

**Object Georiënteerd Ontwerp**

Zeeslag

Yanice Slegers

Kevin Peelman

Louis Roeben

Dennis Stallaert

2de fase Toegepaste Informatica

21 december 2016

University Colleges Leuven-Limburg  
Campus Proximus  
Geldenaaksebaan 335  
3001 Heverlee  
Tel. +32 (0) 16 375 700

Inhoud

[Algemene opmerking: 3](#_Toc403305641)

[Vereisten 3](#_Toc403305642)

[Patterns 5](#_Toc403305643)

[Speciale topics 6](#_Toc403305644)

[Werkverdeling 7](#_Toc403305645)

[Klassediagramma 8](#_Toc403305646)

# Algemene opmerking

Dit verslag gaat over de opdracht ‘Zeeslag’, dat zal meetellen voor 5 punten van je totaalscore van dit vak. Dit verslag vormt de toelichting bij jouw code.

**Je wordt geacht om je voor het verslag volledig aan deze template te houden. Buiten deze ‘algemene opmerking’ zijn alle hoofdingen verplichte onderdelen van het verslag. Je print dit verslag af (dubbelzijdig) -1 exemplaar per groep en levert dit in bij de start van labo OOO op woensdag 21/12**

**EXTRA: Je bent verplicht om je in code gebruik te maken van het @author commentaarveld!**

**Je maakt een zip file van als je source code (.java bestanden, geen .class bestanden) van alle klassen die van belang zijn voor 000 opdracht. Je voegt tevens de laatse versie van je verslag(word document) toe aan deze zip file. Je mailt deze zip file naar** [**patrick.fox@ucll.be**](mailto:patrick.fox@ucll.be) **uiterlijk voor zondag 25/12 23.59 uur. Naam van de zip file:Zeeslag\_UCLL gevolgd door familienamen van groepsleden gescheiden door underscore (vb Zeeslag\_UCLL\_Dox\_Lenaerts\_Jansen.zip)**

**Tevens maak je van je zzeslag oplossing een jar file die kan runnen onder Java8. Naam van de jar file is dezelfde als naam zip file maar met extentie .jar vb Zeeslag\_UCLL\_Dox\_Lenaerts\_Jansen.jar). Deze jar file mail je samen met je zip file naar** [**patrick.fox@ucll.be**](mailto:patrick.fox@ucll.be)

**Het spreekt voor zichzelf dat de doorgemailde code overeenstemt met de gecommitted code in je repository op projectwerk.khleuven.be**

# Vereisten

Geef voor het gehele project aan welke vereisten (evt. zelf verder uitgewerkt/opgesplitst aan de hand van de opgave) je succesvol hebben geïmplementeerd, en welke topics niet gelukt zijn. Indien je een deel van de voorziene vereisten niet afgewerkt hebt, geef dan aan waarom niet… De reden kan “tijdsgebrek” zijn, het kan een issue zijn “wist niet hoe, het crashte”, of het kan zijn dat je een zeer goede reden had om het niet te implementeren…

Voeg het definitieve gegenereerde klassendiagramma van je code toe, als afzonderlijke afbeelding, als bijlage bij dit rapport.

Als er zaken zijn uit de opdracht die je niet hebt kunnen uitwerken of die je beter zou willen uitwerken, dan som je deze hier tevens op (in laatste rij van tabel)

|  |  |  |
| --- | --- | --- |
| User story | OK? | Issues? (indien niet ok) |
| 1. Teken bord | Ja |  |
| 1. Plaats schepen | Ja |  |
| 1. Controle plaats schepen | Ja |  |
| 1. AI speler plaatst schepen |  |  |
| 1. Start spel | Nee | We hebben geen State patroon gebruikt voor het verschil tussen een nieuw en gestart spel te maken. We gebruiken state bij targets. Meer uitleg kan u vinden bij het overlopen van de verschillende patterns. |
| 1. Aanvallen |  |  |
| 1. Schip geraakt |  |  |
| 1. Aangevallen worden |  |  |
| 1. Toon score |  |  |
| 1. Einde spel |  |  |
| Extra stories? |  |  |

# Patterns

Geef voor elk gezien patroon aan waar je het gebruikt hebt (mogelijkerwijs meer dan eens). Genereer een klassendiagram voor elke situatie waarin je het patroon gebruikt hebt.

Geef extra informatie (voordelen / waarom / …). Als je een patroon niet toegepast hebt, leg je uit waarom niet.

|  |  |  |
| --- | --- | --- |
|  | Toegepast (ja/nee)  In welke stories(nr)  Waarom toegepast(voordeel) | Bijhorend klassendiagram (uit object aid) |
| Observer | We gebruiken dit in user story 9. Wanneer een speler aanvalt roept dit een update functie aan. Deze functie geeft de score door aan de view die deze toont achter de speler zijn naam.  Dit heeft als voordele dat de view automatisch wordt aangepast wanneer er een bepaalde actie heeft plaatsgevonden. (De speler klikt, de AI stuurt een target.) | Nog niet geïmplementeerd. |
| Strategy | We gebruiken Strategy voor onze AI. Deze heeft de keuze om een simpel en advanced vorm aan te nemen. Dit was handig omdat de plaats en aanval strategie gelijkaardige acties hebben maar een andere, soms moeilijkere implementatie moeten hebben. Zo dupliceren we geen code.  Het is nu ook eenvoudiger om nieuwe strategies toe te voegen aan het spel. Deze moeten dan met relatief minder moeite worden geïmplementeerd omdat we programmeren naar een interface. | UML 9 |
| Simple Factory | We gebruiken simple factory onder ander voor schepen en targets (een deel van een schip) aan te maken. | UML 6 |
| Façade |  | UML 5 |
| Singleton | Ja, story 1,2,3 er mag maar één soort van deze klassen worden aangemaakt. We gebruiken deze klassen om bepaalde settings van het spel op te slaan. (Groote, aantal schepen van een bepaald type,…) | UML 2 |
| State | We gebruiken de State strategie in verschillende story’s. Onder andere in user story 1, 2 en 7. Wanneer we een veld aanklikken verandert dit target van state. Wanneer een schip geplaats wordt dit field healty, als een deel van een schip geraakt wordt krijgt dit een damaged state. Als het schip zinkt een sunken state,…  Een van de voordelen hiervan is dat de state at runtime kan worden veranderd en dat we op voorhand kunnen defïnieren naar welke state men moet veranderen wanneer men geraakt wordt. | UML 8 |
| MVC | We hebben dit principe over de stories heen gebruikt. Onze view kent enkel maar de controller en onze klasse’s in de model kunnen enkel maar interactie (waar nodig) hebben met de controller.  Het voordeel hiervan is dat de controller werkt als een soort van facade voor de view. We kunnen hier complexiteit verbergen voor de user die enkel maar acties doet zoals, klikken, een item selecteren,… | kleurenoverzicht |
| Andere… |  |  |

# Speciale topics

Geef voor elke “speciale topic” aan of je het gebruikt hebt of niet, en zo ja, waar. Toon eventueel aan aan met een klassendiagramma.

|  |  |  |
| --- | --- | --- |
|  | Toegepast (ja/nee)  In welke stories(nr)  Waarom toegepast(voordeel) | Bijhorend klassendiagram (uit object aid) of extra uitleg |
| Enum | We gebruiken dit onder andere in story 2,3,4. We hebben via enum de verschillende schepen geïmplementeerd. Zo konden we makkelijk alle type schepen oproepen, vergelijken, … | UML 6 |
| Properties |  |  |
| Reflection |  |  |
| Mockito testing |  |  |
| Andere… |  |  |

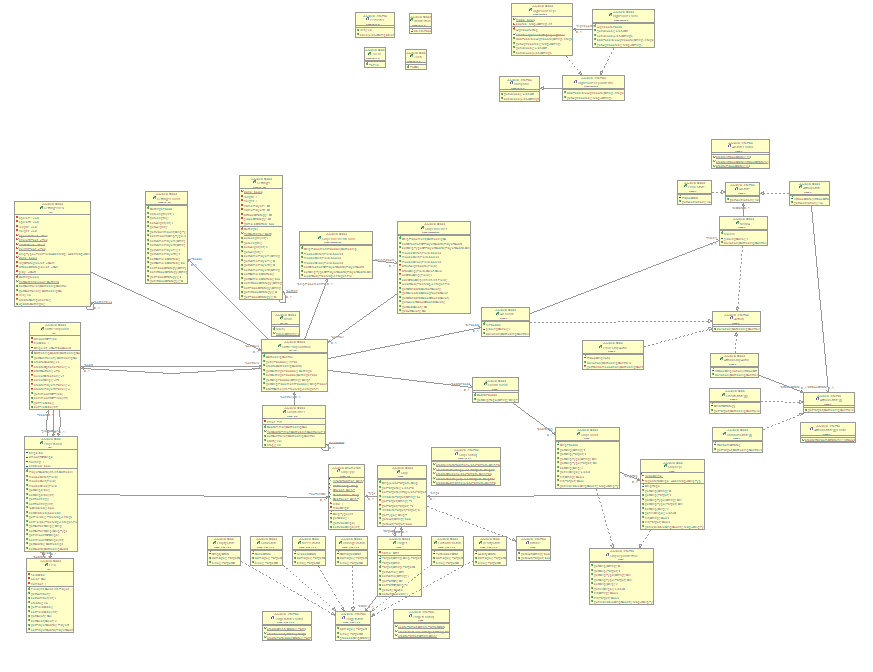
# Werkverdeling

Geef aan in percentages hoeveel je bij benadering gespendeerd hebt aan deze opdracht

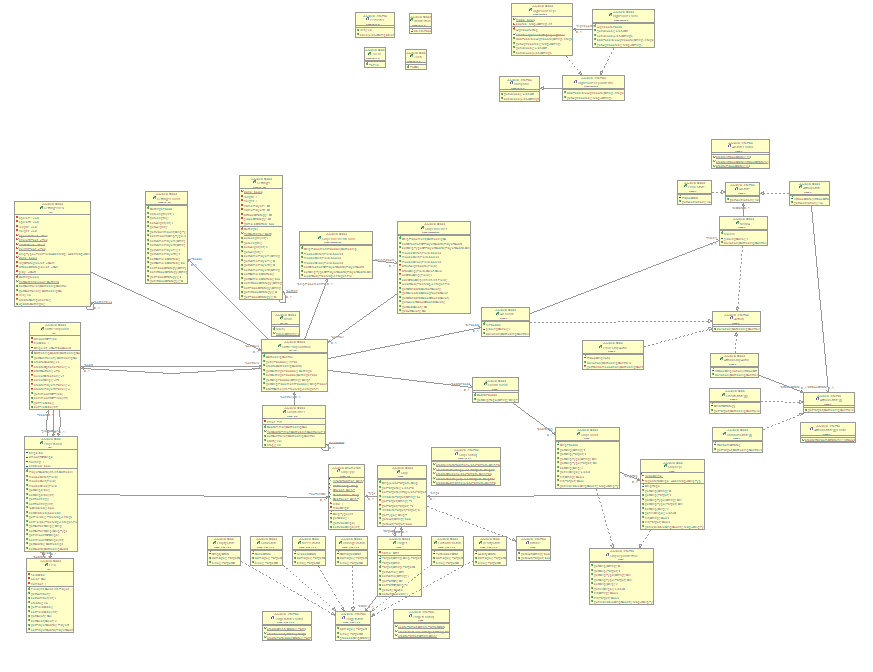
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Yanice Slegers | Kevin Peelman | Louis Roeben | Dennis Stallaert |
| Ontwerp | 30 | 30 | 40 | 0 |
| Klassendiagrammen | 33 | 33 | 33 | 0 |
| Implementatie | 33 | 33 | 33 | 0 |
| Verslag | 50 | 50 | 0 | 0 |

# Klassediagramma

UML, view is blauw, controller is rood, model is de rest.



Overzicht van de verdere opdeling klassendiagrammas.



1

3

2

4

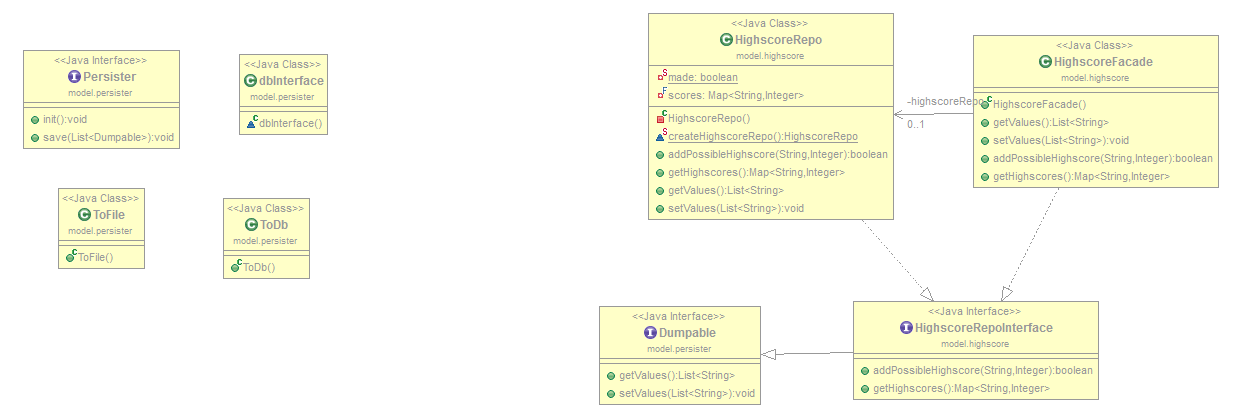
5

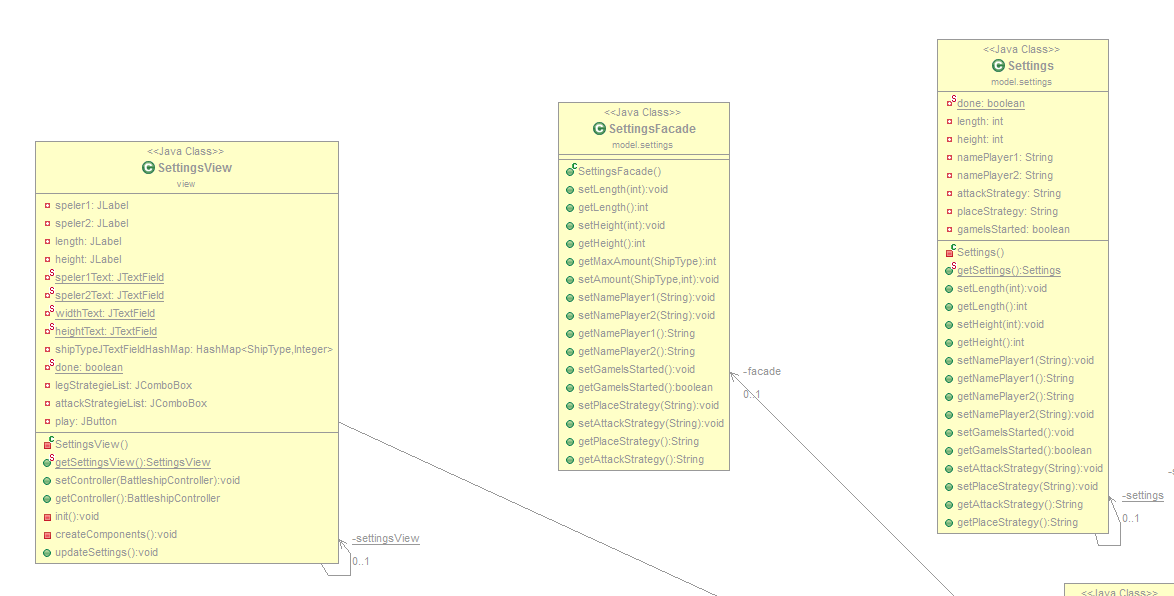
6

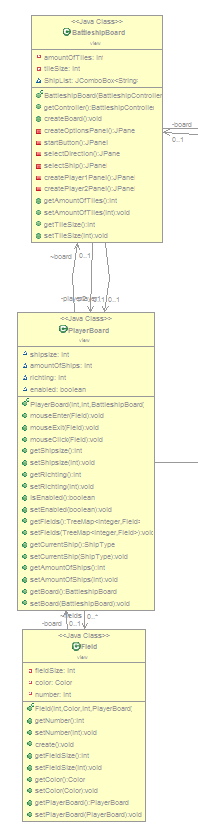
7

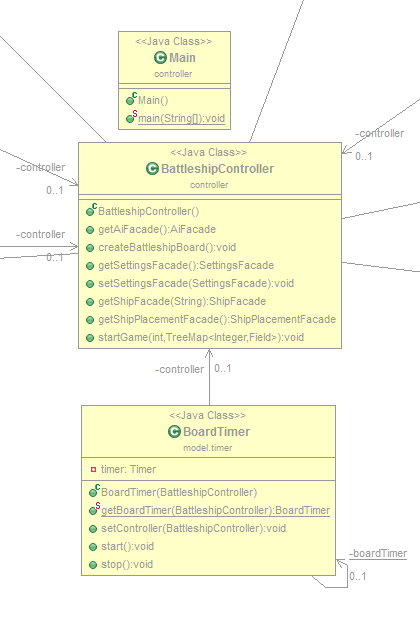
8

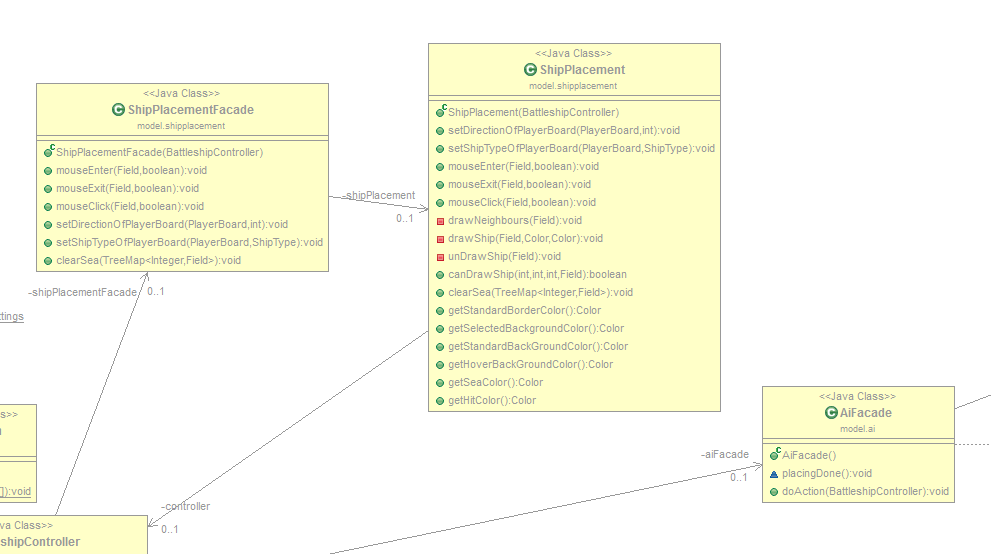
9

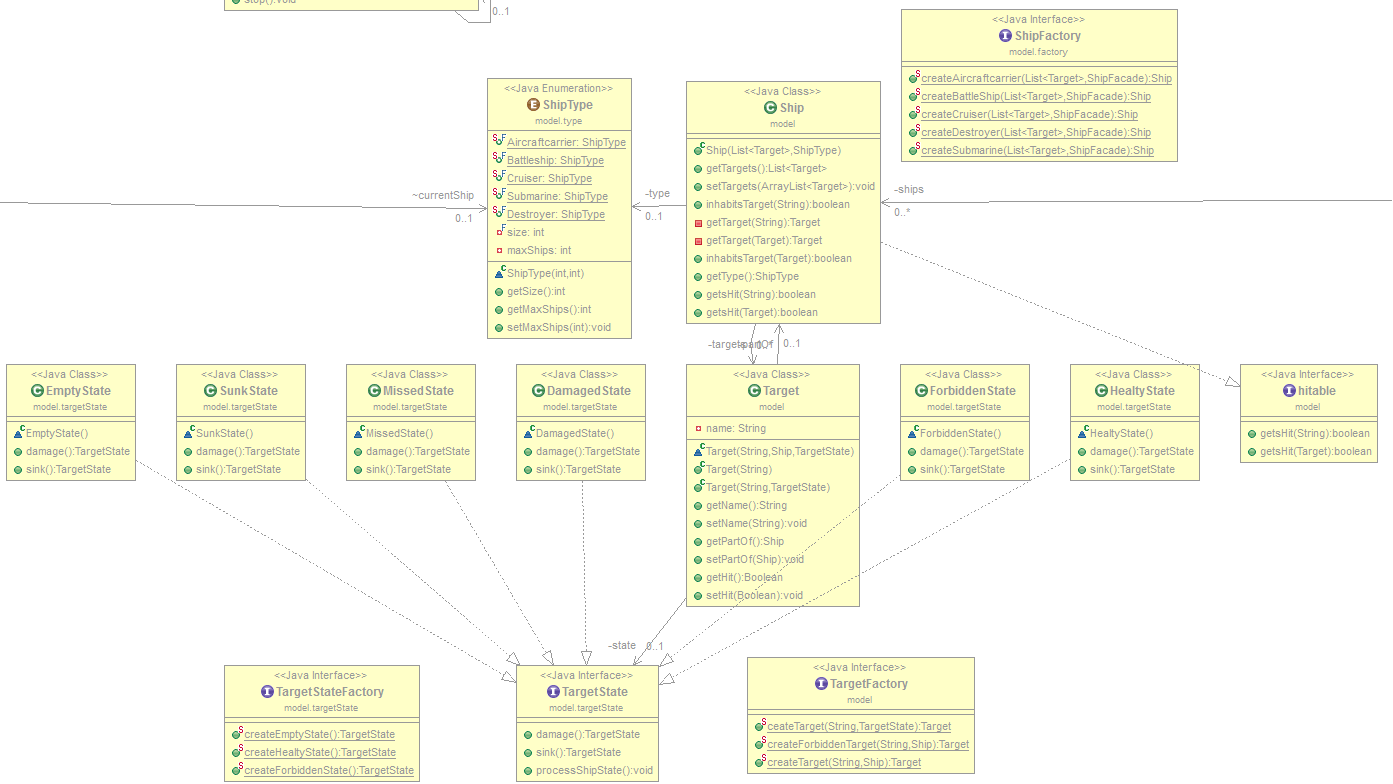
1 Database + highscores

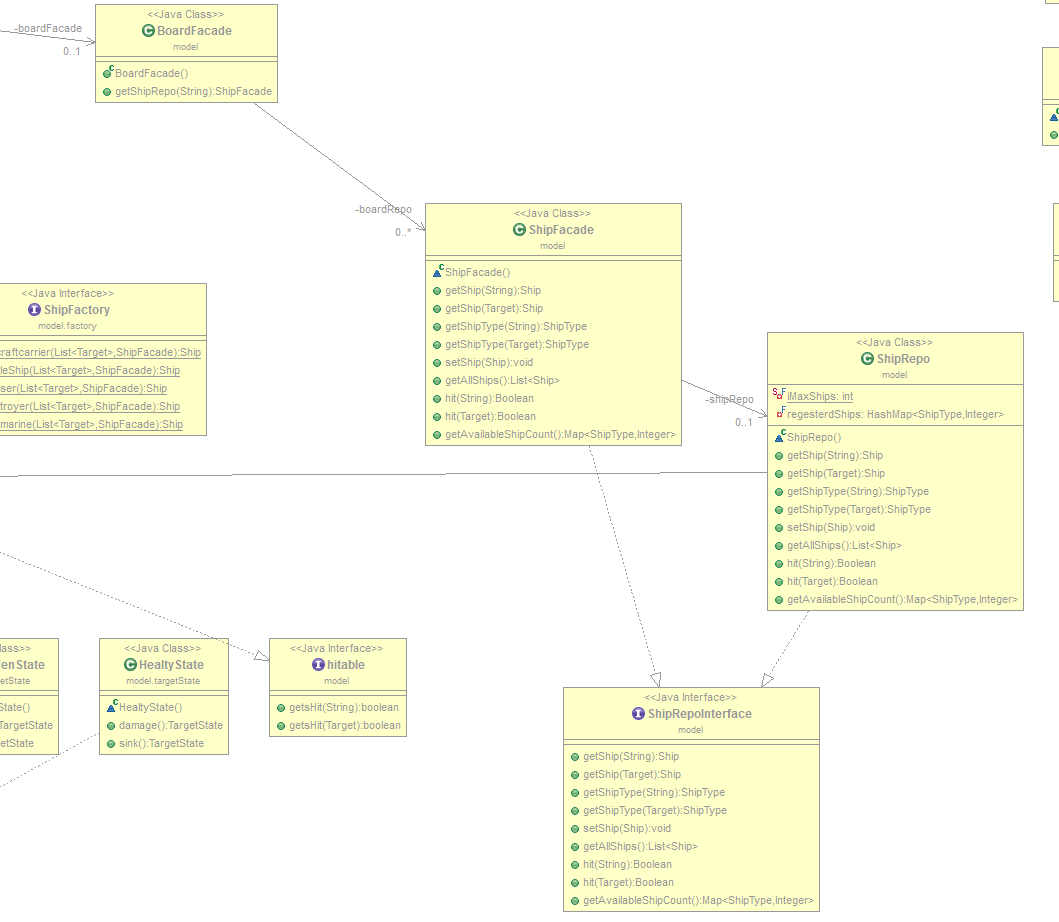
2 Settings

3 View

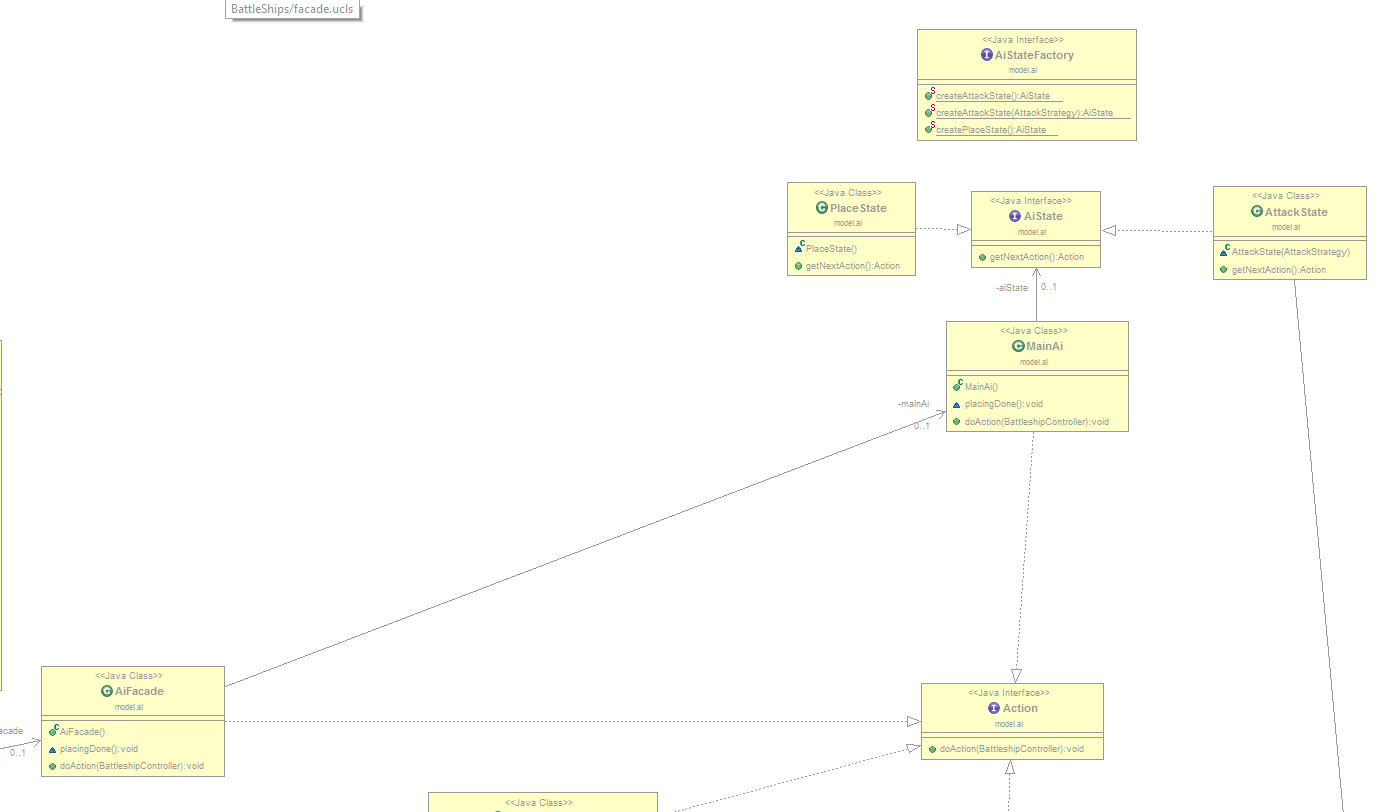
4 Controller

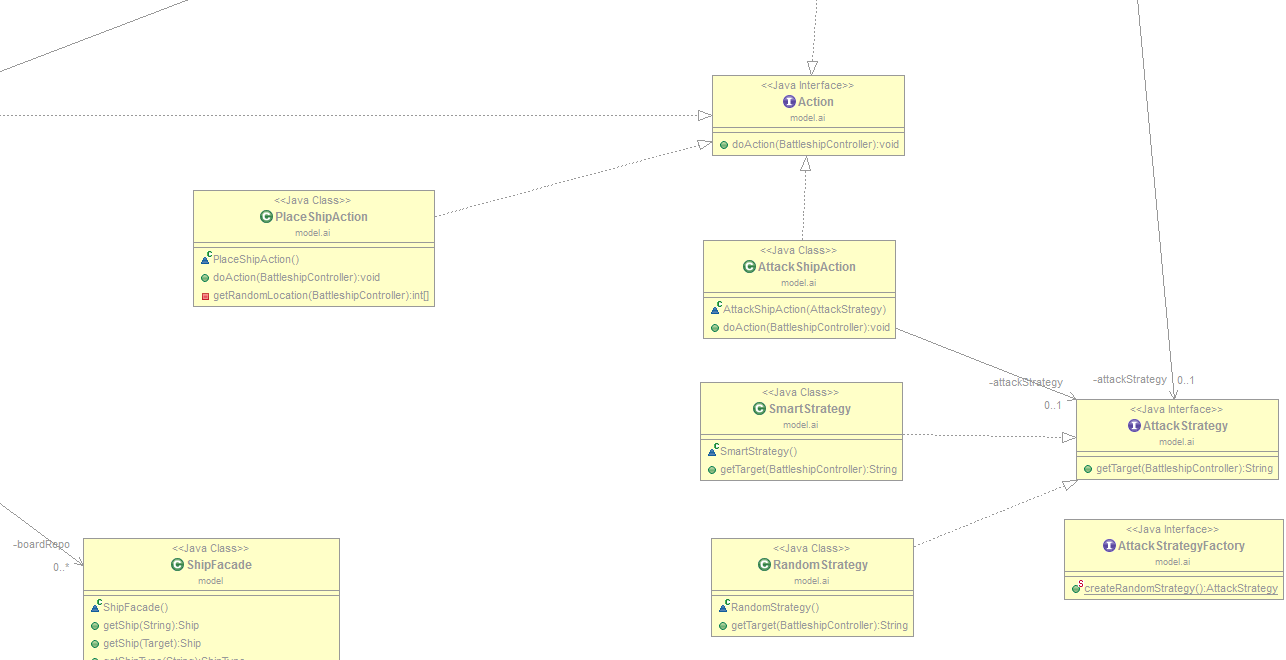
5 ShipPlacement

6 Ships, Targets, TargetState

7 ShipRepo

8 AIState



9 Strategy