|  |  |
| --- | --- |
| Reflectie-verslag  Leertaak 2: Project Vossen & Konijnen | oVER  Het reflectieverslag van het project Vossen & Konijnen. Dit project wordt uitgevoerd in periode twee van het eerste jaar.  Door  Frank Noorlander, Rick van der Poel en Tsjeard de Winter |

Reflectie-verslag

Leertaak 2: Project Vossen & Konijnen



**Auteurs:** Frank Noorlander, Rick van der Poel en Tsjeard de Winter

**Opdrachtgever**: Dhr. Heijnen

**Organisatie**: Hanzehogeschool Groningen

**Datum** 24-01-2015

**Locatie**: Groningen

# Samenvatting

Inhoudsopgave

[Samenvatting 2](#_Toc409877482)

[Verklarende woordenlijst 4](#_Toc409877483)

[Voorwoord 5](#_Toc409877484)

[probleemstelling 6](#_Toc409877485)

[Analyse huidige situatie 6](#_Toc409877486)

[Hoofdstuk 1 7](#_Toc409877487)

[Hoofdstuk 2 8](#_Toc409877488)

[Hoofdstuk 3 9](#_Toc409877489)

[Conclusie 10](#_Toc409877490)

[Advies / aanbevelingen / actieplan 11](#_Toc409877491)

[Noten 12](#_Toc409877492)

[Literatuurlijst 13](#_Toc409877493)

[Bijlage 1 14](#_Toc409877494)

[Bijlage 2 15](#_Toc409877495)

[Register 16](#_Toc409877496)

# Verklarende woordenlijst

MVC = Een programmeer patroon waarin de software uit de onderdelen model, view en controller bestaan.

Model = Verwerkt alle informatie.

View = Toont alle informatie van het model op het scherm.

Controller = ontvangt alle gebruikersinvoer en geeft dit door aan het model.

V&K = Vossen & Konijnen

Package = geeft een groep bestanden weer. Een package kan ook nieuwe packages bevatten.

Subclasse = een klasse die een andere klasse die een andere klasse uitbreidt.

# Voorwoord

Voor u ligt het reflectieverslag van het project Vossen & Konijnen. Het project wordt uitgevoerd in de tweede periode van het eerste schooljaar, in opdracht van de Hanzehogeschool. De projectleden zijn: Frank Noorlander, Rick van der Poel en Tsjeard de Winter

Dit verslag zal een reflectie zijn op de werkzaamheden die we als projectgroep hebben verricht om dit project tot een succes te brengen. Meer informatie over het project vindt u in de volgende hoofdstukken.Inleiding

# Probleemstelling

Onze opdrachtgever; de Hanze Hogeschool heeft veel verzoeken gekregen om een applicatie te maken waarmee het de levenscyclus van dierenpopulaties kan simuleren. Omdat er tegenwoordig veel meer aandacht is voor de natuur en het behouden ervan zijn er verschillende partijen geïnteresseerd in zo’n applicatie.

De opdrachtgever heeft zelf al een poging gedaan om een dergelijke applicatie te maken. Deze applicatie genaamd ‘*Vossen & Konijnen’* voldoet echter niet aan de eisen om een goede simulatie te genereren.

Als projectgroep hebben we de taak gekregen om de huidige applicatie van de opdrachtgever aan te vullen en te verbeteren om tot een realistisch resultaat te komen. De meegegeven applicatie is momenteel erg simpel, dat betekend dat we ons momenteel alleen zullen gaan richten op een bosgebied met daarin vossen en konijnen. Later tijdens dit project zal de applicatie echter uitgebreid worden met meer functionaliteiten.

Als de applicatie een succes wordt, kan het gebruikt kunnen worden om veel verschillende dierenpopulaties te simuleren en daarmee een enorme schat aan informatie te verkrijgen. Deze informatie kan op zijn beurt helpen de natuur en haar inwoners te behouden.

# Analyse huidige situatie

In de huidige versie van het programma dat hebben gekregen van de opdrachtgever is het mogelijk om via een visuele simulator het leven van vossen en konijnen te simuleren.

De vossen en konijnen hebben de volgende eigenschappen:

* Beide dieren hebben een maximale leeftijd. Als deze leeftijd overschreden wordt zullen de dieren verdwijnen uit het programma.
* Beide dieren hebben de mogelijkheid om te paren.
* De vossen kunnen de konijnen opeten als deze naast elkaar komen te staan in de simulator.
* De vossen kunnen dood gaan als ze te lang geen konijnen gegeten hebben.

Zoals hier boven al blijkt, is de applicatie nog erg kaal. De enige overlevingscondities waar de applicatie momenteel rekening mee houdt zijn de leeftijd en de honger. Er wordt bijvoorbeeld geen rekening gehouden met weersomstandigheden, andere dieren die hoger in de voedselketen staan zoals beren of jagers, vervuiling enzovoort.

Het doel van deze applicatie is om de levenscyclus van dierenpopulaties zo goed en realistisch mogelijk te simuleren. Om dat doel te kunnen realiseren moet de applicatie uitgebreid worden met nieuwe elementen.

Verbeteringen aan het programma

MVC

De applicatie zoals deze aangeleverd is bevatte naast het gebrek aan functionaliteit ook gebrek aan flexibiliteit. Wanneer er een functie toegevoegd of juist gewijzigd moest worden zou er teveel veranderd moeten worden waardoor dit veel tijd in beslag zou nemen. Er is gekozen om het programma flexibeler te maken door het MVC-model toe te passen. Door dit model veranderd er niks aan het uiterlijk van de applicatie maar het is wel duidelijk merkbaar tijdens het wijzigen of toevoegen van functies.

**Wat is MVC?**

MVC staat voor *Model, View, Controller* en is een ontwerppatroon dat het ontwerp van toepassingen opdeelt in drie onderdelen met elk hun eigen verantwoordelijkheden. Door het opdelen wordt de leesbaarheid en herbruikbaarheid van de code vergroot.

Het belangrijkste onderdeel is het *model*. Deze zorgt voor de informatie door de applicatie logica toe te passen op de data. Het model weet welke *views* bij hem horen en geeft steeds een melding aan al deze *views* dat de informatie veranderd is.

De *view* zorgt dat de informatie op het scherm getoond wordt. Wanneer de *view* gemaakt wordt, geeft deze door aan een *model* dat hij van zijn informatie afhankelijk is. De *view* doet zelf geen verwerking van de gegevens en geeft ook niks aan het *model* door.

De *controller* zorgt voor de invoer van gebruikers. Alle knoppen, tekstvelden, menu’s, etc. zijn dus van de controller. De controller wordt aan één of meerdere *models* verbonden en geeft elke gebruikers invoer direct door aan het *model* zonder iets met de informatie te doen.

Wanneer deze drie onderdelen goed geïmplementeerd zijn kan je heel makkelijk een nieuwe view toevoegen of weghalen zonder dat de applicatie daardoor niet werkt. Maar dit geldt ook voor controller en view hoewel deze vaak wat in elkaar verwikkeld zijn, de kunst is om dit zo beperkt mogelijk te houden.

**MVC in Vossen en Konijnen**

In de applicatie moest er op minstens vijf plekken code gewijzigd worden voordat er een nieuwe view toegevoegd kon worden. Een nieuwe view zou bijvoorbeeld een cirkel- of staafdiagram kunnen zijn. Daarom is er in de vernieuwde applicatie gekozen om het MVC-model te implementeren. Dit is gedaan door eerst een duidelijke packagestructuur te maken en daarna de bestaande classes op te splitsen in de drie onderdelen.

De packagestructuur is veranderd om meer overzicht te krijgen en optimaal gebruik te kunnen maken van de opties die Java biedt. De eerste drie packesages (hanze > PIV1E > leertaak2) zijn om duidelijk te maken van welk bedrijf de code is en waar de code voor dient (zie figuur 4.1). In dit geval is de code van de Hanze en het project is leertaak2. Binnen leertaak2 zijn de drie onderdelen van het MVC-model opgenomen in de packages model, view en controller. Alle packages met uizondering van de main-package ondersteuen de *model*.



Figuur 1 De packagestructuur van V&K

Na het aanbrengen van de packagestructuur kon deze gevuld worden. Veel klassen bestonden uit delen die zowel de UI deden als het verwerken van de informatie die getoond moest worden. Deze klassen moesten uit elkaar worden gehaald en opgedeeld worden in hun eigen onderdelen, *model* en *view* in dit geval.

Wanneer alle klassen één onderdeel van het MVC-model bevatten kunnen deze aan elkaar gekoppeld worden. Dit wordt gedaan in de klasse *Simulator.java* in de main-package. Hieronder ziet u een fragment code uit Simulator.java waarin de onderdelen aan elkaar gekoppeld worden.

simulation = new SimulationModel(DEFAULT\_DEPTH, DEFAULT\_WIDTH);

view = new SimulatorView(DEFAULT\_DEPTH, DEFAULT\_WIDTH, simulation);

…

graph = new GraphView(140, 140, 100, simulation);

…

pie = new PieView(140, 140, simulation);

…

chart = new ChartView(140, 140, simulation);

…

RunController controller = new RunController(simulation);

menuController = new MenuController(simulation);

Tekstfragment 1 Codefragment uit *Simulator.java*

Eerst wordt het model gemaakt, *simulation* in dit geval. Deze voert alle berekeningen uit en maakt houdt bij wat er in welk hokje van de simulatie zit. Daarom krijgt deze het aantal vakjes waaruit de simulatie bestaat mee.

Daarna wordt de belangrijkste view aangemaakt. *View* laat de informatie die *simulation* heeft zien, om deze informatie op te halen moet de view weten van welk model hij deze vandaan moet halen. Daarom krijgt *view* als laatste parameter *simulation* mee. Dan worden er drie views gemaakt die alle informatie op een overzichtelijke manier weergeven. Elke view heeft een afmeting nodig en het model waarvan deze zijn informatie vandaan moet halen. Die drie views zijn: *graph*, *pie* en *chart*. Als laatst worden er 2 controllers aangemaakt, deze hoeven allen te weten waar ze de gebruikers invoer naar toe moeten sturen en krijgen daarom ook *simulation* mee.

Elke view is een subclasse van *AbstractView*, in deze klasse staat een constructor die er voor zorgt dat elke view aangemeld word bij het model. Het fragment code hieronder laat deze constructor zien.

public AbstractView(AbstractModel simulation) {

this.simulation = simulation;

simulation.addView(this);

}

Tekstfragment 2 Codefragment uit AbstractView.java

Elk model is een subclasse van *AbstratModel* en hierin staan onder andere de methoden voor het toevoegen en notificeren van views. Het fragment code hieronder laat deze methodes zien.

public void addView(AbstractView view) {

views.add(view);

}

public void notifyViews() {

for(AbstractView v: views) v.updateView();

}

Tekstfragment 3 Codefragment uit AbstractModel.java

Op deze manier wordt in Vossen & Konijnen het MVC-model gebruikt.

Verplichte uitbreidingen

Bonus uitbreidingen

Conclusie

Advies / aanbevelingen / actieplan

Noten

Literatuurlijst

Bijlage 1

Bijlage 2

Register