Project Software-Ontwikkeling I: Behaviour-based robotics

Pieter De Baets
Jasper Van der Jeugt
(Groep 31)

12 december 2009

Ontwerpbeslissingen

Factories voor het aanmaken van obstakels

We besloten het Factory-design pattern te gebruiken om obstakels aan te maken. Dit leek ons een elegante keuze, omdat we zo op een eenvoudige manier een `string` kunnen linken aan een `Obstacle`. Voor elk type van `Obstacle` (`Trench`, `ThinWall` en `ThickWall`) moet er dus ook een `ObstacleFactory` bestaan. Omdat dit tot veel herhaling zou leiden qua code, gebruikten we templates, zodat we deze drie klassen konden implementeren in één algemene klasse, namelijk `TObstacleFactory`.

EventProducer

Omdat het aantal lijnen code in `Map.cpp` zeer snel steeg, leek het ons een goed idee deze klasse verder op te spitsen. We splitsten het deel dat op een abstracte manier met events omgaat af in een klasse `EventProducer`. Op deze manier wordt de code van `Map` iets overzichtelijker.

BinarySearchTree

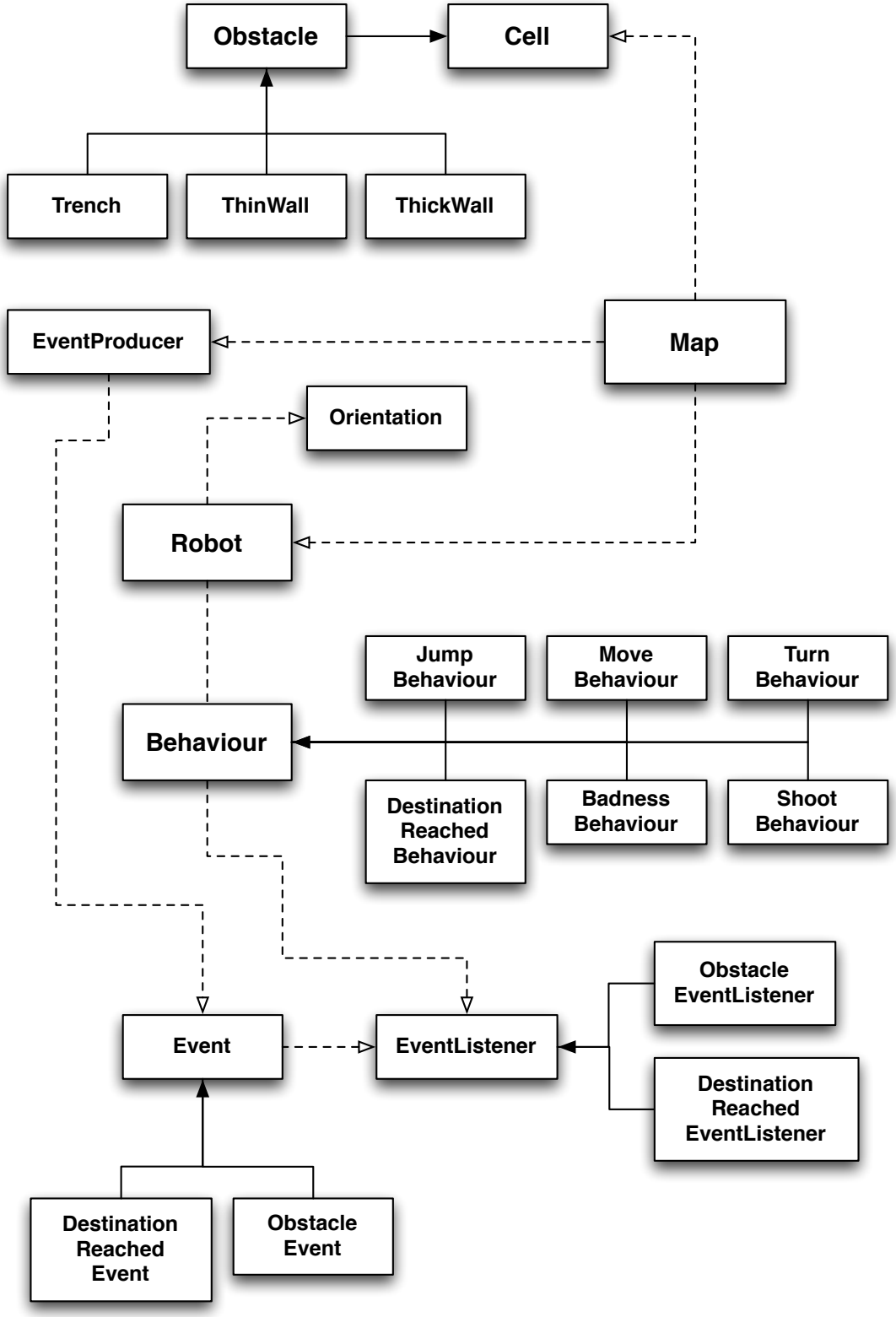
Volgens het principe van *information hiding* beslisten we van `TreeElement` een `private nested` klasse te maken. Ook leek het ons interessant om een balanceringsmechanisme te implementeren. Specifiek kozen we voor de *semi-splay* techniek.

BadnessBehaviour

Om de meer geavanceerde maps op te lossen implementeerden we een extra `Behaviour` klasse. Deze klasse werkt door een bepaalde score (`badness`) te geven aan reeds bezochte plaatsen. Door dit systeem zal de robot de gehele map proberen te verkennen.

Wie deed wat?

Klasse structuur



Figuur 1: Een diagram dat de klassestructuur van het programma toont.