Auteurs:

Robert Stocker

Nick Klein

Mark Jacobs

Rob Dambrink

Inhoudsopgave

[Inleiding 2](file:///C:\Users\Rob\Documents\GitHub\VossenEnKonijnen\ProjectVossenenkonijnen.docx#_Toc378273031)

[Prognose 3](file:///C:\Users\Rob\Documents\GitHub\VossenEnKonijnen\ProjectVossenenkonijnen.docx#_Toc378273032)

[Conclusie 4](file:///C:\Users\Rob\Documents\GitHub\VossenEnKonijnen\ProjectVossenenkonijnen.docx#_Toc378273033)

[Samenvatting 5](file:///C:\Users\Rob\Documents\GitHub\VossenEnKonijnen\ProjectVossenenkonijnen.docx#_Toc378273034)

[Begrippenlijst 6](file:///C:\Users\Rob\Documents\GitHub\VossenEnKonijnen\ProjectVossenenkonijnen.docx#_Toc378273035)

# Inleiding

# 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.

# Beschrijving van de code week 8

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.

De volgende code wordt gebruikt om te bepalen wat de diersoort en het geslacht van een mogelijke partner is.

**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

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.

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

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.

Bijlage 1 bevat een klassediagram met de toevoegingen van deze weekopdrachten.

# Conclusie

# Samenvatting

# Begrippenlijst

Food value: De food value van een dier is het aantal stappen dat een jachtdier kan overleven zonder opnieuw te moeten eten.