Auteurs:

Rob Dambrink

Mark Jacobs

Nick Klein

Robert Stocker

Inhoudsopgave

[Hoofdstuk 1 - Inleiding 2](#_Toc379160455)

[Probleemstelling 2](#_Toc379160456)

[Analyse van de huidige situatie 2](#_Toc379160457)

[Hoofdstuk 2 - Uitbreiding 1 4](#_Toc379160458)

[Voortplanten 4](#_Toc379160459)

[Interface Actor en superklasse Animal 4](#_Toc379160460)

[Jager (Hunter) 4](#_Toc379160461)

[Alligator 4](#_Toc379160462)

[Hoofdstuk 3 - Het gebruik van MVC in V&K 5](#_Toc379160463)

[Hoofdstuk 4 - Uitbreiding 2 6](#_Toc379160464)

[Gras 6](#_Toc379160465)

[Nuke 6](#_Toc379160466)

[Zombie Rabbits 6](#_Toc379160467)

[Conclusie 7](#_Toc379160468)

[Samenvatting 8](#_Toc379160469)

[Bijlagen 9](#_Toc379160470)

# 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

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.

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.

# 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 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.

# Conclusie

# Samenvatting

# Bijlagen