Rapportage project vossen en konijnen



Inhoudsopgave

[1. Probleemstelling 3](#_Toc409080555)

[1.1 Inleiding hoofdstuk 3](#_Toc409080556)

[1.2 Waarom ontwikkelen we deze applicatie? 3](#_Toc409080557)

[1.3 Voor wie ontwikkelen we deze applicatie? 3](#_Toc409080558)

[2. Analyse van huidige situatie 3](#_Toc409080559)

[2.1 inleiding Hoofdstuk 3](#_Toc409080560)

[2.2 Beperkingen van huidige situatie 3](#_Toc409080561)

[2.3 Eventuele oplossingen voor beperkingen 3](#_Toc409080562)

[2.4 Analyse van beschikbare gegevens 4](#_Toc409080563)

[2.4.1 Varkenscyclus 4](#_Toc409080564)

# 1. Probleemstelling

In dit hoofdstuk wordt duidelijk gemaakt waarom deze applicatie ontwikkeld wordt, en voor wie deze geschikt is.

## 1.1 Reden van ontwikkeling

Met deze applicatie wordt aangetoond hoe verschillende dierenpopulaties zich in de natuur verhouden. Het onderzoek zou voornamelijk worden gebruikt om te kijken hoe populaties zich ontwikkelen na verloop van tijd. Dit kan van belang zijn om te bepalen of dieren moeten worden bijgeplaatst of juist moeten worden weggehaald. Het idee is dat deze applicatie zo wordt opgebouwd dat deze kan worden hergebruiken voor andere dierenpopulaties. Centraal probleem bedenken

## 1.2 Geschikt voor deze personen/organistaties

Deze applicatie is geschikt voor iedereen die zich bezig houdt met populaties in de natuur. Iedereen die interesse heeft in de data, kan en mag deze applicatie gebruiken. Betere definitie voor we het bedoeld is

# 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 moet gerefactord worden om meer overzicht te houden, en om makkelijker uitbreidingen toe te voegen. Meer beperkingen, duidelijkheid van code

## 2.2 Eventuele oplossingen

De in hoofdstuk 2.2 gestelde problemen 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.

## 2.3 Analyse van beschikbare gegevens

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

Hierin valt duidelijk te zien dat de populaties nogal op en neer schommelen. Waar het bij de konijnen om een redelijk stabiele daling en stijgingen gaat, is dat bij de vossen niet echt terug te zien. Hieruit valt ook af te leiden dat zodra de populatie van konijnen beneden de 2000 zakt, de aantallen weer sterk aantrekken. Deze tekst klopt niet in vergelijking met sectie 2.4

## 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

## 3.1 Veranderingen

## 3.2 Klassendiagram oude situatie

## 3.3 Sequencediagram

## 3.4 Klassendiagram nieuwe sitatie

# 4. Het gebruik van MVC in V&K

## 4.1 Overeenkomsten en verschillen mvc-projecten

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.

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