De jachtsimulatie

Auteurs:

Rob Dambrink

Mark Jacobs

Nick Klein

Robert Stocker

Inhoudsopgave

[Hoofdstuk 1 - Inleiding 2](#_Toc379210382)

[Probleemstelling 2](#_Toc379210383)

[Analyse van de huidige situatie 2](#_Toc379210384)

[Hoofdstuk 2 - Uitbreiding 1 3](#_Toc379210385)

[Voortplanten 3](#_Toc379210386)

[Interface Actor en superklasse Animal 3](#_Toc379210387)

[Jager (Hunter) 3](#_Toc379210388)

[Alligator 3](#_Toc379210389)

[Hoofdstuk 3 - Het gebruik van MVC in V&K 4](#_Toc379210390)

[Verschillen tussen DMT en DMTG 4](#_Toc379210391)

[Verschillen tussen DMTG en life 4](#_Toc379210392)

[De life MVC-structuur in V&K 4](#_Toc379210393)

[Hoofdstuk 4 - Uitbreiding 2 5](#_Toc379210394)

[Gras 5](#_Toc379210395)

[Nuke 5](#_Toc379210396)

[Zombie Rabbits 5](#_Toc379210397)

[Hoofdstuk 5 - Beschrijving uiteindelijke simulatie 6](#_Toc379210398)

[Belangrijkste toegevoegde features 6](#_Toc379210399)

[Testproces 6](#_Toc379210400)

[Bekende problemen 6](#_Toc379210401)

[Refactoren van V&K 7](#_Toc379210402)

[Hoofdstuk 6 - Conclusie 8](#_Toc379210403)

[Hoofdstuk 7 - Samenvatting 9](#_Toc379210404)

[Hoofdstuk 8 - Bijlagen 10](#_Toc379210405)

[Individuele bijdrage 10](#_Toc379210406)

[Reflectie Nick Klein 10](#_Toc379210407)

[Reflectie Robert Stocker 10](#_Toc379210408)

[Reflectie Mark Jacobs 11](#_Toc379210409)

[Reflectie Rob Dambrink 11](#_Toc379210410)

[Klassediagram bij hoofdstuk 2 (lesweek 8) 12](#_Toc379210411)

[Klassediagram bij hoofdstuk 4 (lesweek 9) 13](#_Toc379210412)

[Sequentiediagram hoofdstuk 4 (lesweek 9) 14](#_Toc379210413)

# Hoofdstuk 1 - Inleiding

De aangeleverde versie van vossen en konijnen was een simpele simulatie. Konijnen en vossen worden in een veld geplaatst en proberen beide te overleven. De opdracht hield in dat deze simulatie uitgebreid moest worden.

Dit verslag is gemaakt om als handleiding voor de nieuwe, aangepaste simulatie te dienen. De aanpassingen aan de simulatie worden uitgelegd en aan de hand van diagrammen en fragmenten code geïllustreerd.

De hoofdstukken 2 en 4 bevatten de uitleg voor de aanpassingen die in respectievelijk lesweek 8 en 9 zijn gemaakt. De bijbehorende diagrammen zijn als bijlagen toegevoegd. Hoofdstuk 3 is gewijd aan het MVC-model en hoe dit effectief op de simulatie kan worden toegepast.

## Probleemstelling

Er is een gebrek aan inzicht in de mate waarin externe invloeden effect hebben op een ecosysteem. Het doel van deze applicatie is om, door dergelijke situaties te simuleren, meer inzicht op dit vlak te verkrijgen.

## Analyse van de huidige situatie

De aangeleverde code heeft een aantal belangrijke gebreken. Zo is er geen mogelijkheid voor de gebruiker om input aan de applicatie te geven. Ook levert de simulatie constant hetzelfde resultaat. Dit komt doordat de random waardes allemaal met een van tevoren ingestelde seed werken. Dit is makkelijk te verhelpen door de seed leeg te laten of te randomizen.

De resultaten van de simulatie zijn zeker vergelijkbaar aan de varkenscyclus. In de varkenscyclus zijn vraag en aanbod beide een sinus. Wanneer het aanbod hoog is, daalt de vraag en andersom. In de simulatie zullen konijnen meer voortplanten wanneer de vossen in kleinere aantallen voorkomen. Meer konijnen betekend meer eten voor de vossen waardoor de vossenpopulatie kan groeien. De vossen zullen langzaam sterven van de honger door gebrek aan voedselbronnen (konijnen) wat er voor zorgt dat deze weer in grotere getalen kunnen voorkomen.

Het valt op dat de vossen konijnen vaak omsingelen en zo de groep doodmaken. Konijnen die uit de groep weten te komen rennen weg bij de vossen en breiden de konijnenpopulatie weer uit.

De aangeleverde code is zeer goed uit te breiden. Veel van de bestaande methodes lenen zich uitstekend voor gebruik in nieuwe methodes. Ook is het aanmaken van nieuwe diersoorten niet moeilijk. De code voor jagende dieren lijkt allemaal erg veel op elkaar, enkel de balans tussen leeftijd, breeding probability en food values moet gevonden worden.

# Hoofdstuk 2 - Uitbreiding 1

In dit hoofdstuk worden de aanpassingen die in lesweek 8 zijn gemaakt, uitgelegd. Een klassediagram met een overzicht van de klassen na al deze aanpassingen is als bijlage toegevoegd.

## Voortplanten

Het voortplanten van de dieren is aangepast om het realistischer te maken. Voorheen hadden de dieren geen geslacht en konden zij op zichzelf voortplanten zonder daarbij een partner nodig te hebben. Bij het aanmaken van bijvoorbeeld een konijn wordt nu een geslacht meegegeven.

**if**(**this**.getClass().equals(animals.get(a).getClass())) {

**if**(**this**.getSex()!=animals.get(a).getSex()) **return** **true**; }

Codefragment 1: Check voor de diersoort en het geslacht van mogelijke partners

## Interface Actor en superklasse Animal

De interface Actor en superklasse Animal zijn aangemaakt. De klasse Animal implementeerd de interface Actor. Actor bevat abstracte methoden die door elk dier-object in het veld gebruikt moeten worden. Enkele voorbeelden van deze methoden zijn: setDead(), een methode die het dier 'dood' maakt door het object uit het veld te halen en de methode chooseSex(), een methode om het geslacht van een dier te bepalen. De klasse Animal vult al deze methoden in zodat zij ook daadwerkelijk gebruikt kunnen worden. Iedere diersoort in de simulatie is een subklasse van Animal.

## Jager (Hunter)

De hunter is een speciaal roofdier. Door de toevoeging van de hunter is de vos niet alleen een roofdier, maar ook een prooi. De hunter heeft een unieke methode genaamd closestPrey() waarmee de dichtsbijzijnde prooien gezocht worden. Wanneer er geen prooi in de buurt is geeft deze methode een willekeurig vrije locatie bij de hunter terug. closestPrey() heeft een bereik van 2 vakjes, groter dan alle andere roofdieren om het wapen van de hunter te simuleren. Naast closestPrey() heeft de hunter nog een unieke methode: closestMate(). Met deze methode wordt uit alle compatible actors gekeken welke partner het dichtste bij de hunter is. Dit wordt berekend door het verschil in de kolommen en rijen van het veld te vergelijken. Het kleinste verschil staat voor de dichtsbijzijnde partner.

**for**(**int** a = 0; a < all.size(); a++) {

**int** comparable = Math.*abs*(**this**.getLocation().getRow()-all.get(a).getRow()) + Math.*abs*(**this**.getLocation().getCol()-all.get(a).getCol());

**if**(comparable<closest) {

closest = comparable;

mate = all.get(a);

}

Codefragment 2: Berekening voor het verschil tussen de kolom en rij voor hunter en partner

## Alligator

De alligator is toegevoegd als nieuw roofdier in de simulatie. Oorspronkelijk moest de alligator in de buurt van water zijn, maar water is uiteindelijk niet doorgevoerd. De alligator kan zich nu overal in het veld verplaatsen. De alligator staat bovenaan in de voedselketen. Rabbits, Foxes en Hunters zijn allemaal een prooi van de alligator terwijl geen enkel ander dier op de alligator jaagt. Omdat een grote voedselvoorraad en een gebrek aan vijanden de alligator een groot voordeel geven, is de creation probability het laagst van alle dieren.

# Hoofdstuk 3 - Het gebruik van MVC in V&K

## Verschillen tussen DMT en DMTG

Het project MVCDynamicModelThread (hierna genoemd als DMT) verschilt op twee manieren van

MVCDynamicModelThreadGeneralized (hierna genoemd als DMTG). Zo is DMTG opgedeeld

in verschillende packages met allemaal een eigen doel. De package namen en functies komen

overeen met de verschillende klassen die in DMT staan. DMT heeft alle klassen echter in 1 package

staan. Verder maakt DMTG gebruik van abstracte methoden. Iets wat in DMT niet voorkomt.

## Verschillen tussen DMTG en life

DMTG lijkt erg veel op life. Een duidelijk verschil is wel dat life veel gebruik maakt van exceptions,

waar deze in DMTG niet gebruikt worden.

## De life MVC-structuur in V&K

Life gebruikt de volgende structuur: Controller, exception, logic, main, runner en view.

In vossen en konijnen is deze structuur grotendeels overgenomen. Er zijn packages gemaakt

voor alle onderdelen en deze zijn gevuld met de bijbehorende java onderdelen. Zo is er een duidelijk

overzicht van wat alles doet in de simulatie en is het makkelijker om aanpassingen te maken. Ook is

het makkelijker voor een derde partij om het project te bekijken.

Een verschil tussen het life project en het V&K project is het gebruik van exceptions. Deze worden in

life veel gebruikt, maar komen in V&K niet aan bod. Ook zijn de namen voor controller, logic en

runner naar view, simulation en field veranderd. View is opgedeeld in meerdere packages, die samen

de onderdelen voor de complete view bevatten.

# Hoofdstuk 4 - Uitbreiding 2

In dit hoofdstuk worden de aanpassingen die in lesweek 9 zijn gemaakt, uitgelegd. Een klassediagram met een overzicht van de klassen na al deze aanpassingen is als bijlage toegevoegd.

## Gras

Gras is het eerste object in het veld dat geen dier is. Gras is toegevoegd als voedselbron voor konijnen. Gras jaagt niet, beweegt niet en heeft zelf geen voeding nodig. De roofdieren in de simulatie hebben geen baat bij het gras en zullen het kapot trappen wanneer zij er mee in aanraking komen.

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

Grass grass = (Grass) animal;

**if**(grass.isAlive()){

grass.setDead();

**return** where;

}

Codefragment 3: Wanneer een dier, dat geen konijn is, met gras in aanraking komt vergaat het gras.

## Nuke

Om de simulatie te herstarten is een nucleaire bom toegevoegd. De knop, voorzien van een afbeelding van een nucleair symbool, zal het gehele veld legen en vervolgens een nieuw veld vol dieren en gras genereren. De simulatie zal opnieuw op stap 0 staan. De gebruiker kan vervolgens de nieuwe simulatie starten door op 'Start' of 'One Step' te klikken.

## Zombie Rabbits

Konijnen hebben elke stap een kans van 1 op 1.000 om geïnfecteerd te raken met een dodelijk zombie virus. Geïnfecteerde konijnen sterven direct en worden vervangen door zombie konijnen. De zombies lijken op het eerste gezicht zwak. Ze kunnen slechts 4 jaar oud worden en hebben een kans van 20% om te lopen. Wanneer zombies bij normale konijnen in de buurt komen is er een kans van 40% dat dit konijn ook geïnfecteerd raakt. Zombies hebben geen eten nodig en zullen dus altijd (opnieuw) sterven na 4 stappen.

**private** **static** **final** **int** *INFECTION\_CHANCE* = 10000;

**private** **void** infectionChance(List<Actor>newRabbits) {

Field field = getField();

**int** random = *rand*.nextInt(*INFECTION\_CHANCE*+1);

**if**(random==*INFECTION\_CHANCE*) {

Location location = getLocation();

setDead();

ZombieRabbit rabbit = **new** ZombieRabbit(**true**, field, location);

newRabbits.add(rabbit); }

}

Codefragment 4: De berekening waarmee bepaald wordt of een konijnen geïnfecteerd raakt of niet.

# Hoofdstuk 5 - Beschrijving uiteindelijke simulatie

In dit hoofdstuk worden verschillende processen en andere aspecten van de simulatie, die nog niet eerder in het verslag aan bod zijn gekomen, beschreven. Een klassediagram van de uiteindelijke situatie en een compleet overzicht met alle javadoc van de simulatie zijn als bijlage toegevoegd.

## Belangrijkste toegevoegde features

De twee belangrijkste toevoegingen aan de simulatie zijn een klasse en een methode. De klasse gras is interessant omdat gras er voor zorgt dat ook de konijnen eten moeten hebben om te overleven. Voorheen was het enkel voortplanten nodig om de konijnenpopulatie groot te houden. Het gras voegt meer dynamiek toe aan de simulatie en zorgt voor meer gevarieerde uitkomsten.

De methode closestPrey() van jagers is een unieke en handige methode. De methode zoekt het veld af naar een prooi die binnen of vlak binnen het bereik van de jager staat. De jager zal zich naar zijn prooi toe bewegen. Dit simuleert de intelligentie van de menselijke jagers.

## Testproces

De simulatie is altijd in zijn geheel getest. Er is bij het testen zowel van de black-box en de white-box test methoden gebruik gemaakt. De schrijver van de code test of de code goed doorlopen wordt. De andere groepsleden testen de code en proberen deze te breken voordat zij de code zelf hebben bekeken. Door de code te testen voordat deze doorgenomen is, wordt een gebruikerservaring gecreëerd. Door het gebruik van de white-box methode is meer inzicht verkegen in hoe de code doorlopen wordt waardoor fouten makkelijker gevonden en opgelost kunnen worden.

## Bekende problemen

De simulatie heeft op het moment een aantal gebreken. Handige methoden zijn niet aan elk dier toegevoegd zo hebben rabbits de avoidZombies() methode om weg te rennen van ZombieRabbits. Een soortgelijke methode zou natuurlijk handig zijn voor andere prooidieren om weg te rennen van de dieren die op hen jagen. Verder is alleen de hunter altijd op zoek naar partners. Andere dieren zoeken elkaar niet in het veld, maar komen toevallig langs elkaar. Als laatste is gras voor sommige dieren een blokkade. Niet elk dier eet het gras of trapt het kapot.

Het settings menu kan meer opties bevatten. Ook verplaatst het veld zich tijdens de simulatie constant wanneer een andere grote voor het veld wordt ingesteld.

De diagrammen die informatie over de simulatie moesten leveren zijn helaas niet op tijd afgerond.

Tot slot is er nog een probleem met de snelheid van de simulatie. Als de simulatie wordt afgespeeld terwijl er hunters in het veld staan, kan de simulatie soms een stuk langzamer lopen. Dit komt omdat hunters het gehele veld controleren voor mogelijke partners.

## Refactoren van V&K

Bij de implementatie van vossen en konijnen is er op gelet dat elke methode en klasse een zo specifiek mogelijk doel heeft om ervoor te zorgen dat zij niet afhankelijk van elkaar zijn. Dit is meegenomen bij het ontwerpen van het project.

Refactoren is eigenlijk niet gedaan omdat vanaf het begin van het project er voor werd gezorgd dat alles op de goede plaats stond. De MVC structuur heeft hierbij ook geholpen omdat die ook van het begin af aan is gevolgd. Er is verder ook op de cohesie en inkapseling gelet, want alles dat in een package staat werkt goed met elkaar samen om de functie uit te voeren.

# Hoofdstuk 6 - Conclusie

Door het vroeg toevoegen van de MVC-structuur aan V&K is het niet moeilijk om nieuwe toevoegingen of aanpassingen binnen dit model door te voeren. Ook is het toevoegen van nieuwe diersoorten erg snel en eenvoudig geworden door de superklasse Animal. Het grootste probleem bij het toevoegen van nieuwe diersoorten is het vinden van de balans met de andere dieren in het veld.

Sommige functies van de simulator zijn helaas nog onvoldoende geïmplementeerd om maximaal gebruikt te kunnen worden. De basis van de simulatie is echter goed afgerond en werkt zoals deze voorgesteld was aan het begin van het project.

# Hoofdstuk 7 - Samenvatting

De methode mate() is realistischer uitgewerkt. Dieren hebben een partner van het tegenovergestelde geslacht nodig om voort te planten.

ZombieRabbit toegevoegd. Rabbits hebben elke beurt een kans van 1 op 1.000 om in ZombieRabbits te veranderen en ZombieRabbits hebben een kans van 40% om normale Rabbits te infecteren.

Rabbits hebben de avoidZombies() methode om van zombies te vluchten.

Gras is toegevoegd als voedselbron voor de Rabbits. Alligators trappen het gras kapot en andere diersoorten zullen om het gras heen lopen.

Alligators zijn toegevoegd. Alligators jagen op Hunters, Rabbits en Foxes. Alligators vertrappen gras wanneer zij daarmee in aanraking komen.

Hunter zijn toegevoegd. Hunters jagen op Rabbits en Foxes en hebben een aanvals bereik van 2 vakjes in plaats van de standaard van 1 vakje. Hunters hebben twee unieke methoden: ClosestPrey() waarmee de positie van de dichtsbijzijnde prooi berekend wordt en closestMate() waarmee de positie van de dichtsbijzijnde partner berekend wordt.

Interface Actor met abstrace methoden voor elke diersoort toegevoegd.

Superklasse Animal implementeerd de interface Actor en is superklasse voor elke diersoort.

De package vk.model bevat de code om het veld en de locaties in het veld in de simulatie te maken.

De package vk.view bevat de code die gebruikt wordt om de resultaten van de simulatie op het scherm te laten zien.

De package vk.simulation bevat de code voor de simulatie zelf. Ook de randomizer klasse staat in deze package.

De package vk.view.button bevat de code voor de knoppen die in de simulatie door de gebruiker gebruikt worden. Ook de afbeelding op de reset(nuke) knop staat in deze package.

De packages vk.view.menu en vk.view.menu.item bevatten de code voor het menubalkje dat bovenaan de simulatie staat.

De package vk.view.textfields bevat de code voor de menu optie die gebruikt wordt voor het instellen van de width en depth van het veld.

De package vk.view.windows bevat de code voor het frame om de simulatie.

# Hoofdstuk 8 - Bijlagen

## Individuele bijdrage

Rob heeft het verslag geschreven en de klassediagrammen gegenereerd. Voor de simulatie heeft Rob de klasse alligator toegevoegd en samen met Mark aan de methode closestPrey() gewerkt. Daarnaast heeft Rob meerdere tests uitgevoerd.

Mark heeft de uitgewerkte voortplantingsmethode geschreven. Verder heeft Mark meegewerkt aan de uiteindelijke implementatie van de zombieRabbits en ook de avoidZombies() methode toegevoegd aan normale rabbits. Tot slot heeft Mark de opties voor field width en height in het settings menu gemaakt en threads geimplementeerd in de simulatie.

Nick heeft de originele GUI gemaakt. Verder heeft Nick de MVC-structuur toegepast op V&K, de interface actor aangemaakt en de zombieRabbits toegevoegd.

Robert heeft Grass en Hunter klassen aangemaakt en de menubalk bovenin de simulatie toegevoegd. Tot slot heeft Robert de image bij voor de nuke geimplementeerd.

Alle groepsleden hebben elkaar geholpen door naar problemen te kijken en mee te denken over oplossingen. Kleinere aanpassingen die gemaakt zijn aan bestaande code zijn niet meegenomen in de bijdrages van de groepsleden.

## Reflectie Nick Klein

Het project zelf vond ik wel te doen, maar wel slecht geregeld. Er was geen duidelijkheid over inleverdata en allerlei andere details.

De samenwerking in ons groepje ging in het begin wel goed, maar naarmate de deadline dichterbij kwam werd deze steeds minder. Er waren woordenwisselingen, maar uiteindelijk is het op z'n pootjes terecht gekomen.

Het eindresultaat vind ik goed gelukt voor een product van ruwweg 2 week werken. Uiteraard had het wat meer opgepoetst kunnen worden, maar daar was gewoon geen tijd meer voor.

Van dit project heb ik veel geleerd. Communicatie binnen een projectgroep is essentieel, anders loopt alles langs elkaar heen en krijg je misverstanden.

## Reflectie Robert Stocker

**Algemene indruk:** Het project vond ik in het begin niet super interessant. Dat komt misschien ook wel omdat Java mij toen nog niet super aansprak zoals PHP dat wel deed. Ik vond dat het project thema wel grappig en je kon je fantasie er in kwijt, wat positief is.

**Wat vind ik leuk:** Wat ik leuk vond was het oplossen van de problemen die in het project voor kwamen. Dit project heeft mij meer enthousiast gemaakt voor Java opdrachten.

**Wat vond ik minder leuk:** Dat sommige onderwerpen zoals een cirkeldiagram maken lastig bleken te zijn. Zeker omdat er bijna geen informatie stond op blackboard over de onderdelen die moesten worden geïmplementeerd. Hier door hebben we enorm veel tijd verloren. Qua teamwork ontstond er op het einde wrijving doordat het project werd uitgesteld.

**Wat neem ik mee:** Eerder een agenda maken. De punten die moeten worden behaalt in de groep, goed noteren en meer onderzoek doen naar onderdelen die moet worden geïmplementeerd.

## Reflectie Mark Jacobs

Het project was voor mij wel een leuke ervaring. Het was alleen jammer dat wij alle vier wat chaotisch zijn waardoor bepaalde dingen uiteindelijk of niet goed of helemaal niet af kwamen. Ik had nog veel dingen willen toevoegen, maar daar was jammer genoeg de tijd niet voor. Mijn groepspartners waren prima, ik was niet de enige die bijvoorbeeld goed kon programmeren. Die ervaring heb ik eerder wel gehad.

Wat ik wel jammer vind is dat dit een “uitbreiding” project was en dat we op iemand anders zijn code moesten uitbreiden. Dat is soms veel lastiger omdat je niet precies weet wat diegene heeft gemaakt. Als het project bestond uit het compleet maken van een simulatie was het misschien wel leuker en beter geweest. Dan weet je precies waar je code voor dient en is het ook makkelijker te overzien.

Ik heb best wel veel geleerd van het project. Nu weet ik hoe ik threads moet maken en starten, hoe ik goed met MVC moet werken, mijn kennis over alle verschillende collecties(LinkedList, HashMap etc) is versterkt, ik heb kennis opgedaan over GUI maar voornamelijk ben ik gewoon veel bekender geworden met de taal java.

## Reflectie Rob Dambrink

De opdracht was interessant. Doordat MVC moest worden toegepast en veel op het scherm weergegeven moest worden, moest er toch veel geleerd worden naast de eerder vergaarde kennis over java.

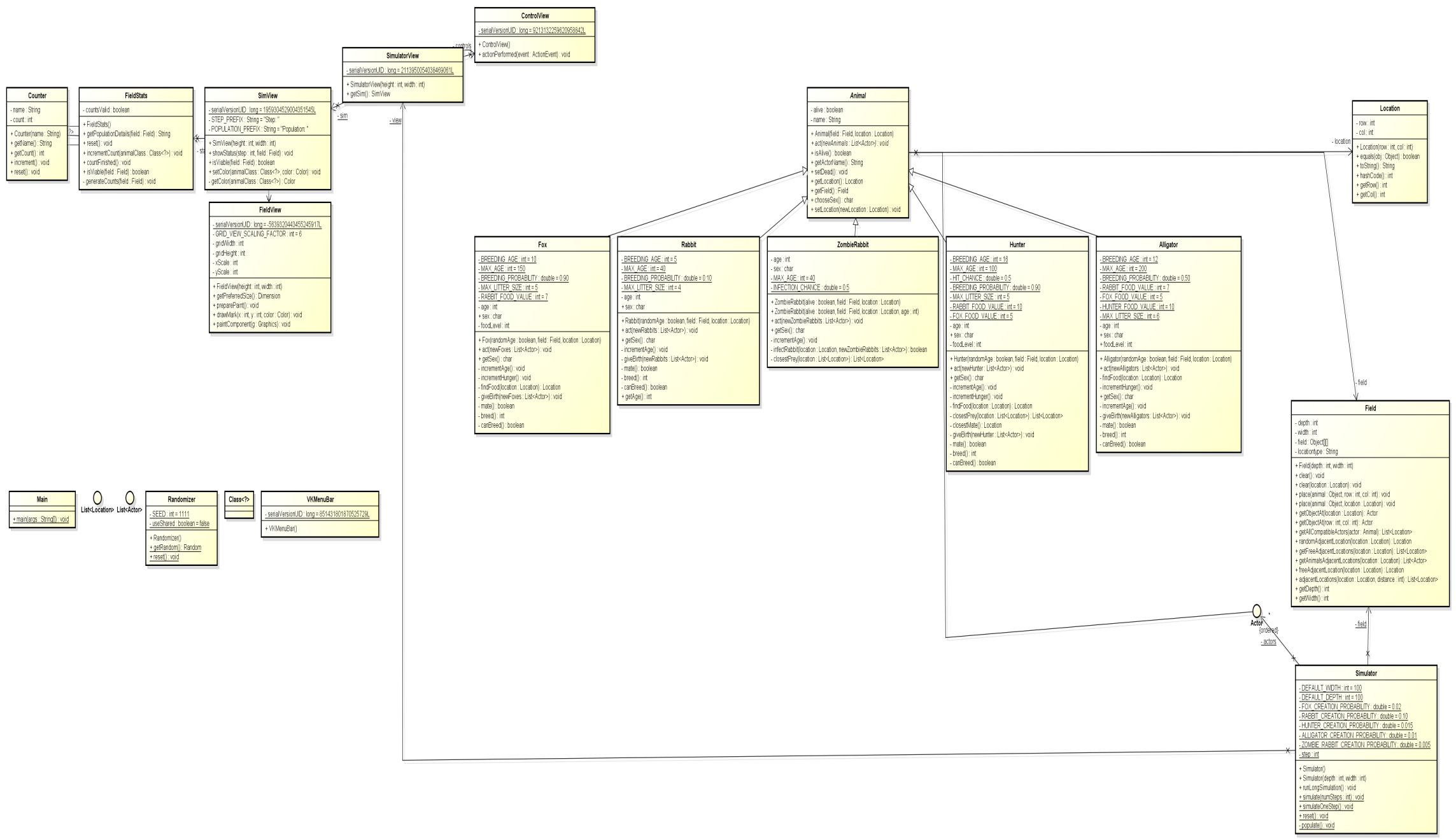
De projectgroep was gezellig en kon goed samenwerken. Al waren er op het laatst enige misverstanden over bepaalde delen van de simulatie en het verslag. Er was een duidelijke rolverdeling en daar heeft iedereen zich goed aan gehouden. Iedereen heeft elkaar geholpen en dingen getest. Al het werk dat is verricht is goed gecontrolleerd door anderen en waar nodig werd goed advies gegeven over eventuele verbeteringen.

De opgeleverde simulatie vind ik erg succesvol. De simulatie heeft verschillende uitkomsten en (op de vossen na) elk dier heeft een kans om als 'winnaar' uit de simulatie te komen. Alles werkt soepel en intuïtief. Het is interessant om te zien hoe de blokjes zich over het veld verplaatsen en op elkaar reageren.

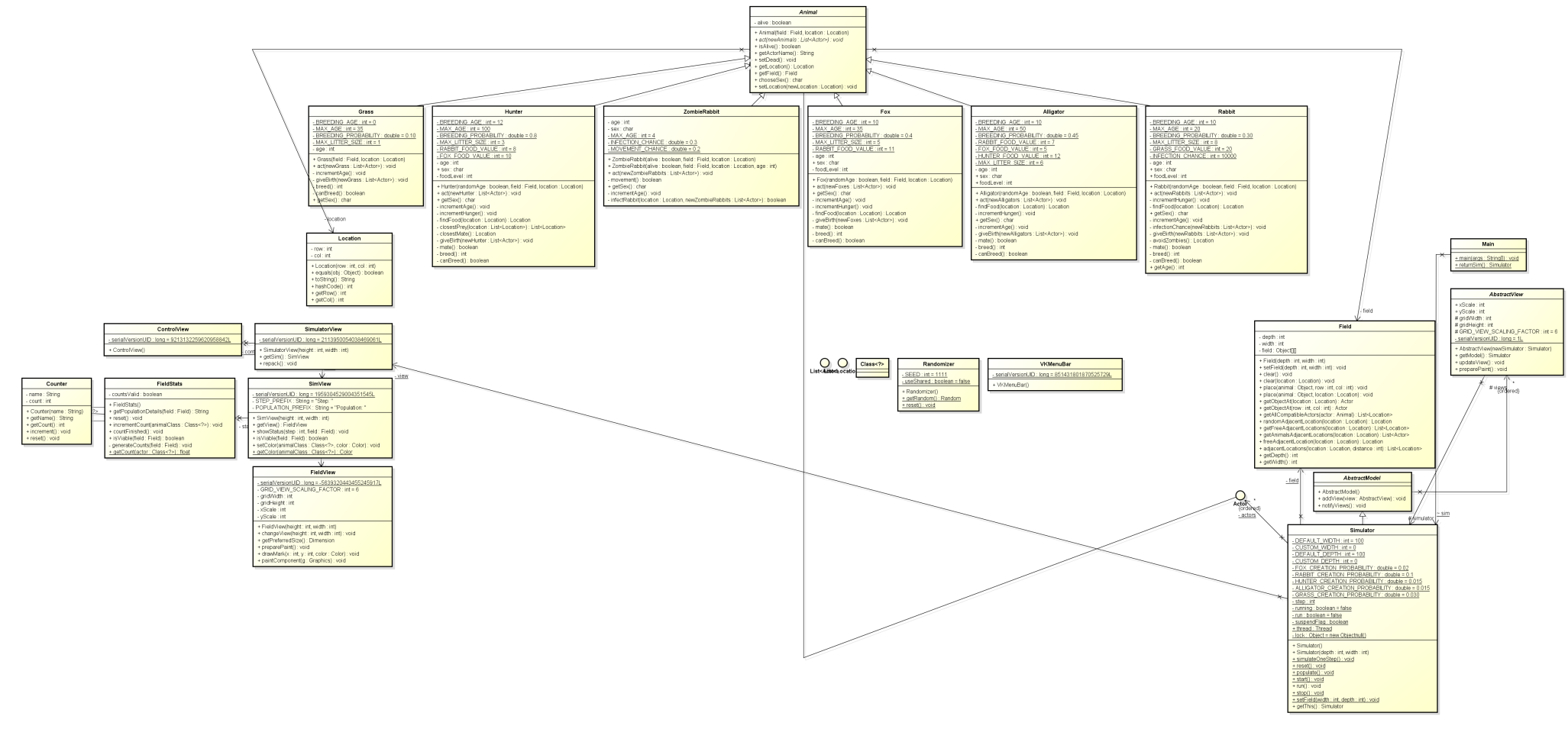
Het project heeft meer inzicht gegeven in het werken in groepsverband. Omdat alle groepsleden dezelfde interesses hadden en goed met elkaar op konden schieten hebben we het wat nauw genomen met de strenge afspraken. Uiteindelijk zorgde dit toch voor frustraties in latere stadia van het project.

Ik heb veel geleerd over de verschillende aspecten die in het project aan bod kwamen. Ook heb ik kunnen oefenen met het schrijven van een zakelijk verslag, waar ik verbazingwekkend veel plezier mee had.

## Klassediagram bij hoofdstuk 2 (lesweek 8)



## Klassediagram bij hoofdstuk 4 (lesweek 9)



## Sequentiediagram hoofdstuk 4 (lesweek 9)

