Emergent Architecture Design document

S&R Context group 2

June 12, 2014

1 Introductie

1.1 Ontwerp doelen

De ontwerpdoelen van dit deel van de S&R context gaan voornamelijk over het maken van meer features om de BW4T omgeving realistischer te maken, maar daarnaast zijn er ook nog wat andere doelen:

1.1.1 Makkelijk uit te breiden Een van de ontwerp doelen van onze groep (en van alle groepen binnen de S&R context) is het makkelijk maken om de omgeving uit te breiden. Voor ons houdt dit in dat het voor latere programmeurs makkelijk moet zijn om aan de hand van wat wij geschreven hebben een uitbreiding te schrijven die van ons werk gebruik maakt en op ons werk voortborduurt. Wij streven er dan ook naar om de implementaties van de features zo herbruikbaar mogelijk te maken, zonder dat er veel oude code aangepast hoeft te worden.

1.1.2 Kwaliteit van de code Een onderdeel van een makkelijk uit te breiden programma is uiteraard een hoge kwaliteit van de code. Als de code goed geschreven en goed gedocumenteerd is, dan zal het voor latere groepen alleen maar makkelijker zijn om toevoegingen of aanpassingen te maken. Om dit ontwerp doel te bereiken maken we dan ook gebruik van een checkstyle plugin die ons helpt bij het verhogen van de code kwaliteit. Deze checkstyle plugin gebruikt een .xml file met bepaalde specificaties die een indicator geven van hoe nette code eruit zou moeten zien. De plugin waarschuwt ons dan ook als we code schrijven die volgens de specificaties in de .xml file niet netjes is. Zo kunnen we dan makkelijk zien wat er fout is en dat oplossen.

2 Software architectuur

Zoals eerder vermeld is de architectuur van ons deel van het systeem (namelijk handicaps en bot store) ontworpen met als idee dat het makkelijk uit te breiden is met nieuwe handicaps door mensen die later aan dezelfde code werken. Hoe

dit voor elkaar gekregen is, zal naast andere structurele onderdelen van de code in dit deel van het verslag behandeld worden.

2.1 Tests

Tests worden geschreven en uitgevoerd met JUnit, een test framework voor Java dat zorgt voor hele makkelijke unit tests. Bij uitvoer van de unit tests geschreven met behulp van de annotaties van dit framework zal er een overzicht komen van de tests, met daarin welke geslaagd of gefaald zijn, en indien ze gefaald zijn wordt ook nog een reden gegeven waarom, zodat er makkelijk opgespoord kan worden waar de fout zit. Sommige klassen die onderworpen worden aan een unit test zullen afhankelijkheden hebben. Door middel van een tweede framework, namelijk Mockito, kunnen we deze afhankelijkheden eenvoudig mocken, waarna we de objecten van de klassen met afhankelijkheden makkelijk aan kunnen maken. Met Mockito en gemockte objecten kunnen we ook het gedrag van de gemockte objecten aanpassen, zoals dat ze bij een bepaalde soort invoer een bepaalde soort uitvoer geven indien een van de methoden van die test een dusdanige uitvoer vereist. Ook kan met Mockito geverifieerd worden dat sommige methoden daadwerkelijk aangeroepen worden, wat erg handig is in het maken van unit tests die gedrag testen.

2.2 Continue integratie

Als server die dient bij continue integratie gebruiken we Jenkins. TODO.