le altre classi rimanenti saranno testate tramite il test di integrazione e di sistema

2.3 Strategia

La strategia usata per il testing è la strategia Black-Box, che si concentra sul comportamento Input/Output ignorando la struttura interna della componente.
Per minimizzare il numero di test case, i possibili input verranno partizionati attraverso il metodo del Category Partition, solo però alcune funzionalità saranno testate con test di unità.
Il framework di supporto utilizzato è JUnit.

2.4 Pass/fail criteria

Nel caso in cui dovessero riscontrarsi errori durante la fase di testing, si procederà con la correzione dei fault intervenendo direttamente sulle porzioni di codice che generano il problema, ed iterando nuovamente con la fase di unit testing verificando che la correzione non abbia impattato altre componenti.

2.5 Criteri di sospensione e ripristino

- Criteri di sospensione

  Comprendono tutti quei casi critici di quando gli errori hanno un impatto dannoso sul progresso dell'attività di testing. Esempi possono essere:

  - Fallimento di funzionalità interne
  - Problemi relativi all'ambiente di sviluppo del testing

- Criteri di ripristino

  La ripresa del sistema avviene solo quando tali errori vengono risolti, ripartendo dal test case che ha causato l'errore.

3. Unit Test Case

3.1 UTC_Robot

Nome della classe da testare	**Robot.java**
Nome della classe test	**UTC_Robot.java**
ID Unit Test Case	**Metodo**
UTC_Robot_01	testGetTipo()
UTC_Robot_02	testGetX()
UTC_Robot_03	testGetY()
UTC_Robot_04	testGetEnergia()
UTC_Robot_05	testGetEnergiaSpostamento()
UTC_Robot_06	testtGetMaxEnergia()
UTC_Robot_07	testGetColore()
UTC_Robot_08	testModificaEnergia()
UTC_Robot_09	TestSpostamentoX()
UTC_Robot_10	TestspostamentoY()
UTC_Robot_11	testModificaX()
UTC_Robot_12	testModificaY()
UTC_Robot_13	testModificaEnergiaSpostamento()
UTC_Robot_14	testModificaColore()

3.2 UTC_RobotLavoratore

Nome della classe da testare	**RobotLavoratore.java**
Nome della classe test	**UTC_RobotLavoratore.java**
ID Unit Test Case	Metodo
UTC_ RobotLavoratore _01	testGetTipo()
UTC_ RobotLavoratore _02	testGetEnergiaCura()
UTC_ RobotLavoratore _03	testGetCapacitàCura()
UTC_ RobotLavoratore _04	testSpostaOgetto()
UTC_ RobotLavoratore _05	testModificaEnergiaCura()
UTC_ RobotLavoratore _06	testModificaCapacitàCura()
UTC_ RobotLavoratore _07	testCuraRobot()

3.3 UTC_RobotCombattente

Nome della classe da testare	**RobotCombattente.java**
Nome della classe test	**UTC_RobotCombattente.java**
ID Unit Test Case	Metodo
UTC_ RobotCombattente _01	testGetTipo()
UTC_ RobotCombattente _02	testGetDannoAttacco()
UTC_ RobotCombattente _03	testGetEnergiaAttacco()
UTC_ RobotCombattente _04	testAttacca()

3.4 UTC_Ostacolo

Nome della classe da testare	**Ostacolo.java**
Nome della classe test	**UTC_Ostacolo.java**
ID Unit Test Case	Metodo
UTC_Ostacolo_01	testGetTipo()
UTC_Ostacolo_02	testGetX()
UTC_Ostacolo_03	testGetY()
UTC_Ostacolo_04	testGetEnergia()
UTC_Ostacolo_05	testModificaX()
UTC_Ostacolo_06	testModificaY()
UTC_Ostacolo_07	testSpostamentoX()
UTC_Ostacolo_08	testSpostamentoY()
UTC_Ostacolo_09	testModificaEnergia()

3.5 UTC_BarraEnergia

Nome della classe da testare	**BarraEnergia.java**
Nome della classe test	**UTC_BarraEnergia.java**
ID Unit Test Case	Metodo
UTC_BarraEnergia_01	testGetEnergia()
UTC_BarraEnergia_02	testGetMaxEnergia()

3.6 UTC_BancoRifornimenti

Nome della classe da testare	**BancoRifornimenti.java**
Nome della classe test	**UTC_BancoRifornimenti.java**
ID Unit Test Case	**Metodo**
UTC_BancoRifornimenti_01	testGetTipo()
UTC_BancoRifornimenti_02	testGetX()
UTC_BancoRifornimenti_03	testGetY()
UTC_BancoRifornimenti_04	testGetRiservaEnergia()
UTC_BancoRifornimenti_05	testModificaX()
UTC_BancoRifornimenti_06	testModificaY()
UTC_BancoRifornimenti_07	testModificaRiservaEnergia()