Rapportage project vossen en konijnen

Een kijk in de wereld van dierenpopulaties





**Door Simulation Solutions**

Project vossen & konijnen

KLAS 1I

SCMI te HanzehogeschoolKarla Derks & Jan-Wiepke Knobbe

30-01-2015, Groningen

Door: D. Bor (TI) 287175

F. Mulder (I) 331021

R. Scholten (TI) 335003

Inhoudsopgave

[Inleiding 4](#_Toc410478600)

[1. Analyse van huidige situatie 5](#_Toc410478601)

[1.1 Beperkingen 5](#_Toc410478602)

[1.2 Voorgestelde oplossingen 5](#_Toc410478603)

[1.3 Beschikbare gegevens van huidige situatie 5](#_Toc410478604)

[1.4 Varkenscyclus 6](#_Toc410478605)

[2. Uitbreiding 1 7](#_Toc410478606)

[2.1 Veranderingen 7](#_Toc410478607)

[2.1.1 Lynx 7](#_Toc410478608)

[2.1.2 Hunter 7](#_Toc410478609)

[2.1.3 Interface Actor 7](#_Toc410478610)

[2.1.4 Animal 7](#_Toc410478611)

[2.1.5 Views 7](#_Toc410478612)

[2.2 Klassendiagram oude situatie 8](#_Toc410478613)

[2.3 Sequencediagram 8](#_Toc410478614)

[2.4 Klassendiagram nieuwe sitatie 8](#_Toc410478615)

[3. Het gebruik van MVC in V&K 9](#_Toc410478616)

[3.1 Overeenkomsten en verschillen mvc-projecten 9](#_Toc410478617)

[3.1.1 Overeenkomsten en verschillen tussen MVCDynamicModelThread en -ThreadGeneralized 9](#_Toc410478618)

[3.1.2 Overeenkomsten en verschillen tussen MVCDynamicModelThreadGeneralized en Life 9](#_Toc410478619)

[3.2 MVC Klassendiagrammen 9](#_Toc410478620)

[3.3 MVC Sequencediagram 9](#_Toc410478621)

[4. Uitbreiding 2 10](#_Toc410478622)

[4.1 Gras 10](#_Toc410478623)

[4.2 Instellingen 10](#_Toc410478624)

[4.3 Ziekte 10](#_Toc410478625)

[4.4 Stenen 11](#_Toc410478626)

[4.5 Geslacht (extra uitbreiding) 11](#_Toc410478627)

[5. Onopgeloste fouten in de applicatie 12](#_Toc410478628)

[5.1 Threading 12](#_Toc410478629)

[5.2 Geslacht 12](#_Toc410478630)

[6. Slot 13](#_Toc410478631)

[Literatuurlijst 14](#_Toc410478632)

[Bijlagen 15](#_Toc410478633)

[Bijlage A (Klassendiagram oude situatie) 15](#_Toc410478634)

[Bijlage B (Sequencediagram) 16](#_Toc410478635)

[Bijlage C (Klassendiagram nieuwe sitatie) 17](#_Toc410478636)

[Bijlage D (MVC Klassendiagrammen) 18](#_Toc410478637)

[Bijlage E (MVC Sequencediagram) 19](#_Toc410478638)

# Inleiding

Dit rapport is geschreven om de ontwikkeling van de applicatie in beeld te brengen. Hierin worden een aantal aspecten behandeld, onder andere waarom de applicatie is gebouwd, voor wie de applicatie is gebouwd en welke veranderingen er zijn toegepast.

Vanuit Staatsbosbeheer is er een opdracht gekomen om een applicatie te ontwikkelen die het verloop van een aantal dierenpopulaties kan bijhouden. De gedachte hierachter is dat Staatsbosbeheer op deze manier gemakkelijk en tijdig kan bepalen of er van een bepaalde populatie dieren moeten worden bijgezet, of juist moeten worden weggehaald.

Deze applicatie beperkt zich niet alleen voor gebruik bij Staatsbosbeheer. Iedereen die interesse heeft in het verloop van dierenpopulaties kan hier gebruik van maken. Deze applicatie kan ook gebruikt worden om andere populaties mee te meten.

De applicatie zoals gegeven door de opdrachtgever bevat een aantal tekortkomingen. De oplossingen die hiervoor gemaakt zijn worden ook gemotiveerd in dit rapport. Deze worden eerst algemeen behandeld en vervolgens worden de uitbreidingen per week besproken.

In hoofdstuk 1 staat beschreven wat de staat is van de applicatie zoals opgeleverd door de opdrachtgever. Hierin wordt behandeld wat de tekortkomingen zijn, en hoe dit eventueel zou kunnen worden opgelost.

Hoofdstuk 2 gaat in op de verbeteringen die in de tweede week van ontwikkeling zijn aangebracht. De uitbreidingen worden afzonderlijk behandeld. Om de veranderingen beter duidelijk te maken wordt er gebruik gemaakt van klassendiagrammen en een sequencediagram.

**In hoofdstuk 3 wordt het gebruik van MVC behandeld. Meer info gewenst**

Hoofdstuk 4 geeft een overzicht van de uitbreidingen weer die in de laatste week van de ontwikkeling zijn toegevoegd. Van iedere uitbreiding wordt kort uitgelegd wat het voor invloed het heeft op de simulatie.

In hoofdstuk 5 worden de tekortkomingen die de applicatie nog bevat, nadat de ontwikkeling is afgerond, kort besproken.

# 1. Analyse van huidige situatie

In dit hoofdstuk wordt gekeken naar de huidige situatie. Al snel valt te zien dat de huidige situatie een aantal beperkingen heeft. Aan de hand van deze beperkingen is het mogelijk om de applicatie verder uit te breiden. De voorbeelden die genoemd gaan worden, worden niet per definitie geïmplementeerd.

## 1.1 Beperkingen

Bij de huidige situatie zijn er een aantal gebreken. Ten eerste is de functionaliteit van de simulatie zeer beperkt. Er zijn geen instellingen mogelijk waardoor het niet mogelijk is om zelf voorwaarden te bepalen. De simulatie is eentonig en biedt geen inzicht in wat er zich precies afspeelt. Iedere run van de simulatie geeft hetzelfde resultaat (zowel in aantallen als de positie van de dieren). Dit is niet realistisch voor een simulatie wat zich afspeelt in de natuur.

Daarnaast is de code ook niet echt optimaal. Zo is het in de huidige situatie moeilijk om extra’s toe te voegen. De code is nu zo opgebouwd dat alle klassen van elkaar afhankelijk zijn. Dit resulteert in het feit dat een kleine verbetering al heel veel moeite kost om te implementeren.

Ook is de GUI aan de matige kant. Zo valt op dit moment de simulatie niet te stoppen of te resetten. Hierdoor vertoont de applicatie soms gedrag dat niet wenselijk is.

## 1.2 Voorgestelde oplossingen

De in paragraaf 2.1 gestelde beperkingen kunnen met simpele theoretische oplossingen worden aangepakt. Het is mogelijk om te zorgen dat er een aantal eigenschappen instelbaar worden. Een aantal voorbeelden hiervan zijn:

* De maximale leeftijd van dieren veranderen
* Een minimale leeftijd waarop een dier zich kan voortplanten veranderen
* De dierenpopulaties aanpassen

Om meer inzicht te krijgen in de data is het mogelijk gebruik te maken van grafieken en diagrammen. Zo is het makkelijker om de gegevens te gebruiken voor een onderzoek, omdat eventuele trends of afwijkingen dan beter zijn op te sporen. Daarnaast is het ook mogelijk om gebruik te maken van statistieken om meer inzicht te krijgen in de data.

Doordat de code een hoge afhankelijkheid heeft moet de code gerefactord worden. Door de code te refactorren wordt het makkelijker om uitbreidingen toe te gaan voegen. Daarnaast zou de code ook veel beter leesbaar moeten worden.

Voor de GUI is er een hele simpele oplossing. Door knoppen toe te voegen en hier acties aan te verbinden krijgt de eindgebruiker al veel meer invloed op hoe de applicatie werkt.

## 1.3 Beschikbare gegevens van huidige situatie

Omdat het in de huidige situatie nog niet mogelijk is om een geautomatiseerde analyse uit te voeren, wordt dit nu handmatig gedaan. De volgende grafiek geeft een beeld over de verhoudingen na 2000 stappen.

Figuur 1. Verticale as zijn de aantallen, en de horizontale as zijn de stappen

De dierenpopulaties zijn over het algemeen stabiel te noemen. Bij de konijnenpopulaties is het duidelijk te zien in figuur 1, dat deze steeds op en neer gaan en eigenlijk nooit onder de 2000 uitkomen. Bij de vossen ligt dit iets anders. Doordat de vossen sowieso al met een veel kleinere populatie te maken hebben, zijn de gegevens per 100 stappen niet bepaald representatief. In de bovenstaande grafiek is er namelijk geen duidelijk verband te zien tussen de populatie van vossen en konijnen.

Deze is er echter wel. In paragraaf 1.4 wordt dit uitgelegd aan de hand van een voorbeeld en de reden waarom dit zo is.

## 1.4 Varkenscyclus

Een vraag over deze gegevens zou kunnen zijn: “Is er hier sprake van een varkenscyclus?” Om deze vraag goed te beantwoorden is het eerst van belang om uit te leggen wat een varkenscyclus is.

Een varkenscyclus is een verschijnsel dat voorkomt in de economie. Dit verschijnsel geeft aan dat een overschot van een bepaald product afgewisseld wordt met een tekort, en vice versa. Op dit moment speelt er een heel goed praktijkvoorbeeld. De olieprijs wordt steeds lager. Dit is te danken aan het feit dat de olieprijs enkele maanden geleden vrij hoog was. Door de hoge olieprijs wordt er meer olie opgeboord in de hoop een graantje mee te pikken van de hoge prijzen. Na verloop van tijd komt er zoveel olie uit de grond dat er een overschot ontstaat waardoor deze prijzen weer dalen. Vervolgens gaat men minder boren en ontstaat er een tekort, waardoor de prijzen weer omhoog gaan. (Sijs, 2010)

Het hierboven beschreven verschijnsel is ook van toepassing op de simulatie. Zodra het aantal vossen hoger word, is er een daling te zien in het aantal konijnen. Als het aantal konijnen hoog is, zijn er minder vossen.

# 2. Uitbreiding 1

In dit hoofdstuk worden de veranderingen besproken die in week 2 van het project zijn gemaakt. Er valt in de klassendiagrammen te zien hoe de oude situatie is, en hoe de nieuwe situatie is geworden.

## 2.1 Veranderingen

Tijdens de 2de week van de ontwikkeling zijn er een aantal verbetering toegevoegd. Deze verbetering worden hier afzonderlijk besproken.

package vossen\_en\_konijnen**;**

**import** java**.**util**.**List**;**

/\*\*

\* De interface die uitgebreid kan worden door elke klasse

\* die deel wil nemen aan de simulatie

\*/

public interface Actor

**{**

/\*\*

\* Voer het gebruikelijke gedrag van de deelnemer uit.

\* @param newActors Een lijst waarin zojuist gemaakte

\* deelnemers worden opgelsagen.

\*/

void act**(**List**<**Actor**>** newActors**);**

/\*\*

\* Is de deelnemer nog steeds actief?

\* @return true als de deelnemer nog actief, anders false.

\*/

boolean isActive**();**

**}**

### 2.1.1 Lynx

Er is een nieuwe diersoort toegevoegd, namelijk de lynx. De lynx is een diersoort dat jaagt op vossen en konijnen. De eigenschappen van een lynx zijn strenger dan van de vos, dit is gedaan omdat een lynx makkelijker aan voedsel kan komen. In de loop der worden er nog uitbreidingen toegevoegd waarmee het mogelijk wordt om deze eigenschappen zelf in te stellen.

### 2.1.2 Hunter

Er is ook een niet-dier toegevoegd, de jager. Een jager is vrij simpel van opzet. Hij jaagt op alle dieren en gaat niet dood. Er is ook geen limiet voordat een jager een volgend dier doodschiet.

Figuur 2. De code van de nieuwe interface Actor

### 2.1.3 Interface Actor

Om alle groepen op het veld goed te kunnen gebruiken is er een interface Actor aangemaakt. Deze interface bevat alle dieren, objecten en personen die aanwezig zijn in de simulatie. Door het aanmaken van deze interface is ervoor gezorgd dat alles nu in een lijst staat. Het was ook mogelijk om dit per groep te doen, maar dit zou de code nodeloos ingewikkeld maken. Hierdoor wordt het een stuk moeilijker om andere functionaliteit toe te voegen. De code van de interface Actor staat in figuur 2.

### 2.1.4 Animal

In de abstracte klasse Animal zijn wijzigen aangebracht. Een aantal methodes die bij ieder dier opnieuw gedeclareerd stonden zijn nu verplaatst naar de klasse Animal. Hierdoor is de code beter leesbaar en een stuk onafhankelijker geworden.

### 2.1.5 Views

Om de statistieken van de simulatie overzichtelijk weer te geven, zijn er verschillende diagrammen toegevoegd. De gebruikte diagrammen zijn:

* Cirkeldiagram
* Staafdiagram
* Lijndiagram

De cirkeldiagram en staafdiagram geven op twee verschillende manieren de verhouding weer tussen de actoren. Dit maakt het inzichtelijk wat er op het veld gebeurt en hoe het veld er op dat moment uitziet.

De lijndiagram geeft een korte geschiedenis van de aantallen van de verschillende actoren weer. Daardoor word er een duidelijk beeld gecreëerd van de ontwikkelingen die plaats vinden op het veld.

## 2.2 Klassendiagram oude situatie

Zie bijlage A.

## 2.3 Sequencediagram

Zie bijlage B.

## 2.4 Klassendiagram nieuwe sitatie

Zie bijlage C.

# 3. Het gebruik van MVC in V&K

## 3.1 Overeenkomsten en verschillen mvc-projecten

### 3.1.1 Overeenkomsten en verschillen tussen MVCDynamicModelThread en -ThreadGeneralized

De projecten MVCDynamicModelThread en -ThreadGeneralized zijn voor het grootste gedeelte aan elkaar gelijk. Beide gebruiken de klassen Controller, CountView, Model, MVCDynamicModelThread (en Generalized), MVCRunner en PieView. De code van deze klassen is dan ook nagenoeg identiek aan elkaar.

Eén verschil is dat ThreadGeneralized abstracte klassen heeft waar zoveel mogelijk van de MVC-aspecten in zitten. Dit zijn dus de klassen AbstractView, -Controller en -Model. Het voordeel hiervan is dat het een stuk gemakkelijker wordt om meer klassen toe te voegen. Er kan bijvoorbeeld nog een View-klasse toegevoegd worden die van AbstractView erft.

Een ander pluspunt is dat de code duidelijker te volgen is en beter is georganiseerd. Het project ModelThread heeft geen aparte abstracte klassen voor MVC. De MVC-aspecten van bijvoorbeeld de ontbrekende AbstractController zitten daar in de klasse Controller. Het kost in zo'n geval meer tijd om wijzigingen aan te brengen in de code.

Een ander verschil is dat de klassen in verschillende packages zijn geordend. De klassen AbstractModel en Model (overerving) zitten in de package nl.hanze.t12.mvc.controller. De folderstructuur is dus nl/hanze/t12/mvc/controller.

### 3.1.2 Overeenkomsten en verschillen tussen MVCDynamicModelThreadGeneralized en Life

Het Project Life volgt dezelfde structuur voor MVC als ThreadGeneralized. Life heeft dus aparte abstracte klassen voor Model, View, en Controller die ook zijn geordend in packages. Een andere overeenkomst is dat ze beide een losstaande main-package hebben, waar de GUI gevormd wordt van de klassen uit de view-package. Ook is er een package voor de runner, waar alleen de main methode zich bevindt.

Een verschil tussen de twee projecten is dat Life in totaal 3 controllers heeft, inclusief een abstracte klasse. ThreadGeneralized heeft er 2; AbstractController en Controller. Life gebruikt het principe van het uitbreiden van de abstracte MVC-klassen dus al meer dan ThreadGeneralized.

## 3.2 MVC Klassendiagrammen

Zie bijlage D.

## 3.3 MVC Sequencediagram

Zie bijlage E.

# 4. Uitbreiding 2

In dit hoofdstuk worden de wijzigingen doorgenomen die in de laatste week van de ontwikkeling van het project zijn toegevoegd.

## 4.1 Gras

Een van de nieuwe toevoegingen voor deze week is de klasse Grass. De implementatie van Grass dient als voedsel voor konijnen. Gras ‘plant’ zich voort net zoals dieren zich voortplanten. Dit is gedaan om alle andere populaties in stand te houden. Een veld waar gras in voorkomt kan namelijk geen dieren bevatten. Als gras dus niet zou doodgaan is er geen plek meer voor de dieren, wat de simulatie niet ten goede komt.

## 4.2 Instellingen

Een andere toevoeging is de optie voor gebruikers om zelf instellingen te veranderen. Dit wordt gedaan door middel van sliders, die in een losstaand paneel te zien zijn. De instelling variëren van de maximale leeftijd dat een dier kan worden tot het aantal jongen of welpen een dier kan krijgen. De minimale leeftijd waarop een konijn zich kan voortplanten is bijvoorbeeld in te stellen tussen de 0 en ???. Dit is real-time te zien terwijl de simulatie loopt, dus is het niet nodig om eerst op ‘Ok’ te klikken. De instellingen zijn per dier gesorteerd in een eigen tabblad.

## 4.3 Ziekte

Voor deze week is er ook een ziekte toegevoegd dat konijnen infecteert. In het begin van de simulatie wordt bepaald of een konijn vatbaar is voor de ziekte. De kans dat een konijn vatbaar is voor de ziekte is ongeveer 90%.

/\*\*

\* Look for rabbits adjacent to the current location.

\* Only the first live rabbit is eaten.

\* @return Where food was found, or null if it wasn't.

\*/

private Location findFood**()**

**{**

Field field **=** getField**();**

List**<**Location**>** adjacent **=** field**.**adjacentLocations**(**getLocation**());**

Iterator**<**Location**>** it **=** adjacent**.**iterator**();**

Location goTo **=** **null;**

**while(**it**.**hasNext**())** **{**

Location where **=** it**.**next**();**

Object animal **=** field**.**getObjectAt**(**where**);**

**if(**animal **instanceof** Grass**)** **{**

Grass grass **=** **(**Grass**)** animal**;**

**if(**grass**.**isActive**()** **&&** goTo **==** **null)** **{**

grass**.**setDead**();**

setFoodLevel**(**GRASS\_FOOD\_VALUE**);**

// Remove the dead rabbit from the field.

goTo **=** where**;**

**}**

**}**

**else** **if(**animal **instanceof** Rabbit**)** **{**

Rabbit r **=** **(**Rabbit**)** animal**;**

**if(**getZiekte**()** **&&** r**.**getZiekteGen**())** **{**

r**.**setZiekte**(true);**

**}**

**}**

**}**

**return** goTo**;**

**}**

Zodra de ziekte wordt geactiveerd, worden 10% van de konijnen die vatbaar zijn voor de ziekte ziek. Deze ziekte verspreid zich vervolgens over de andere konijnen die vatbaar zijn. Als een konijn in de buurt komt van een andere ziek konijn wordt deze ziek. In figuur 3 is de code te zien waarin een konijn besmet wordt.

Figuur 3. De code waar een konijn een ander konijn besmet

Als een konijn ziek is, veranderd de kleur van het konijnen van geel naar oranje. Zo is het makkelijk te zien welke konijnen met de ziekte besmet zijn. Een ziek konijn kan nog vijf stappen lopen voordat het dood gaat.

## 4.4 Stenen

Dit is de meest simpele uitbreiding van de simulatie. Een steen doet niks en doet dienst als een obstakel voor de dieren en het gras. Een steen heeft alleen maar een locatie en een indicatie of de steen nog in ‘leven’ is.

## 4.5 Geslacht (extra uitbreiding)

De toevoeging van geslacht houdt in dat de dieren (in de huidige simulatie vossen, konijnen en lynxen) een geslacht krijgen toegewezen en dat dieren nu een paringsmaat van het andere geslacht moeten zoeken om zich voort te kunnen planten.

# 5. Onopgeloste fouten in de applicatie

## 5.1 Threading

Tijdens het programmeren zijn er bepaalde functionaliteiten van de applicatie kapot gegaan. Dit had tot gevolg dat er gebruik moest worden gemaakt van Threads om alles goed te laten werken.

Deze Threads zijn niet goed geïmplementeerd en kunnen af en toe fouten geven. Door deze fouten kan de applicatie vast lopen en moet deze opnieuw opgestart worden om deze verder te kunnen gebruiken. De oorzaak van de fout kon helaas niet worden achterhaald.

## 5.2 Geslacht

Bij het programmeren van het geslacht moest er meer gebeuren dan in eerste instantie was verwacht. Dit heeft er toe geleidt dat het geslacht dus nog niet volledig is geïmplementeerd in de huidige versie van het applicatie.

In de huidige versie gaan dieren nog niet actief opzoek naar een paringsmaat, met als gevolg dat als er van een diersoort weinig zijn en deze verspreidt zijn over het veld, er geen paartjes gemaakt kunnen worden en het diersoort dus uitsterft. Dit zou in een latere versie van de applicatie nog geïmplementeerd moeten worden.

# 6. Slot

Bij de ontwikkeling van deze applicatie is de hoofddoelstelling een applicatie op te leveren, waarbij de ontwikkeling van dierenpopulaties kunnen worden bijgehouden. De applicatie die geleverd was door de opdrachtgever voldeed hier niet aan. Door de uitbreidingen die zijn geïmplementeerd is de applicatie nu wel geschikt voor deze taak.

Door de grote hoeveelheid aan instellingen en opties is het nu mogelijk om situaties na te bootsen voor specifieke dierenpopulaties. Door de gegevens te analyseren die vanuit de grafieken beschikbaar worden gemaakt is het duidelijk wanneer en waar er dieren moeten worden weggehaald of bijgeplaatst.

Desondanks dat de applicatie nog niet helemaal perfect is, kan deze dus wel gebruikt worden waarvoor het bedoeld is. Om de eventuele fouten er nog uit te halen is simpelweg meer tijd nodig.

# Literatuurlijst

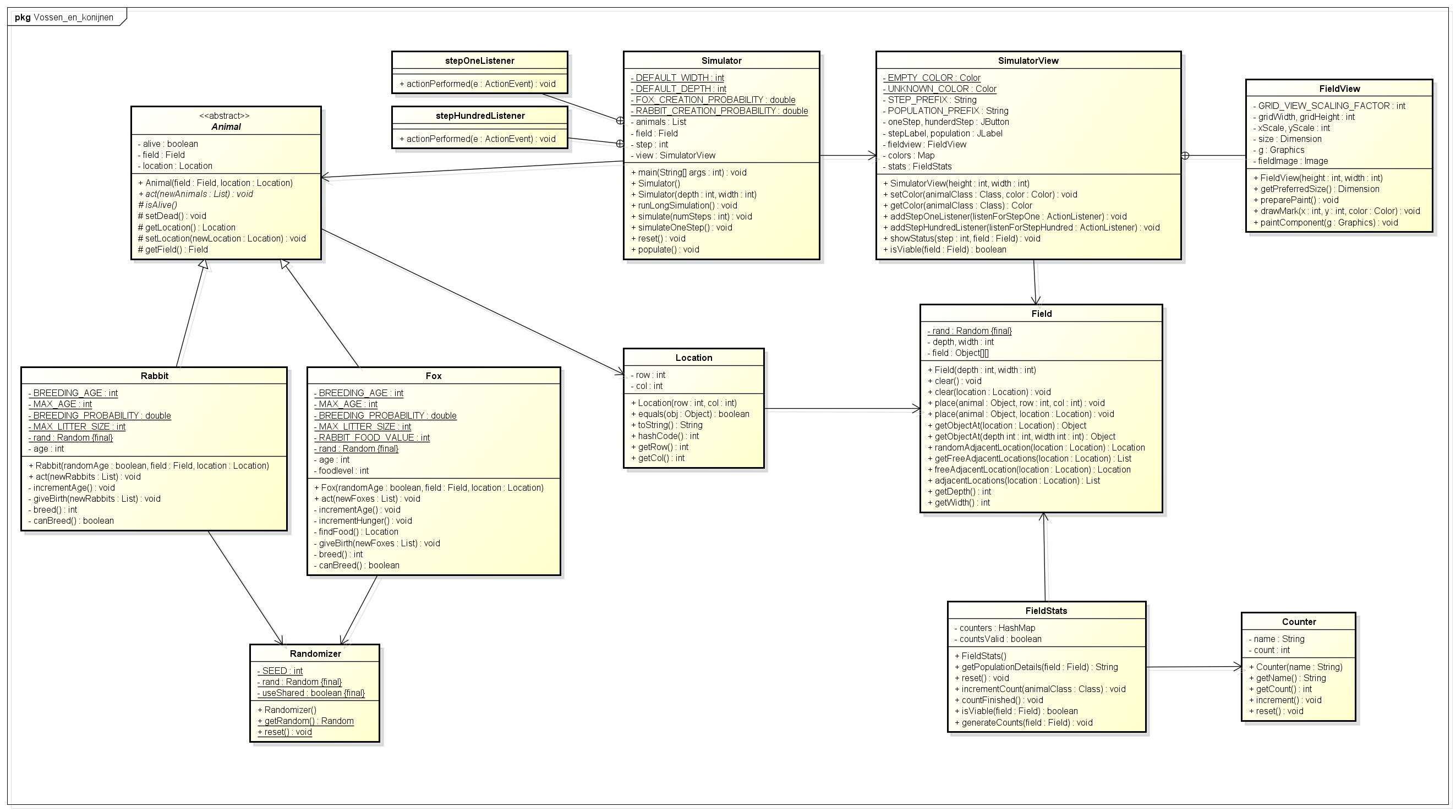
Sijs, N. v. (2010). *Etymologiebank Varkencyclus*. Opgehaald van Etymologiebank: http://www.etymologiebank.nl/trefwoord/varkenscyclus

Warmer, J. (2011). *Praktisch UML.* Amsterdam: Pearson.

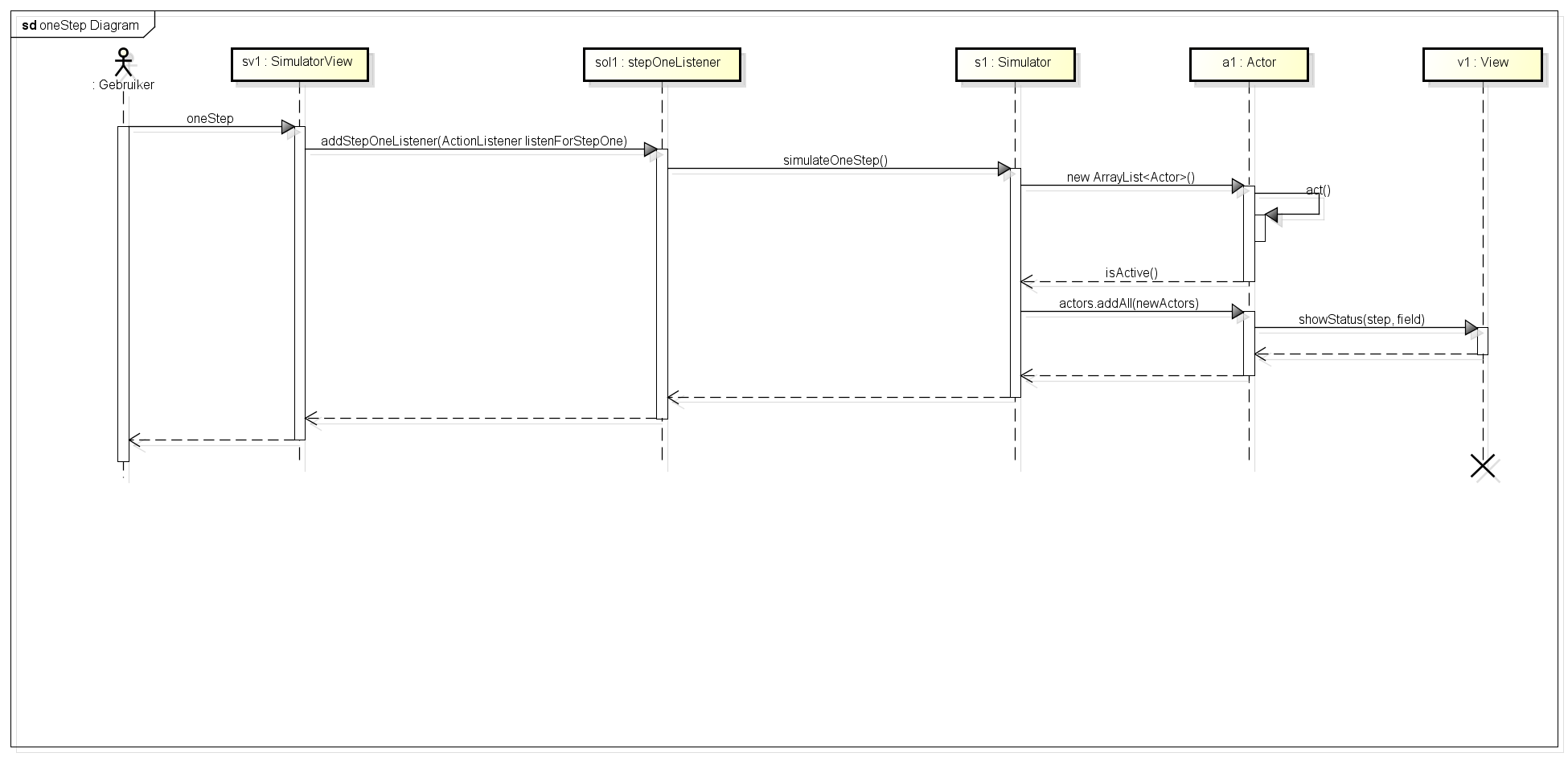
Knispel, K. (2012). *Zakeljke communicatie Schriftelijk.* Amsterdam: Pearson.

# Bijlagen

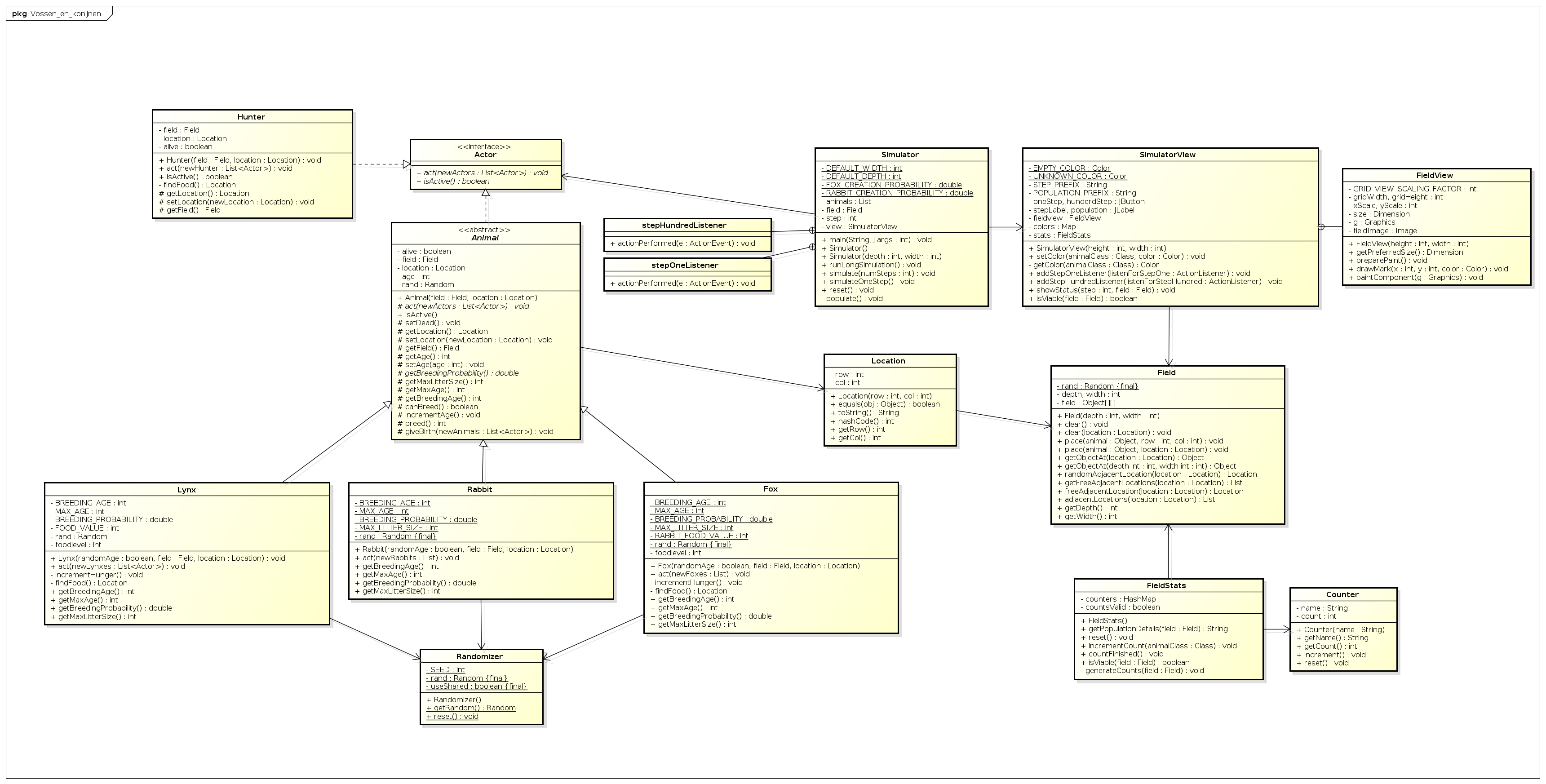
## Bijlage A (Klassendiagram oude situatie)



## Bijlage B (Sequencediagram)



## Bijlage C (Klassendiagram nieuwe sitatie)



## Bijlage D (MVC Klassendiagrammen)

## Bijlage E (MVC Sequencediagram)

