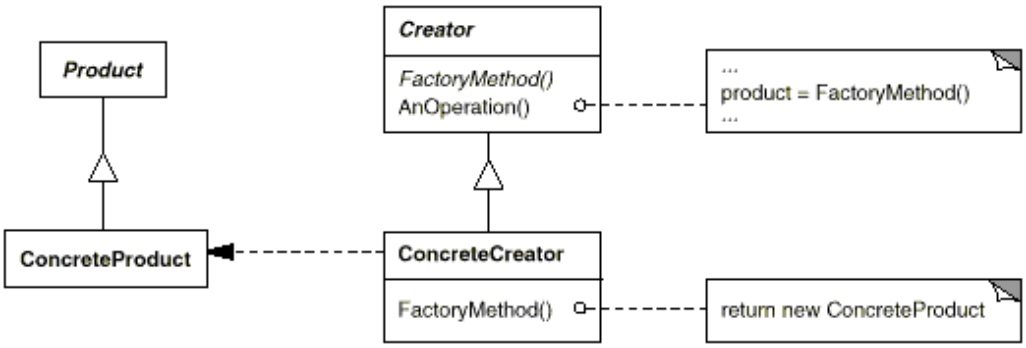
Gebruikte Design Patterns

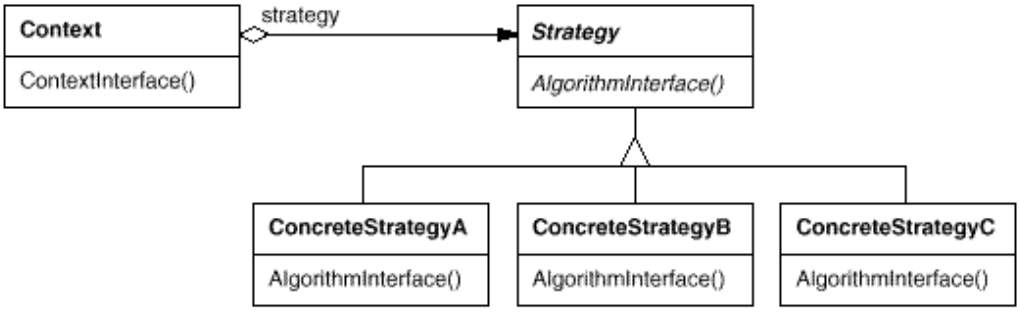
Factory method pattern (Design Patterns p. 121):



Betrokken klassen:
Speler (cfr. Product), Mens en AI (cfr. ConcreteProduct), SpelerFactory (cfr. ConcreteCreator). FactoryMethod wordt gemapt op maakSpeler(type).

Motivatie:
We gebruiken de Speler-klasse doorheen de andere klassen, doch moeten we in staat zijn Mens- en AI-klassen (subklassen van Speler) aan te maken. Het leek ons dus wel handig een SpelerFactory aan te maken. De juiste uitwerking moeten we nog uitdenken. Als we ooit netwerkmogelijkheden zouden implementeren kunnen we zo bvb. Ook makkelijk een nieuw soort speler toevoegen.

Strategy pattern (Design Patterns p. 349):



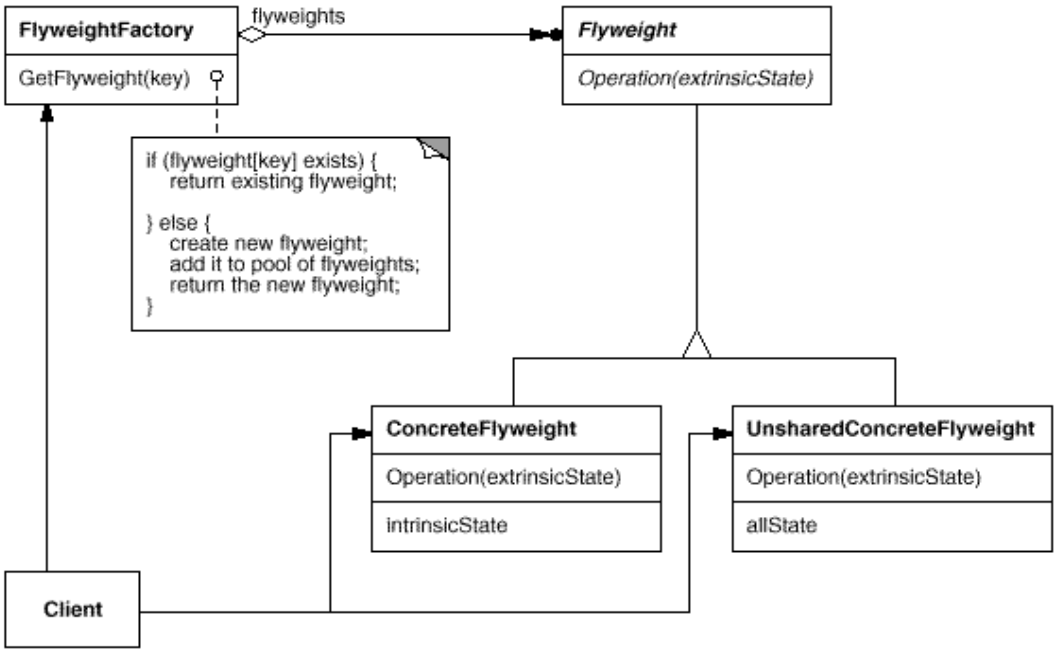
Betrokken klassen:
AI, Easy en Hard

Motivatie:
We kunnen hierdoor een familie van algoritmes definiëren, waardoor we makkelijk kunnen wisselen tussen de verschillende klasse-instanties. Hierdoor zal de A.I. ook makkelijker uitbreidbaar zijn.

Interessante Design Patterns

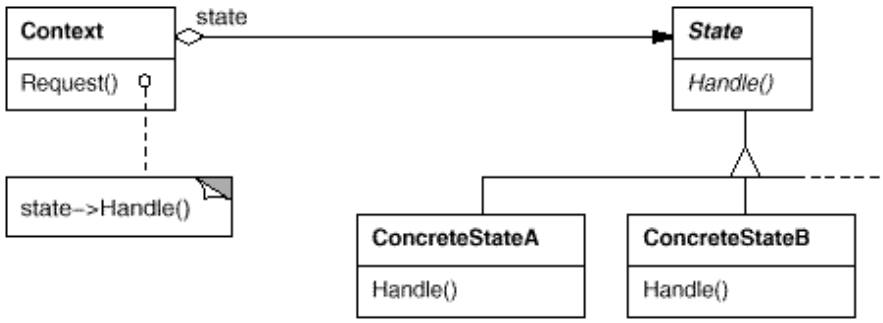
Er moet nog verder onderzocht worden of deze patronen daadwerkelijk een nuttige inbreng aan onze klassestructuur kunnen geven.

Flyweight pattern (Design Patterns p. 218):



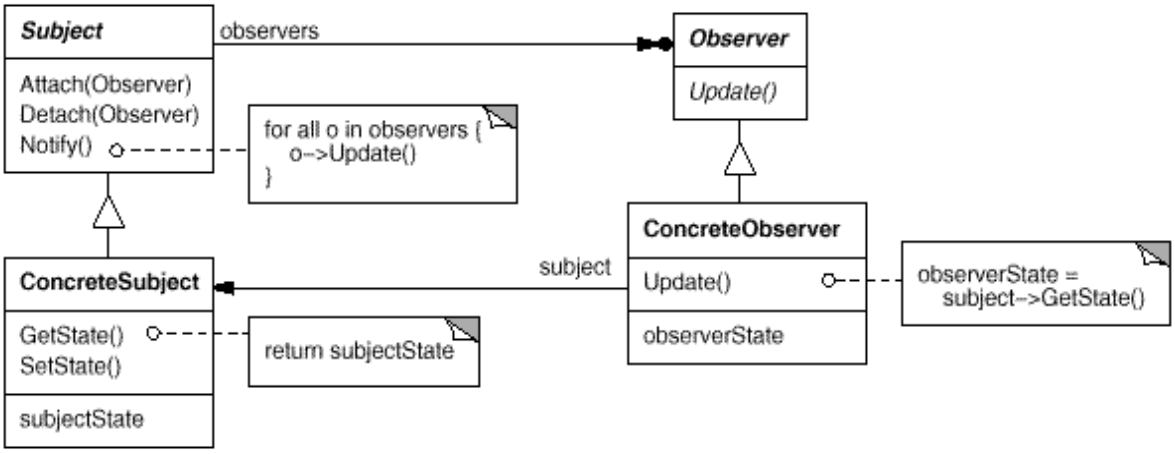
Motivatie:
Dit patroon kunnen we gebruiken voor het voorstellen van de verzameling tegels op de tafel en de verschillende landsdelen op een tegel. Dit patroon zou ons m.a.w. een soort van containerstructuur kunnen geven,

State pattern (Design Patterns p. 338):



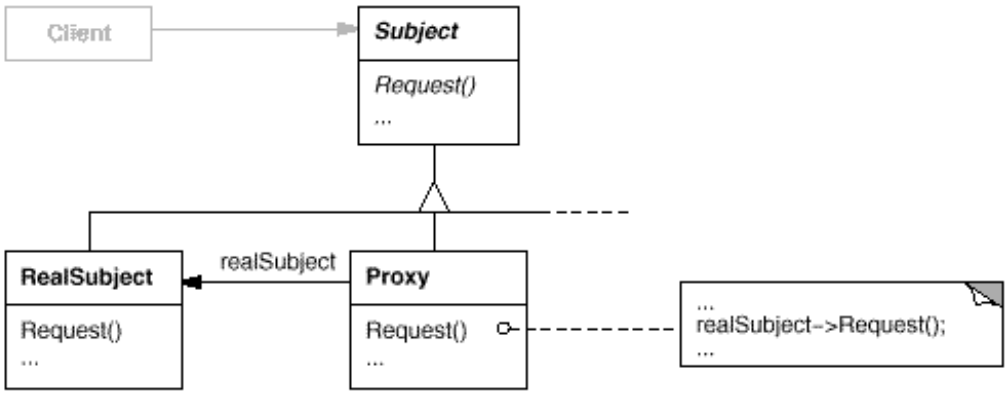
Motivatie:
De puntentelling moet aangepast worden aan de hand van het type van de pion. Dit hangt af van het landsdeel waarop de pion staat, dit hangt dus m.a.w. af van de status (= state) van de pion.

Observer pattern (Design Patterns p. 326):



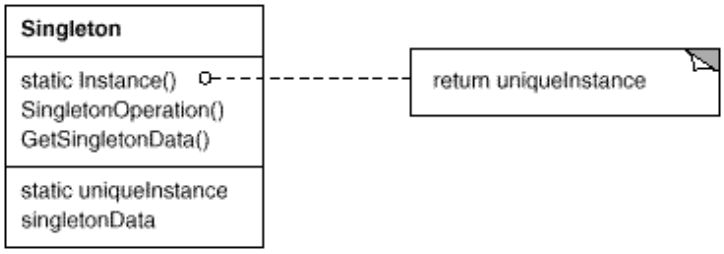
Motivatie:
Om te voorkomen dat elke klasse die als attribuut in Spel zit een Spel-instantie moet meekrijgen om alzo een boodschap terug te kunnen geven aan die Spel-instantie, wordt er van een Observer-structuur gebruik gemaakt.

Proxy pattern (Design Patterns p. 233):



Motivatie:
Dit patroon kunnen we gebruiken voor bijvoorbeeld een file I/O handler te structureren. Hierdoor bekomen we een mechanisme om met files te werken, om alzo een Spel-instantie 1-op-1 op een bestand te mappen.

Singleton pattern (Design Patterns p. 233):



Motivatie:
Dit ontwerppatroon kunnen we gebruiken voor de Help-klasse, omdat er geen nood is aan meerdere Help-instanties.