PROJECT VOSSEN & KONIJNEN

ADVIESRAPPORT



ICT Bureau Ambler  
Frank Noorlander, Tsjeard de Winter en  
Rick van der Poel

PROJECT VOSSEN & KONIJNEN   
ADVIESRAPPORT

**Auteurs:** ICT Bureau AmblerFrank Noorlander, Tsjeard de Winter en Rick van der Poel   
  
**Opdrachtgever**: de heer H. Ripper van SimulationFirst   
  
**Begeleiding:** de heer B. Heijne van Hanzehogeschool Groningen  
 **Datum:**  02-2015  
  
**Locatie**: Groningen  
  
**Bron afbeelding voorpagina:**  
Rick van der Poel. (2015, 30 januari). Screenshot Vossen & Konijnen applicatie [Leertaak 2].  
Hanzehogeschool Groningen, opleiding Informatica.

# Samenvatting

In dit document vindt u de documentatie van het project Vossen & Konijnen. Deze samenvatting geeft u een indruk van de werkzaamheden die verricht zijn tijdens het project.

De opdrachtgever heeft ons van een incomplete applicatie voorzien waarin het gedrag van vossen en konijnen wordt gesimuleerd. De simulatie miste echter nog veel functionaliteiten. De originele applicatie kan alleen vossen en konijnen simuleren en de opdrachtgever wil graag dat ook andere dieren gesimuleerd kunnen worden. Daarnaast moet de GUI van de applicatie zo uitgebreid worden dat er meer informatie over de simulatie in beeld komt voor de gebruiker. In de gewenste GUI moet het ook mogelijkheid zijn de parameters van de simulatie te kunnen wijzigen. De simulatie wordt verder nog uitgebreid met diverse andere onderdelen zoals plaatjes, geluiden en ziektes.  
  
Als eerste is het programma zo aangepast dat het gebruikt kan worden door meer programmeerprogramma’s. Ook is de GUI uitgebreid met een menubalk en knoppen voor het bedienen van de applicatie. Daarnaast zijn er verschillende unittests aangemaakt die testen of de functionaliteit van functies uit het originele programma niet beïnvloed worden door de aanpassing die gaan komen.

De eerste grote uitbreiding van het programma was het introduceren van nieuwe wezens aan de simulatie. De klassen zijn zo aangepast dat er nieuwe dieren en mensen aan de simulatie toegevoegd kunnen worden. Vervolgens zijn beren en jagers toegevoegd aan de simulatie die elk hun eigen invloed hebben op de simulatie. Beren jagen op vossen en konijnen en jagers jagen op de dieren waar er teveel van zijn. De wijzigingen in de applicatie wordt ondersteund met verschillende UML-diagrammen die gemaakt zijn tijdens en na het project. Deze diagrammen geven u een goed beeld van de werkzaamheden die zijn verricht tijdens het project.

Daarna waren de nieuwe simulatie weergaven aan de beurt. Om deze zo genoemde views beter te kunnen implementeren is de hele applicatie gerefactored. Hierdoor is er een betere cohesie tussen de klassen en word ook het implementeren van nieuwe functionaliteit veel makkelijker. Om dit te realiseren is er gebruik gemaakt van een package structuur en is er een MVC-model toegepast tijdens het refactoren.

Na het refactoren is het parameter paneel er ingekomen waarmee het gedrag van de dieren beïnvloed kan worden. Ook is het programma zo aangepast dat het nu ook rekening houd met de voedselvoorraad van de konijnen. Als er veel konijnen zijn komen er minder nakomelingen dan wanneer er weinig konijnen zijn omdat er dan meer gras beschikbaar is. Verder is het programma nog uitgebreid met geluiden en een toerist die af en toe even langs komt om een brand te stichten.

Tot slot zijn er nog een aantal bonus uitbreidingen toegevoegd waaronder een konijnen ziekte die ook invloed heeft om de andere dieren. Konijnen die deze ziekte opvangen gaan vervolgens dood na 5 dagen. Zodra een ziek konijn opgegeten wordt door een vos raakt de vos geïnfecteerd. Word de vos vervolgens weer opgegeten door een beer dan wordt de beer geïnfecteerd en zal ook de beer last hebben van de ziekte.

Door al deze verbeteringen is de applicatie efficiënter en veelzijdiger geworden met veel ruimte tot nog meer uitbreidingen. De gebruiker kan makkelijker met de simulatie overweg en krijgt ook veel meer informatie terug tijdens de simulatie.

Inhoudsopgave

[Samenvatting 2](#_Toc410501495)

[1. Inleiding 6](#_Toc410501496)

[2. Probleemstelling 7](#_Toc410501497)

[3. Analyse huidige situatie 8](#_Toc410501498)

[4. Uitbreiding 1 9](#_Toc410501499)

[4.1 Gebruik buiten BleuJ 9](#_Toc410501500)

[4.2 Ontwikkeling van een simpele GUI 9](#_Toc410501501)

[4.3 Unittests 10](#_Toc410501502)

[4.4 Klasse- en sequentiediagram na uitbreidingen 10](#_Toc410501503)

[4.5 Actor interface, beren en jagers 12](#_Toc410501504)

[5. MVC 13](#_Toc410501505)

[5.1 Wat is MVC? 13](#_Toc410501506)

[5.2 MVC in Vossen en Konijnen 13](#_Toc410501507)

[6. Uitbreiding 2 17](#_Toc410501508)

[6.1 Het parameter paneel 17](#_Toc410501509)

[6.2 Voedselvoorraad van de konijnen 17](#_Toc410501510)

[6.3 Geluiden en plaatjes 18](#_Toc410501511)

[6.4 De toerist 18](#_Toc410501512)

[7. Bonus uitbreidingen 19](#_Toc410501513)

[7.1 Konijnenziekte 19](#_Toc410501514)

[7.2 Opstarten vanaf de commandline 20](#_Toc410501515)

[8. Conclusie en aanbevelingen 21](#_Toc410501516)

[Literatuurlijst 22](#_Toc410501517)

[Bronnen 22](#_Toc410501518)

[Bijlage 1 23](#_Toc410501519)

[Bijlage 2 24](#_Toc410501520)

Verklarende woordenlijst

Refactoren Een programma of een stuk code herschrijven zodat dit efficiënter wordt.   
  
BlueJ Programma om Java in te programmeren.   
   
Eclipse Programma om o.a. Java in te programmeren.  
  
GUI De grafische gebruikersomgeving waarin de gebruiker werkt.  
  
V&K Vossen & Konijnen.

MVC Een programmeermodel 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.

Package Geeft een groep bestanden weer. Een package kan ook andere packages bevatten.

Subclasse Een klasse die een andere klasse uitbreidt.

Voorwoord

Dit adviesrapport is onderdeel van leertaak 2 Project Vossen & Konijnen. Het project wordt uitgevoerd in de tweede thema 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.

We willen graag de heer B. Heijne bedanken voor zijn begeleiding op het technische vlak van de applicatie. Ook bedanken we graag mevrouw W. van Veen die taal en schrijfkundige ondersteuning heeft geleverd tijdens het schrijven van dit rapport.

# 1. Inleiding

Voor u ligt het adviesrapport waarin verslag wordt uitgebracht over verbeteringen die zijn aangebracht aan de Vossen & Konijnen applicatie in opdracht van SimulationFirst. De Vossen & Konijnen (V&K) applicatie is bedoeld voor het simuleren van vossen en konijnen zodat hun gedrag bestudeerd kan worden. De originele applicatie mist echter veel functionaliteit en ICT bureau Ambler is ingeschakeld om de applicatie uit te breiden. SimulationFirst heeft de wens dat de applicatie wordt uitgebreid met een aantal onderdelen die de functionaliteit van de applicatie zullen verbeteren.

In dit rapport worden de verbeteringen in verschillende hoofstukken beschreven.

Hoofdstuk 2 beschrijft uitgebreid de probleemstelling van de opdrachtgever en hoe dit opgelost moet worden.

In hoofdstuk 3 wordt er uitgebreid gekeken naar de huidige situatie van de applicatie. Er wordt beschreven hoe de applicatie werkt en wat het doel is van de applicatie.

Vervolgens worden de eerste uitbreidingen besproken in hoofdstuk 4. In dit hoofdstuk komen de uitbreiding van de GUI en de toevoeging van andere dieren en mensen komen aan bod.

Daarna wordt in hoofdstuk 5 gesproken over het toepassen van een MVC structuur in de applicatie en wat de voordelen hiervan zijn.

Hoofdstuk 6 behandeld de tweede groep uitbreidingen binnen in de V&K applicatie. Hierin komt het parameter paneel, de voedselvoorraad van de konijnen en het gebruik van plaatjes en geluiden aan bod.

In hoofdstuk 7 komen nog een aantal bonus uitbreidingen aan bod. Er wordt hier geschreven over een ziekte die is toegevoegd, hoe het programma van de commandline opgestart kan worden en vertellen we over de toerist die we hebben ingebouwd.

Na deze hoofdstukken volgt een conclusie met een advies voor de opdrachtgever. Vervolgens vindt u nog de literatuurlijst en de bijbehorende bijlagen.

# 2. Probleemstelling

De opdrachtgever SimulationFirst heeft een simulatieapplicatie waarin de levenscyclus van vossen en konijnen gesimuleerd kan worden. Deze applicatie voldoet echter helaas niet aan de eisen van hun cliënten en moet uitgebreid worden met meerdere verschillende functionaliteiten. Dit hoofdstuk beschrijft de huidige problemen in de applicatie.

Hieronder volgt een opsomming van deze problemen:

* Gebruikers kunnen de simulatie niet beïnvloeden via de GUI.
* Er moeten dieren en mensen toegevoegd worden aan de applicatie.
* De applicatie is niet voorzien van unittests die testen of het programma functioneert zoals het bedoeld is.
* Er zijn nog geen UML diagrammen van de applicatie die laten zien hoe de applicatie in elkaar zit.
* De applicatie is niet voorzien van een package structuur volgens een MCV model.
* De applicatie moet meer informatie weergeven aan de gebruiker door middel van extra grafieken en overzichten.
* De parameters van de diersoorten moeten in de GUI aangepast kunnen worden.
* De applicatie moet ondersteund worden met afbeeldingen en/of geluiden.

ICT Bureau Ambler heeft de taak gekregen om de huidige applicatie te verbeteren door de applicatie aan te vullen met de gewenste functionaliteiten. Het doel is om de applicatie verder uit te breiden zodat het gebruikt kan worden voor meerdere diersoorten en meer informatie terug geeft aan de gebruiker. Hierdoor 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.

# 3. Analyse huidige situatie

De Vossen & Konijnen applicatie maakt het mogelijk om via een visuele simulator de levenscycli van vossen en konijnen te simuleren. In dit hoofdstuk krijgt u informatie over wat de huidige simulator doet en wat het tekort komt voor de gewenste situatie.

Vossen en konijnen hebben de volgende eigenschappen in de simulator:

* Beide dieren hebben een maximale leeftijd. Als deze leeftijd overschreden wordt zullen de dieren verdwijnen uit de simulatie.
* 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 u ziet zijn de enige overlevingscondities waar de applicatie momenteel rekening mee houdt de leeftijd en honger van de dieren. Er wordt bijvoorbeeld geen rekening gehouden met andere wezens die hoger in de voedselketen staan zoals beren of jagers, ziektes en de voedselvoorraad van de konijnen.

Daarnaast ontbreekt er ook nog veel aan de GUI. De huidige GUI laat alleen een visuele weergave zien van het veld waar de vossen en konijnen in leven. Er zijn geen mogelijkheden om met knoppen de simulatie te beïnvloeden. Hierdoor is het erg moeilijk om de simulatie te volgen.

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

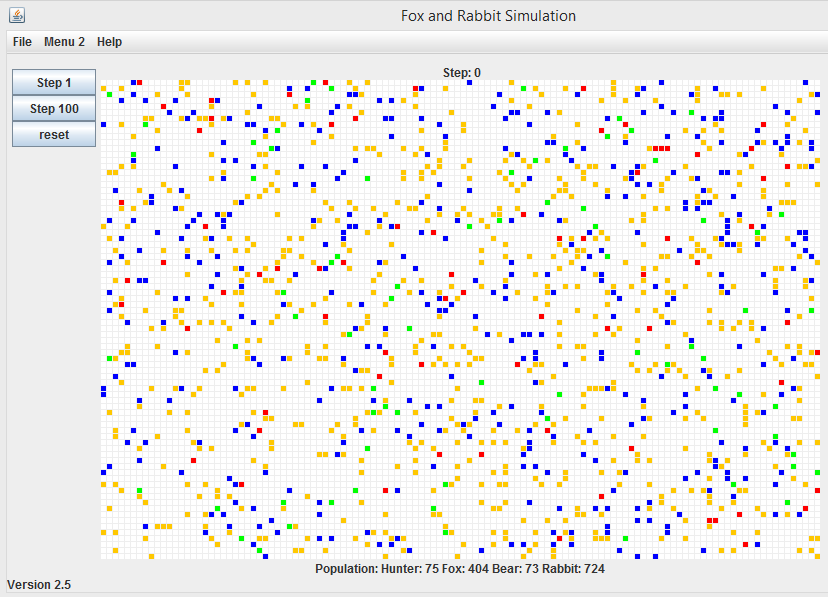
# 4. Uitbreiding 1

De originele versie van de applicatie is verbeterd door een aantal wijzigingen door te voeren en daarna nieuwe onderdelen toe te voegen. In dit hoofdstuk worden de wijzigingen besproken die in de eerste week zijn toegepast.

## 4.1 Gebruik buiten BleuJ

Als eerst is de applicatie aangepast zodat het ook in andere programma’s gebruikt kan worden. Hierdoor is het niet alleen afhankelijk van een programma zoals BleuJ. Door het toevoegen van een main methode kan het programma ook geopend worden in andere programmeer programma’s zoals bijvoorbeeld Eclipse.

## 4.2 Ontwikkeling van een simpele GUI

De applicatie is uitgebreid met een simpele GUI waardoor gebruikers acties kunnen uitvoeren binnen in de applicatie. Ook is er een menu aangemaakt waar basis functionaliteiten ingezet kunnen worden zoals bijvoorbeeld het afsluiten van de applicatie. Daarnaast is er een menu gemaakt met twee knoppen die het uitvoeren van de stappen regelen en een reset knop om te simulatie opnieuw te starten.   
  


Figuur 4.1 – Screenshot van de nieuwe Simulatie GUI

## 4.3 Unittests

Om wijzigingen in het programma te controleren op fouten zijn er ook een aantal unittests aangemaakt. Deze testen kunnen worden uitgevoerd als de applicatie is gewijzigd om te controleren of bepaalde functionaliteit van de applicatie nog steeds goed werkt. Slaagt de applicatie er niet in om door de test te komen dan is er te zien dat er iets mis de code. Omdat unittests specifieke onderdelen controleren kan ook het probleem snel terug gevonden worden.

De volgende unittests zijn gemaakt waarbij vooral het gedrag van de dieren wordt getest:

- Worden de dieren ouder per stap?

- Wordt een konijn opgegeten als het naast een vos staat?

- Sterft een vos ook als er niets te eten is doordat het ingesloten zit?

## 4.4 Klasse- en sequentiediagram na uitbreidingen

In het klassendiagram van de applicatie hieronder (figuur 4.2) ziet u hoe de applicatie er uit ziet na het toevoegen van de veranderingen. In het klassendiagram zijn alle klassen opgenomen samen met alle velden en methodes die er bij horen. Ook ziet u welke klassen iets met elkaar te maken hebben door de lijnen die de klassen met elkaar verbinden.



Figuur 4.2 – Klassendiagram van de V&K applicatie na het toevoegen van de laatste paar uitbreidingen

Naast een klasse-diagram is ook een sequentie-diagram gemaakt. In dit diagram ziet u de interacties tussen verschillende objecten van de applicatie.



Figuur 4.3 - Sequentie-diagram

## 4.5 Actor interface, beren en jagers

Omdat het programma met meerdere diersoorten en zelfs met mensen moet kunnen werken is ook de structuur van de applicatie aangepast zodat dit makkelijker gedaan kan worden in de toekomst. Om dit te realiseren is er een Interface genaamd Actor toegevoegd.

Vervolgens is de klasse Animal abstract gemaakt en is er ook een abstracte klasse Human toegevoegd. Deze klassen implementeren de klasse Actor en zijn op hun beurt weer de super klasse geworden van de wezens die daar bij horen. Om duidelijk de veranderingen weer te geven is er een klassendiagram gemaakt die deze veranderingen weergeeft (zie figuur 4.4).

****

Figuur 4.4 - klassendiagram (Actoren) van de gewenste situatie

# 5. MVC

De huidige applicatie heeft naast het gebrek aan functionaliteit ook het gebrek aan flexibiliteit en dit is vooral zichtbaar in de code. Wanneer er functionaliteit aan de applicatie toegevoegd moet worden kost dit in de huidige applicatie veel tijd en moeite. Er is er voor gekozen om het programma flexibeler te maken door het MVC-model toe te passen. Door het toepassen van dit model veranderd er niets aan het uiterlijk van de applicatie maar is het verschil wel duidelijk merkbaar tijdens het wijzigen of toevoegen van functies.

## 5.1 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 van de onderdelen wordt de leesbaarheid en herbruikbaarheid van de code vergroot.

Het belangrijkste onderdeel is het model. Het model 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 elke view dat er informatie veranderd is.

De view zorgt er voor dat de informatie op het scherm getoond wordt. Wanneer een view gemaakt wordt, geeft de view door aan een model dat hij van zijn informatie afhankelijk is. De view doet zelf geen verwerking van de gegevens en geeft ook niets 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 modellen gekoppeld en geeft de invoer van de gebruiker direct door aan het model zonder iets te doen met deze informatie.

Wanneer deze drie onderdelen goed geïmplementeerd zijn dan kan er makkelijker een nieuwe view toegevoegd of verwijderd worden zonder dat de applicatie daardoor niet meer 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.

## 5.2 MVC in Vossen en Konijnen

In de huidige applicatie moet er op minstens vijf plekken code gewijzigd worden voordat er een nieuwe view toegevoegd kan worden. Een nieuwe view zou bijvoorbeeld een cirkel- of staafdiagram kunnen zijn. Daarom is er gekozen om het MVC-model te implementeren als verbetering. Dit is gedaan door eerst een duidelijke packagestructuur te maken en daarna de bestaande klassen 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 packages (hanze > PIV1E > leertaak2) zijn om duidelijk te maken van welk bedrijf de code is en waar de code voor dient (zie figuur 5.1). 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 uitzondering van de main-package ondersteunen het model.



Figuur 5.1 - De packagestructuur van V&K

Na het aanbrengen van de packagestructuur kon deze gevuld worden. Veel klassen bestonden uit delen die zowel de GUI 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.

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);

Codefragment 5.1 uit Simulator.java

Als eerst wordt het model gemaakt, simulation in dit geval. Dit model voert alle berekeningen uit en maakt houdt bij wat er in elk hokje van de simulatie zit. Daarom word er aan dit model meegegeven uit hoeveel hokjes het moet bestaan.

Hierna wordt de belangrijkste view aangemaakt. View laat de informatie die simulation heeft zichtbaar voor de gebruiker. Om deze informatie op te halen moet de view weten vanuit welk model deze informatie komt. Daarom krijgt view als laatste parameter de parameter simulation mee. Daarna worden er drie views gemaakt die alle informatie op een overzichtelijke manier weergeven. Elke view heeft een afmeting nodig en moet weten van het model waar de view zijn informatie moet weghalen. Die drie views zijn: graph, pie en chart. Tot slot worden er nog 2 controllers aangemaakt.

Deze controllers hoeven alleen te weten waar ze de input van de gebruiker naar toe moeten sturen en krijgen daarom ook de parameter 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 stukje code fragment hieronder laat deze constructor zien.

public AbstractView(AbstractModel simulation) {

this.simulation = simulation;

simulation.addView(this);

}

Codefragment 5.2 uit AbstractView.java

Elk model is weer een subclasse van AbstractModel en hierin staan o.a. de methoden voor het toevoegen en notificeren van de views. Het stukje code fragment hieronder laat deze methodes zien.

public void addView(AbstractView view) {

views.add(view);

}

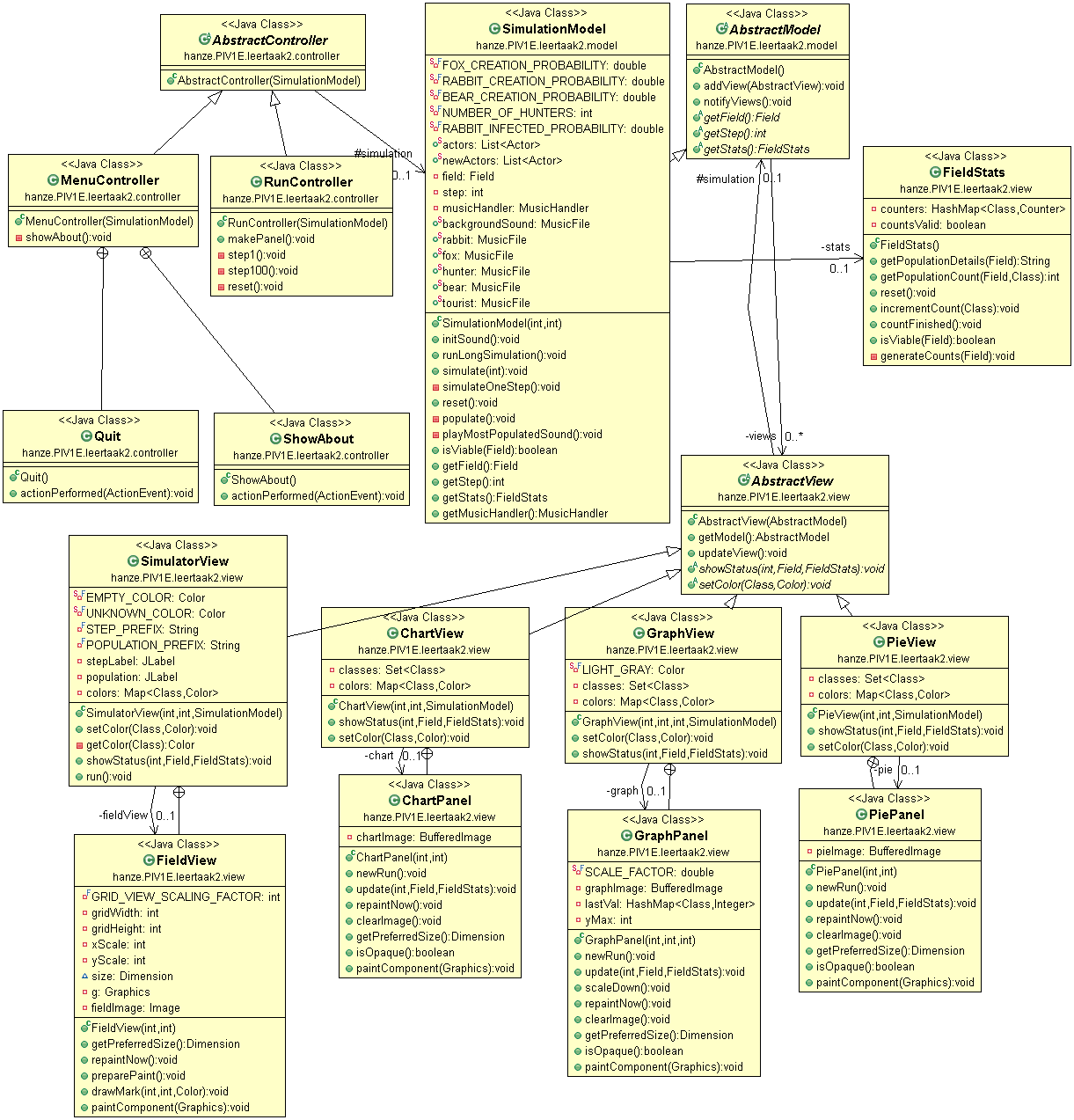
public void notifyViews() {

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

}

Codefragment 5.3 uit AbstractModel.java

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

Hieronder ziet u nog een klassendiagram na het toepassen van het MVC model.  
  


Figuur 5.2 - Klassendiagram van de applicatie na het toepassen het MVC model.

# 6. Uitbreiding 2

Nu de structuur van de applicatie is verbeterd dankzij het MVC model kunnen de laatste paar gewenste uitbreidingen geïmplementeerd worden. In dit hoofdstuk gaan we die uitbreidingen een voor een bij langs. Er wordt verteld over de uitbreiding van de GUI met een parameter paneel, we bespreken de voedselvoorraad van de konijnen en laten zien hoe geluiden en plaatjes geïntegreerd zijn in de applicatie. Ook introduceren we de toerist in de simulatie die voor veel chaos zorgt.

## 6.1 Het parameter paneel

## 6.2 Voedselvoorraad van de konijnen

De huidige versie van het programma houdt geen rekening met het feit dat konijnen ook moeten eten. Als er veel konijnen zijn op een veld is er natuurlijk ook veel minder eten voor een konijn beschikbaar.

Om er voor te zorgen dat ook hier rekening mee gehouden wordt, hebben we een methode ingevoerd die het aantal konijnen kan tellen die op dat moment in het veld leven. Vervolgens wordt er op basis van dat aantal konijnen bepaald of een konijn ook meer of minder nakomelingen krijgt. Hiervoor is de volgende methode geschreven:

Codefragment 6.1 uit Rabbit.java

**private** **int** getLitterSize()

{

**if**(getCount() >= *TO\_MUCH\_RABBITS*){

**return** *MAX\_LITTER\_SIZE* / 2;

}

**return** *MAX\_LITTER\_SIZE*;

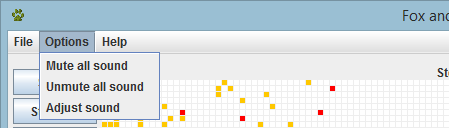
}

De gebruiker kan bepalen wanneer er te veel konijnen zijn en het aantal nakomelingen worden gehalveerd wanneer dat aantal is bereikt.

## 6.3 Geluiden en plaatjes

Om de gebruiker van de applicatie meer feedback te geven zijn ook geluiden en plaatjes toegevoegd.   
Elk diersoort en menssoort heeft zijn eigen geluid die getriggerd wordt wanneer ze de hoogste populatie hebben. Hierdoor kan de gebruiker na het uitvoeren van een stap of een aantal stappen gelijk horen welk dier of soort mens de hoogste populatie heeft. Daarnaast is er een wind geluid dat als achtergrond geluid van de simulatie fungeert.

Het regelen van het geluidsniveau van de geluiden kan ook gedaan worden op een apart geluidspaneel. Het paneel en de opties om het geluid uit en aan te zetten zijn verwerkt in het menu “Options” (zie figuur 6.1).



Figuur 6.1 - Screenshot van de geluidsopties in de verbeterde V&K applicatie

Wanneer de optie “Adjust sound” gebruikt wordt krijgt de gebruiker een schuifpaneel te zien waarin het geluid geregeld kan worden (zie figuur 8). Het volume van elk geluid kan apart ingesteld worden naar de wens van de gebruiker. Daarnaast is er ook nog een reset knop die alle instellingen terug zet naar beginwaardes.

## 6.4 De toerist

De opdrachtgever wou ook graag dat er nog een extra zinvolle en originele aanvulling aan de simulatie word toegevoegd. Hiervoor is de toerist bedacht. Deze toerist zet met zijn onkunde het leven van de dieren in het bos op de kop door het veroorzaken van een bosbrand.

# 7. Bonus uitbreidingen

Naast alle verplichte uitbreidingen zijn er ook nog een tweetal bonus uitbreidingen aan de simulatie toegevoegd. Zo is het bijvoorbeeld mogelijk dat een konijn ziek kan worden en die ziekte kan zich vervolgens verspreiden onder de andere dieren. Verder wil de opdrachtgever graag dat we de simulatie vanaf de commandline kunnen opstarten.

## 7.1 Konijnenziekte

Er bestaat bij het aanmaken van de simulatie een kleine kans van 1% dat een konijn geïnfecteerd is met de konijnenziekte. Deze ziekte is zeer besmettelijk voor andere konijnen en heeft dodelijke gevolgen. Zodra een konijn ziek word heeft het nog maar 5 stappen te leven. Daarnaast hebben de naburige konijnen 90% kans dat ze ook besmet worden met deze ziekte. Andere dieren kunnen ook beïnvloed worden door de ziekte.

Een vos kan de ziekte krijgen wanneer het een ziek konijn op heeft opgegeten. Doordat de vos veel groter is dan een konijn heeft de ziekte meer vrijheid en wordt de ziekte veel krachtiger. Gelukkig heeft de ziekte voor de vos geen nadelige gevolgen en wordt de vos simpel weg alleen een drager van de ziekte.

Een beer daarentegen is wel weer gevoelig voor de ziekte. Zodra een beer een vos heeft opgegeten die geïnfecteerd is gaat hij sneller dood dan normaal. De ziekte werkt op zijn organen en zorgt er voor dat de beer 20 stappen ouder wordt. Hij gaat dus sneller door het eten van zieke vossen. Mensen hebben verder geen last van de ziekte.

De ziekte wordt in de code aangegeven door de protected boolean infected in de klasse Animal die standaard op false staat. Elke subklasse van Animal heeft weer een eigen methode die deze boolean naar true zet en de gevolgen van de infectie activeerd.

Codefragment 7.1 uit Rabbit.java

**public** **void** setInfection(**boolean** infected) {

**if**(infected){

age = *MAX\_AGE* - 5;

}

**this**.infected = infected;

}

Deze methode wordt weer getriggerd door een andere methode genaamd “checkAndInfect”. Deze methode maakt onderdeel uit van de “act” methode en wordt daardoor bij elke stap uitgevoerd. In codefragment 7.2 ziet u hoe deze methode er uit ziet.

Codefragment 7.2 uit Rabbit.java

**public** **void** checkAndInfect()

{

Field field = getField();

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

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

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

Location where = it.next();

Object animal = field.getObjectAt(where);

// Check if one of the adjacent locations has a rabbit

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

Rabbit rabbit = (Rabbit) animal;

// Infect rabbit if other rabbit is infected

**if**(rabbit.checkForInfection()) {

**if**(infected == **false**){

setInfection(***rand***.nextDouble() <= ***INFECTION\_CHANCE***);

}

}

}

}

}

De methode “checkAndInfect” kijkt of er in de naburige vakjes ook konijnen aanwezig zijn en vervolgens wordt er gekeken of deze ook ziek zijn. Dit wordt weer gedaan door een andere methode genaamd “checkForInfection” die in de klasse Animal zit.

Codefragment 7.3 uit Animal.java

**public** **boolean** checkForInfection()

{

**return** infected;

}

Deze “checkForInfection” methode geeft de waarde van de boolean infected terug en als de naburige konijn ziek is dan kan deze konijn ook ziek worden. Elk dier heeft een eigen kans percentage (*INFECTION\_CHANCE*) die bepaald of het dier ook daadwerkelijk ziek wordt of niet.

Om er voor te zorgen dat er zieke konijnen zijn tijdens de simulatie is er een kans van 1% per konijn ingesteld dat ze in het veld geplaatst worden met infected op true.

## 7.2 Opstarten vanaf de commandline

# 8. Conclusie en aanbevelingen

# Literatuurlijst

Programmeren in Java met BlueJ – editie: 5 - Auteurs: David J.Barnes & Michael Kölling > check de Schrijfwijzer Techniek voor de notatie van de bronnen.

# Bronnen

Bijlage 1

Bijlage 2