Rapportage project vossen en konijnen





Inhoudsopgave

[Inleiding 3](#_Toc410121101)

[1. Probleemstelling 3](#_Toc410121102)

[1.1 Reden van ontwikkeling 3](#_Toc410121103)

[1.2 Geschikt voor deze personen/organistaties 3](#_Toc410121104)

[2. Analyse van huidige situatie 3](#_Toc410121105)

[2.1 Beperkingen 3](#_Toc410121106)

[2.2 Voorgestelde oplossingen 4](#_Toc410121107)

[2.3 Beschikbare gegevens van huidige situatie 4](#_Toc410121108)

[2.4 Varkenscyclus 5](#_Toc410121109)

[3. Uitbreiding 1 5](#_Toc410121110)

[3.1 Veranderingen 5](#_Toc410121111)

[3.2 Klassendiagram oude situatie 6](#_Toc410121112)

[3.3 Sequencediagram 6](#_Toc410121113)

[3.4 Klassendiagram nieuwe sitatie 7](#_Toc410121114)

[4. Het gebruik van MVC in V&K 7](#_Toc410121115)

[4.1 Overeenkomsten en verschillen mvc-projecten 7](#_Toc410121116)

[4.1.1 Overeenkomsten en verschillen tussen MVCDynamicModelThread en -ThreadGeneralized 7](#_Toc410121117)

[4.1.2 Overeenkomsten en verschillen tussen MVCDynamicModelThreadGeneralized en Life 7](#_Toc410121118)

[4.2 Klassendiagram 8](#_Toc410121119)

[4.3 Sequencediagram 8](#_Toc410121120)

[5. Uitbreiding 2 8](#_Toc410121121)

[5.1 Nieuwe toevoegingen 8](#_Toc410121122)

[5.1.1 Gras 8](#_Toc410121123)

# Inleiding

Dit is een rapport geschreven om een goed beeld te geven hoe de ontwikkeling van de applicatie is verlopen. Hierin worden een aantal aspecten behandeld; zoals waarom de applicatie is gebouwd, voor wie de applicatie is gebouwd en welke veranderingen er zijn toegepast.

De applicatie die ontwikkeld gaat worden is een simulatie om dierenpopulaties in bij te houden. De daadwerkelijke reden van ontwikkeling wordt behandeld in paraaf 1.1.

De applicatie zoals is 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 uitbreiding per week besproken.

# 1. Probleemstelling

In dit hoofdstuk wordt er dieper gekeken naar voor wie, en waarom deze applicatie is ontwikkeld. Verder wordt er gekeken naar welke gegevens er nu beschikbaar zijn.

## 1.1 Reden van ontwikkeling

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 zo makkelijk en tijdig kan bepalen of er van een bepaalde populatie dieren moeten worden bijgezet of juist moeten worden weggehaald.

Omdat het hier niet om vaste gegevens gaat word het mogelijk gemaakt voor Staatbosbeheer om zelf de gegevens aan te passen. Denk hierbij aan het aantal dieren, en welke populaties er in de simulatie moeten worden meegenomen.

## 1.2 Geschikt voor deze personen/organistaties

Deze applicatie wordt ontwikkeld voor Staatsbosbeheer. Deze applicatie beperkt zich niet alleen voor gebruik bij Staatsbosbeheer. Iedereen die interesse heeft in het verloop van dierenpopulaties kan hier gebruik van maken. Door de zwakke afhankelijkheid van de code is het makkelijk om andere dieren toe te voegen i.p.v. die standaard geleverd worden.

# 2. 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 ook geïmplementeerd.

## 2.1 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 bied 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 ook maar te resetten. Hierdoor vertoont de applicatie soms gedrag dat niet wenselijk is.

## 2.2 Voorgestelde oplossingen

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

* De maximale leeftijd van dieren
* De verhouding van de dieren onderling
* Een minimale leeftijd waarop een dier zich kan voortplanten
* De dierenpopulaties kunnen 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 kunnen wordt het makkelijker om uitbreidingen toe gaan voegen. Daarnaast zou de code ook veel beter leesbaar 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.

## 2.3 Beschikbare gegevens van huidige situatie

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

Verticale as zijn de aantallen, en de horizontale as zijn de stappen

De dierenpopulaties zijn over het algemeen stabiel te noemen. Bij de konijnenpopulaties kunnen is het duidelijk te zien 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 echt representatief. In de bovenstaande grafiek is namelijk niet echt duidelijk een verband te zien tussen het de populatie van konijnen en vossen.

Deze is er echter wel. In paragraaf 2.4 wordt dit duidelijk uitgelegd aan de hand van een voorbeeld en waarom dit zo is.

## 2.4 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 weer 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 vervolgens weer tekort, waardoor de prijzen juist 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. Daarin tegen als het aantal konijnen hoog is, zijn er minder vossen.

# 3. 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.

## 3.1 Veranderingen

Tijdens de 2de week dat deze applicatie ontwikkeld werd, zijn er een aantal verbetering toegevoegd. Er is een nieuwe diersoort toegevoegd, namelijk de lynx. De lynx is een diersoort dat jaagt op konijnen en vossen. De eigenschappen van een lynx zijn strenger als van de vossen, dit is gedaan omdat een lynx makkelijker aan voedsel kan komen. Met de loop van de tijd worden er nog uitbreidingen toegevoegd waarmee het mogelijk wordt om deze eigenschappen zelf in te stellen.

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**();**

**}**

Verder is er nog een jager toegevoegd. 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.

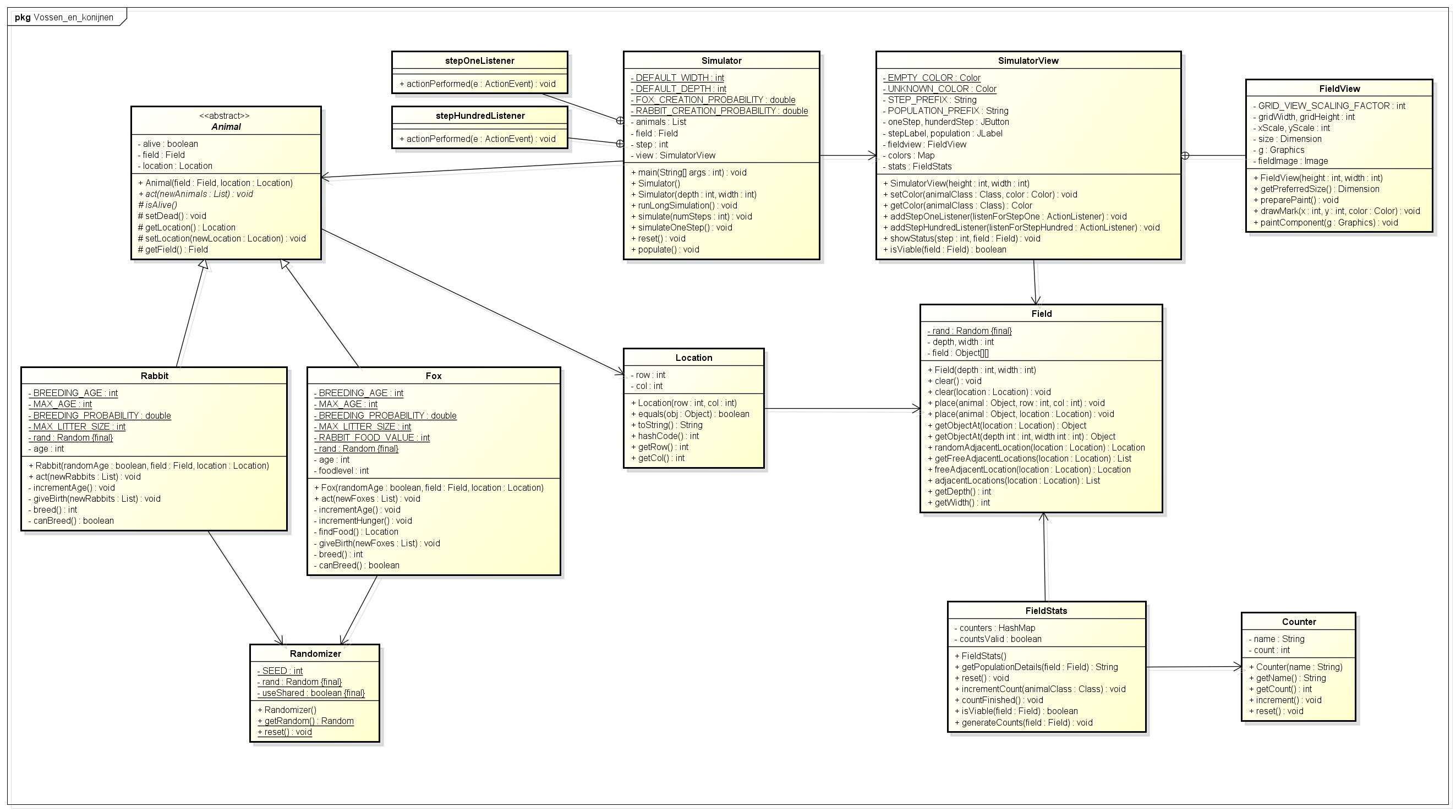
Verder is er nog een interface aangemaakt genaamd Actor. Deze interface bevat alle dieren, objecten en personen die aanwezig zijn in de simulatie. Door het aanmaken van deze interface is ervoor gezorgd dat alles nu in een lijst staat. Het was ook mogelijk om per groep te doen, maar dit zou de code nodeloos ingewikkeld maken, en hierdoor wordt het een stuk moeilijker om andere functionaliteit toe te voegen. De code van de interface Actor staat hiernaast

De code van de nieuwe interface Actor

Ook 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 nu beter leesbaar en een stuk onafhankelijker geworden.

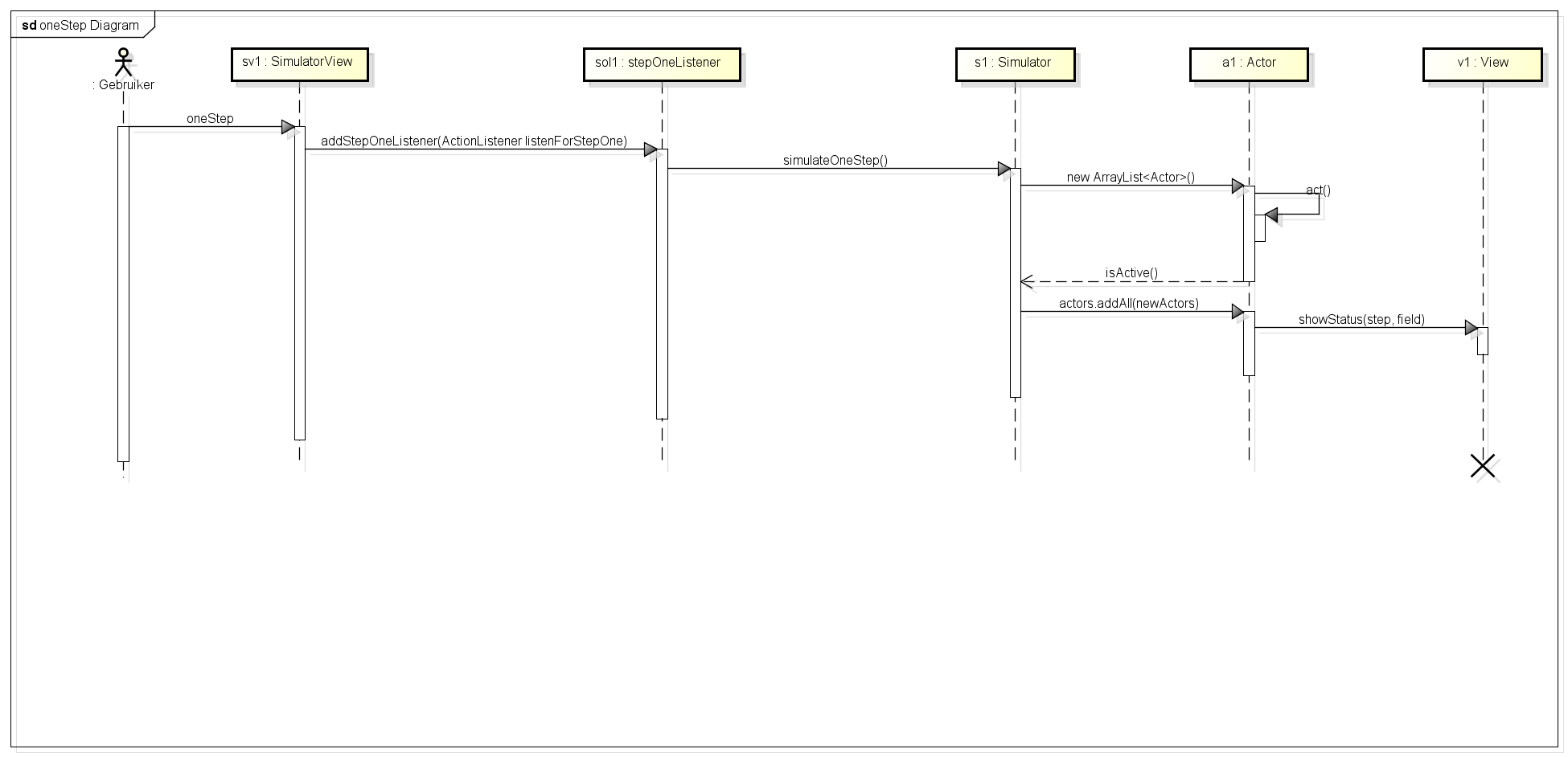
Tot slot zijn er op diverse plekken kleine wijzigen doorgevoerd, maar deze zijn niet de moeite waard om te afzonderlijk te bespreken.

## 3.2 Klassendiagram oude situatie



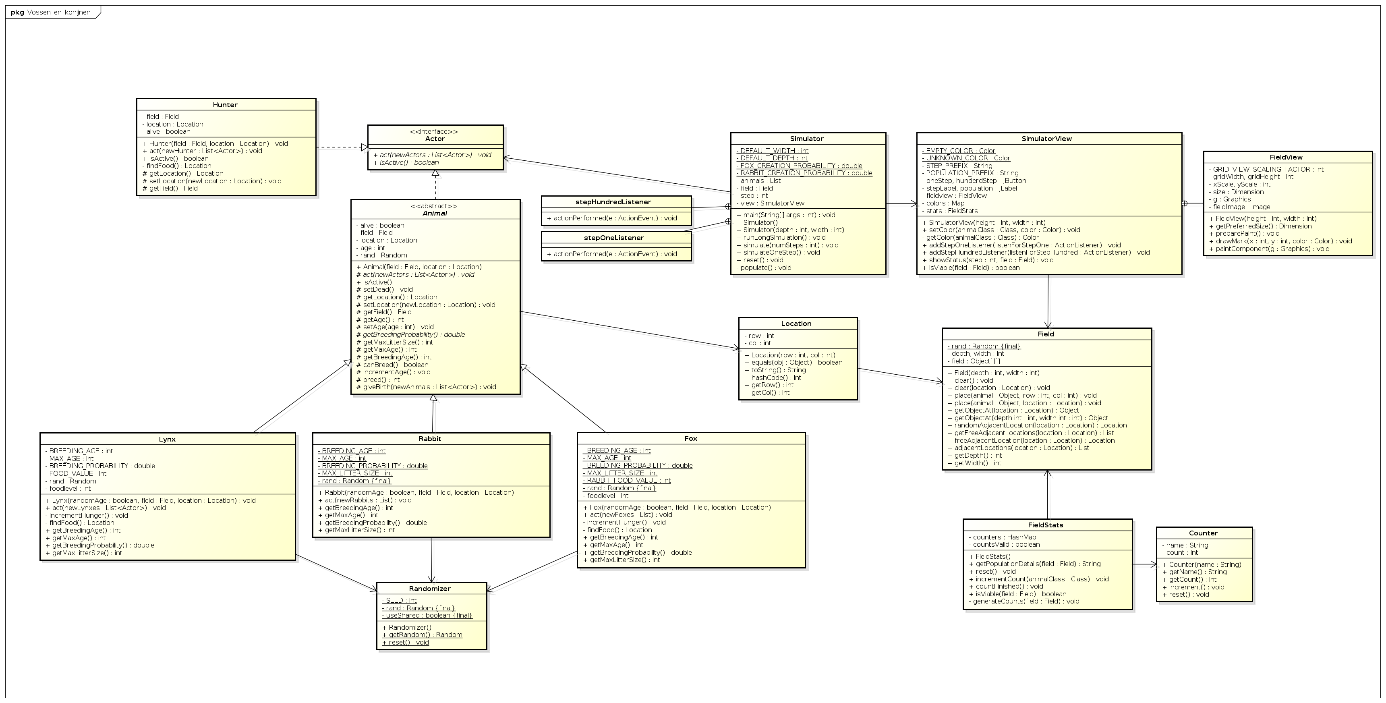
Een klassendiagram van de eerste versie van de applicatie

## 3.3 Sequencediagram



Een sequence diagram van 1 stap in de simulatie

## 3.4 Klassendiagram nieuwe sitatie



Een klassendiagram van de nieuwe situatie, met nieuwe verbeteringen

# 4. Het gebruik van MVC in V&K

## 4.1 Overeenkomsten en verschillen mvc-projecten

### 4.1.1 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.

### 4.1.2 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. Nog een 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.

## 4.2 Klassendiagram

## 4.3 Sequencediagram

# 5. Uitbreiding 2

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

## 5.1 Nieuwe toevoegingen

### 5.1.1 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 het gebeurd bij dieren, 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.