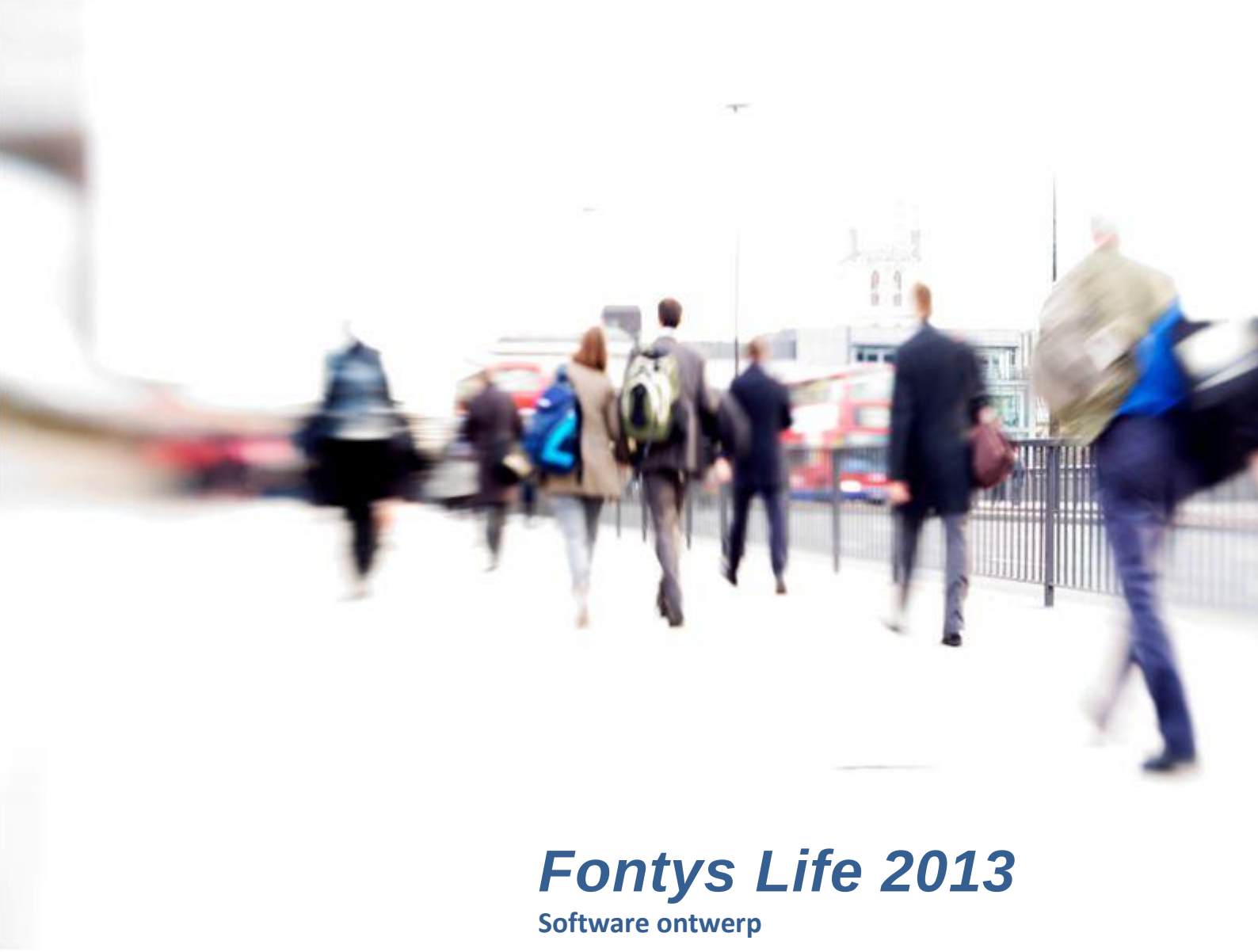


Fontys Life 2013

Software ontwerp

Auteurs:
Bart Janisse
Patrick van ieperen

Datum: 20-01-2014

Fontys Life 2013

Software ontwerp

Opdracht:	Business case simulatie
Begeleider:	Erik van der Schriek (docent)
Instelling:	Fontys hogeschool Eindhoven
Studie:	ICT & Technology
Schooljaar:	2013 - 2014
Datum:	20-01-2014
Student:	Bart Janisse / Patrick van ieperen
Studentnummer:	2213829 / 2221164
Correspondentie email:	b.janisse@fontys.student.nl p.vanieperen@fontys.student.nl

Inhoudsopgave

1 Wijzigingshistorie

Versie	datum	Wijziging	Auteur(s)
0.1	15-12-2013	Concept	Patrick / Bart
0.2	16-12-2013	Klasse Obstakel toegevoegd	Patrick / Bart
0.2	16-12-2013	Klasse plant toegevoegd	Patrick / Bart
1.0	20-01-2014	Ingeleverde versie	Patrick / Bart

2 Inleiding

2.1 doel

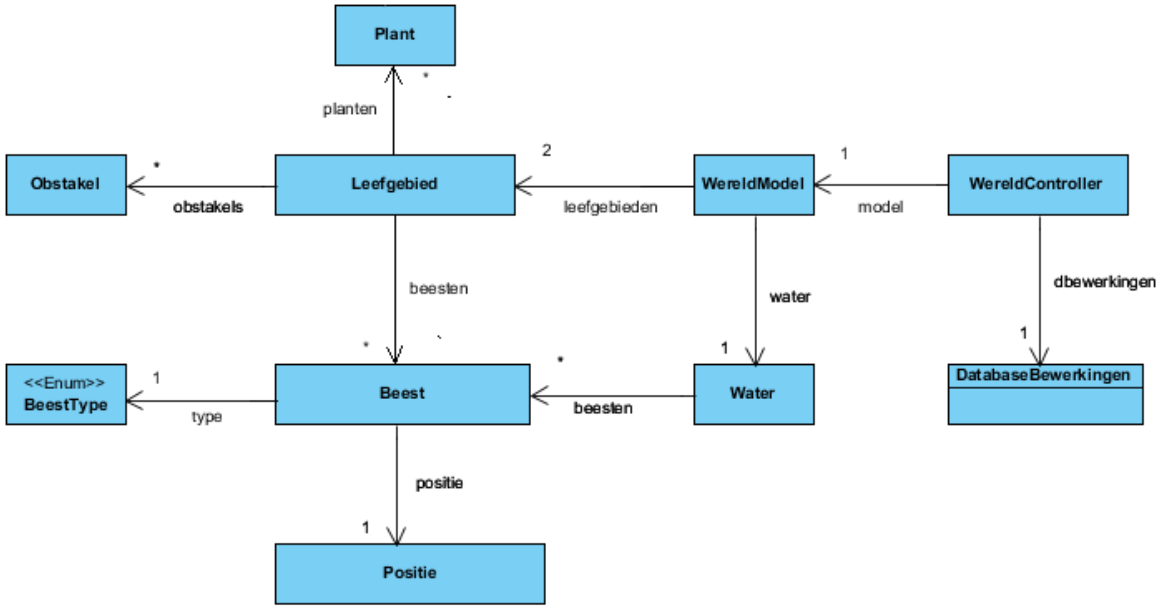
Dit document beschrijft in gedetailleerd het software ontwerp.

2.2 Definities en afkortingen

Afkorting	Betekenis

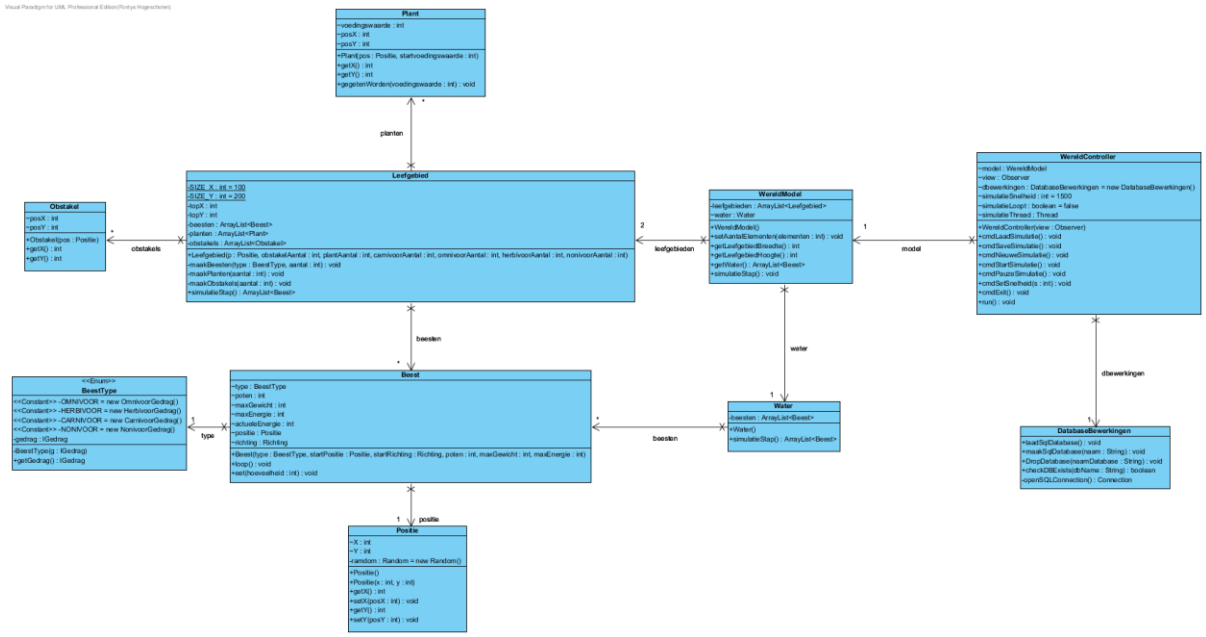
3 Algemene architectuur

Onderstaande afbeelding geeft de globale architectuur weer. Hierin is de onderlinge samenhang van de klassen te zien. Later zal per klasse een gedetailleerde beschrijving worden gegeven.

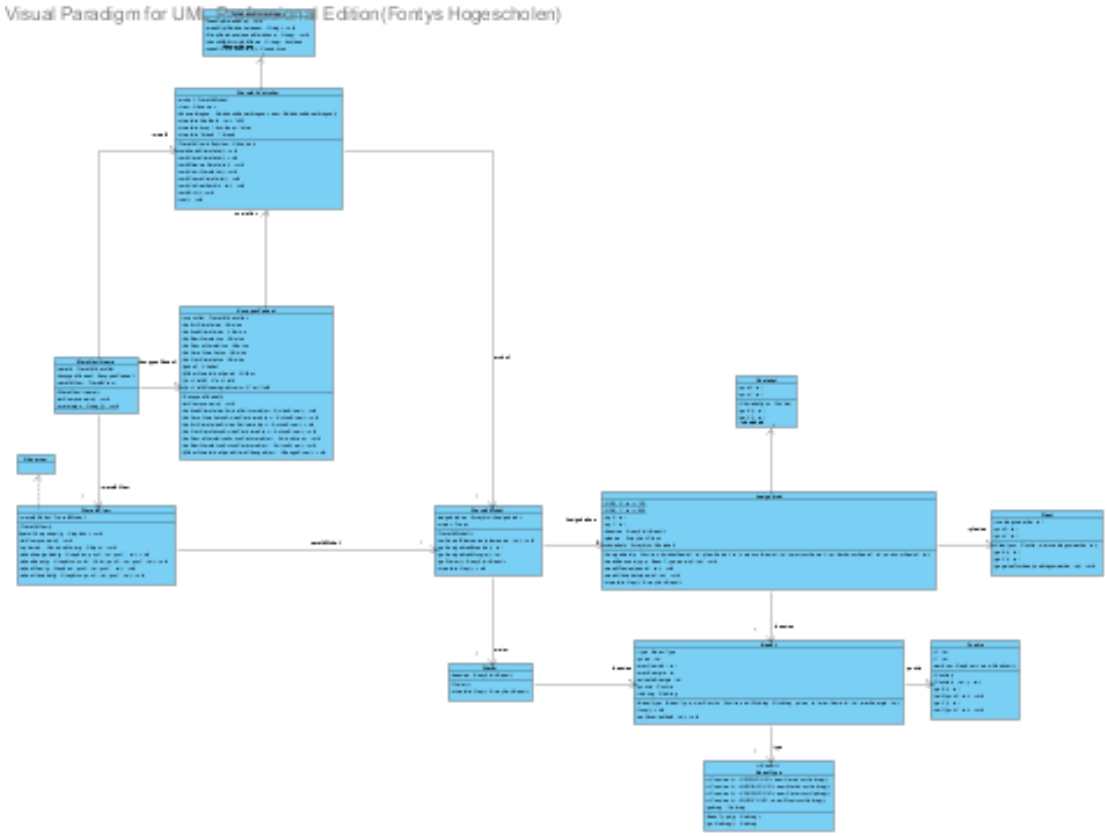


4 Gedetailleerde architectuur

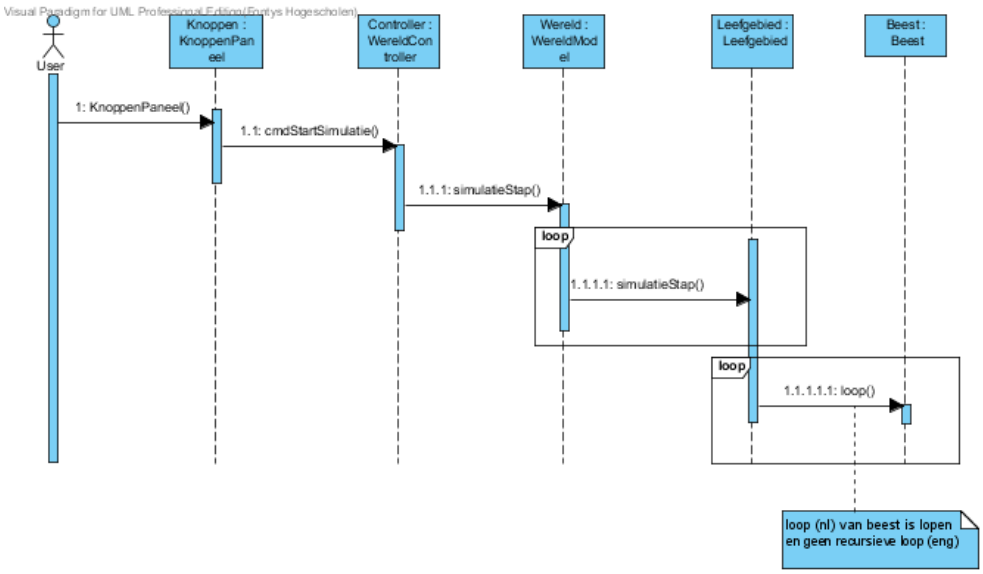
4.1 Klasse diagram model



4.2 Klasse diagram Model, View, Controller



4.3 Sequence diagram simulatiestap



5 Klassen

5.1 Beest

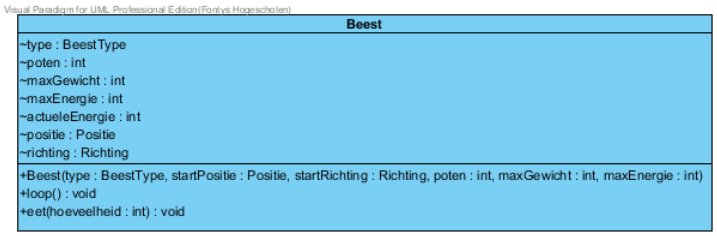
Een beest is in staat om zijn eigen positie te bepalen. Hiervoor is gekozen zodat het leefgebied dit niet van alle beesten hoeft bij te houden.

De vraag die nu rijst is: Hoe weet het beest waar hij naar toe mag? Er kan immers een obstakel in de weg staan.

Uiteindelijk kiezen we er toch voor dat het leefgebied de positie gaat bepalen. Deze weet immers ook waar de obstakels staan enz. Het beest kan wel zijn positie onthouden. Hij krijgt dus van het leefgebied op naar welke positie hij moet.

Omdat het leefgebied ook de obstakels kent zal deze ook de richting van het beest bepalen. Hij kijkt of het nieuwe coördinaat van het beest bezet met een obstakel is en als dit zo is dan zal het leefgebied de richting van het beest veranderen.

5.1.1 Diagram



5.1.2 Functie

Deze klasse is een representatie van een beest.

5.1.3 Constructor(s)

```
public Beest(BeestType type, Positie startPositie, Richting startRichting, int
               poten, int maxGewicht, int maxEnergie)
{
this.type = type;
       this.poten = poten;
       this.maxGewicht = maxGewicht;
       this.maxEnergie = maxEnergie;
       this.positie = startPositie;
       this.richting = startRichting;
}
```

parameters

type	naam	omschrijving
BeestType	type	Type dat een beest kan zijn: Omnivoor, Herbivoor, Carnivoor, Nonivoor
Positie	startPositie	Positie waar het beest begint binnen het leefgebied
Richting	startRichting	De looprichting waarmee het beest start
int	poten	Het aantal poten wat het beest heeft
int	maxGewicht	Het maximale gewicht wat het beest kan krijgen
int	maxEnergie	De maximale hoeveelheid energie die een beest kan krijgen

5.1.4 Methoden

Loop verzorgt een simualtie stap waarbij het beest afhankelijk van zijn richting een gridstap maakt.

```
public void loop()
{
Positie vorige = new Positie();

       vorige.X = positie.X;
       vorige.Y = positie.Y;

       positie.X += richting.getVerplaatsingX();
       positie.Y += richting.getVerplaatsingY();
}
```

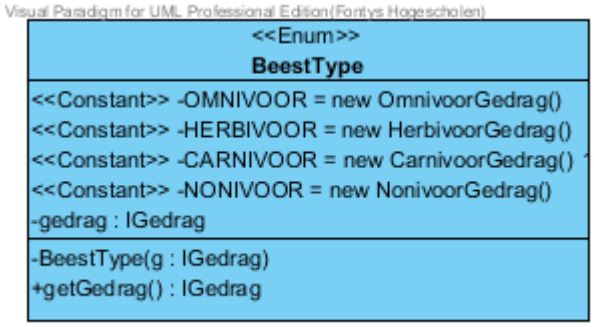
Eet kan worden gebruikt om het beest te laten eten. Afhankelijk van zijn gedrag zal de hoeveelheid worden verwerkt.

```
public void eet(int hoeveelheid)
{
       IGedrag gedrag = this.type.getGedrag();
        actueleEnergie = gedrag.eet(hoeveelheid, actueleEnergie);
}
```

5.2 BeestType

Enumeratie voor het type beest

5.2.1 Diagram



5.2.2 Functie

In de enumeratie BeestType ligt het gedrag vast. Als een beest bijvoorbeeld als type OMNIVOOR wordt gecreëerd, dan krijgt deze automatisch het Omnivoorgedrag. Het gedrag kan worden opgevraagd.

5.2.3 Constructor(s)

De constructor voor de enumeratie is private en kan niet worden aangeroepen.

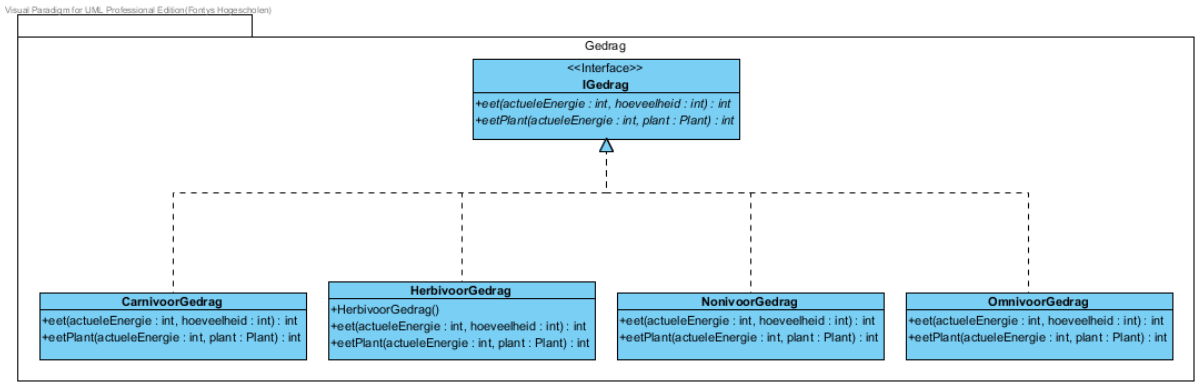
5.2.4 Methoden

Geeft het gedrag terug wat bij het type beest hoort.

```
public IGedrag getGedrag()
{
	return this.gedrag;
}
```

5.3 Gedrag

Het gedrag is vast gelegd in de interface IGedrag. Dit gedrag wordt voor ieder beesttype in een eigen klassen geïmplementeerd. Per implementatie kan het gedrag zoals eten en voorplanten specifiek worden gemaakt. Het gedrag wordt vastgelegd in de enum BeestType



5.4 OminovoorGedrag

Nog te beschrijven!

5.5 HerbivoorGedrag

Nog te beschrijven!

5.6 CarnivoorGedrag

Nog te beschrijven!

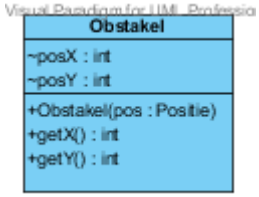
5.7 NonivoorGedrag

Nog te beschrijven!

5.8 Obstakel

Een obstakel krijgt bij de creatie een positie mee via de constructor. Deze positie blijft gedurende hele simulatie het zelfde. Doordat het leefgebied deze positie ook weet, is het niet nodig om een getter voor de positie te
maken. Omdat een obstakel alleen maar een positie gebruikt, is het dan nog wel nodig dat we een klasse obstakel aanmaken. Het leefgebied weet immers waar de obstakels staan, en weet ook waarheen de beesten verplaatsen.

5.8.1 Diagram



5.8.2 Functie

Deze klasse is een representatie van een obstakel.

5.8.3 Constructor(s)

```
public Obstakel(Positie pos)
{
	this.posX = pos.getX();
	this.posY = pos.getY();
}
```

parameters

type	naam	omschrijving
Positie	pos	Positie waar het obstakel staat binnen het leefgebied

5.8.4 Methoden

Getters om de positie van het obstakel op te vragen

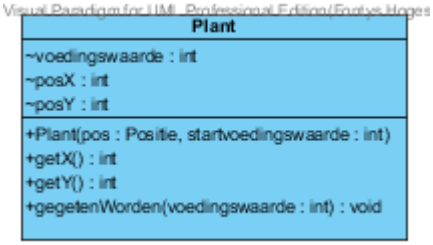
```
public int getX()
{
	return posX;
}

public int getY()
{
	return posY;
}
```

5.9 Plant

Een plant krijgt bij zijn creatie een start positie mee en een start voedingswaarde. De positie van de plant blijft gedurende de simulatie hetzelfde.
Het groeien van een plant gaat als volgt te werk:

- Plant wordt gecreëerd met een voedingswaarde. Mee gekregen via de constructor.
- Elke simulatie stap groet de plant 1 voedingswaarde
- Na het eten worde de voedingswaarde van de plant gereduceerd. Hoeveelheid is afhankelijk van het gedrag van een beest. Dit kan door gaan tot voedingswaarde 0.
- Indien de waarde 0 is kan de plant weer gewoon gaan groeien. Is deze “reïncarnatie” meer dan 10 maal uitgevoerd dan kan de plant niet meer groeien voor 100 simulatie stappen. Daarna gaan de plant weer gewoon groeien.

5.9.1 Diagram



5.9.2 Functie

Deze klasse is een representatie van een plant.

5.9.3 Constructor(s)

```
public Plant(Positie pos, int startvoedingswaarde)
{
	this.posX = pos.getX();
	this.posY = pos.getY();
	this.voedingswaarde = startvoedingswaarde;
}
```

5.9.4 Methoden

Getters om de positie van de plant op te vragen

```
public int getX()
{
	return posX;
}

public int getY()
{
	return posY;
}
```

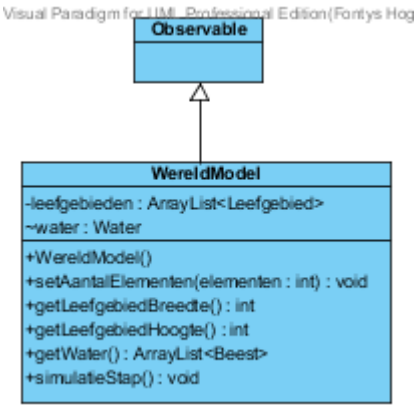
Door middel van de methode ‘gegetenWorden’ kan worden opgegeven hoeveel voedingswaarde er wordt afgegeten van een plant. Het resultaat wordt intern opgeslagen.

```
public void gegetenWorden(int voedingswaarde)
{
	this.voedingswaarde -= voedingswaarde;
}
```

5.10 Wereldmodel

De wereld bestaat uit twee leefgebieden en water. De wereld overerft van Observable zodat een Observer (GUI) geüpdatet wordt.

5.10.1 Diagram



5.10.2 Functie

Deze klasse is een representatie van een plant.

5.10.3 Constructor(s)

De constructor heeft geen parameters maar maakt bij aanroep een collectie aan voor de leefgebeiden en instantieerd een nieuw water object. Beide worden inter opgeslagen.

```
public WereldModel()
{
       leefgebieden = new ArrayList<>();
       water = new Water();
}
```

5.10.4 Methoden

Getters voor het opvragen van interne data.

```
public int getLeefgebiedBreedte()
{
    return 350;
}

public int getLeefgebiedHoogte()
{
    return 200;
}

public ArrayList<Leefgebied> getLeefgebieden()
{
    return leefgebieden;
}

public ArrayList<Beest> getWater()
{
    return water.getBeesten();
}
```

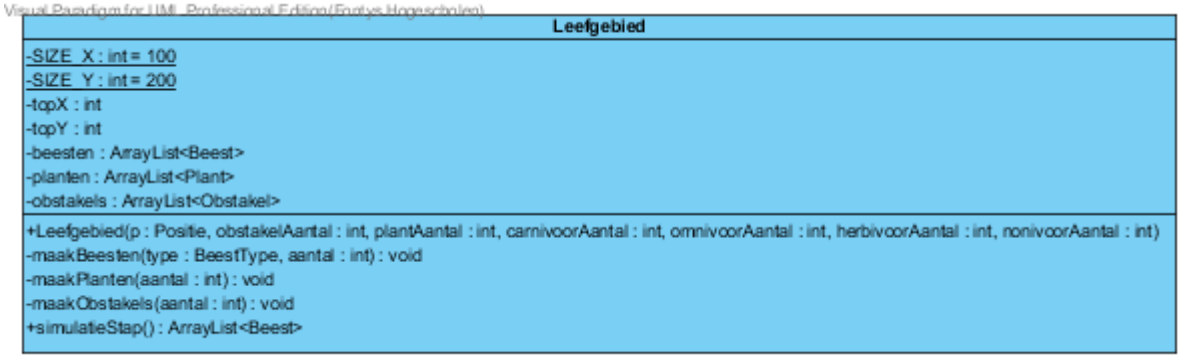
De methode ‘simulatieStap’ wordt vanuit de wereldcontroller aangeroepen. In deze methode worden alle leefgebieden doorlopen en van ieder leefgebied wordt ook de methode ‘simulatieStap’ aangeroepen. Dit wordt ook voor het water gedaan. Beesten die buiten een leefgebied collectie komen, komen in het water terecht. Aan het einde van de simulatie stap wordt notifyObservers van de klasse observable aangeroepen.

```
public void simulatieStap()
{
    ArrayList<Beest> afgevallenBeesten = new ArrayList<>();

    for(Leefgebied leefgebied: leefgebieden)
    {
        afgevallenBeesten = leefgebied.simulatieStap();

        for(Beest beest : afgevallenBeesten)
        {
            water.getBeesten().add(beest);
        }
    }

    afgevallenBeesten = water.simulatieStap();

    this.setChanged();
    this.notifyObservers();
}
```

5.11 Leefgebied

Het leefgebeid is een collectie van planten obstakels en beesten. Als een leefgebied geïnstantieerd wordt dan zal binnen het leefgebied het opgegeven aantal planten, obstakels en beesten worden gecreëerd.

5.11.1 Diagram



5.11.2 Functie

Deze klasse is een representatie van een plant.

5.11.3 Constructor(s)

```
public Leefgebied(Positie p, int obstakelAantal, int plantAantal, int
     carnivoorAantal, int omnivoorAantal, int herbivoorAantal, int nonivoorAantal)
{
    this.topX = p.X;
    this.topY = p.Y;

    this.beesten = new ArrayList<>();
    this.planten = new ArrayList<>();
    this.obstakels = new ArrayList<>();

    maakBeesten(BeestType.CARNIVOOR, carnivoorAantal);
    maakBeesten(BeestType.OMNIVOOR,  omnivoorAantal);
    maakBeesten(BeestType.HERBIVOOR, herbivoorAantal);
    maakBeesten(BeestType.NONIVOOR,  nonivoorAantal);
    maakPlanten(plantAantal);
    maakObstakels(obstakelAantal);
}
```

parameters

type	naam	omschrijving
Positie	p	Gridpositie van een leefgebied in de wereld
int	“aantallen”	Gewenste aantallen

5.11.4 Methoden

Getters voor het opvragen van interne data.

```
public int getTopX()
{
    return topX;
}

public int getTopY()
{
    return topY;
}

public ArrayList<Beest> getBeesten()
{
    return beesten;
}

public ArrayList<Plant> getPlanten()
{
    return planten;
}

public ArrayList<Obstakel> getObstakels()
{
    return obstakels;
}
```

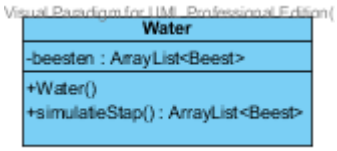
Private methoden voor het aanmaken van planten, obstakels en beesten en deze toe te voegen aan de bijbehorende collectie.

```
private void maakBeesten(BeestType type, int aantal){
    BeestFactory factory = BeestFactory.getInstance();

    for(int i=0; i < aantal; i++)
    {
        beesten.add(factory.createBeest(type));
    }
}

private void maakPlanten(int aantal){
    PlantFactory factory = PlantFactory.getInstance();

    for(int i=0; i < aantal; i++)
    {
        planten.add(factory.createPlant());
    }
}

private void maakObstakels(int aantal){
    ObstakelFactory factory = ObstakelFactory.getInstance();

    for(int i=0; i < aantal; i++)
    {
        obstakels.add(factory.createObstakel());
    }
}
```

5.12 Water

Het water is een klasse met daarin een collectie voor beesten. Beesten komen in het water als ze van een leefgebied af vallen.

5.12.1 Diagram



5.12.2 Functie

Deze klasse is een representatie van een plant.

5.12.3 Constructor(s)

De constructor heeft geen parameters maar maakt bij aanroep een collectie aan voor de beesten die zich in het water bevinden.

```
public Water()
{
    beesten = new ArrayList<>();
}
```

5.12.4 Methoden

Getter voor het opvragen van de collectie met beesten.

```
public ArrayList<Beest> getBeesten()
{
    return beesten;
}
```

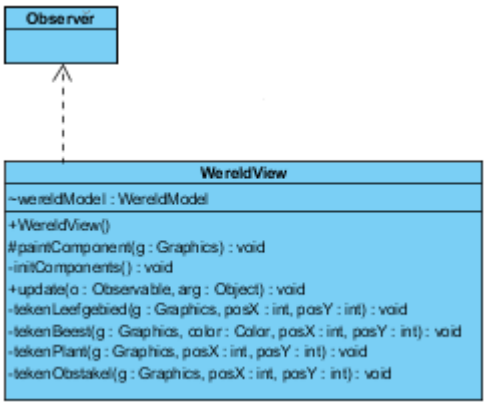
Bij de aanroep van deze methode wordt voor ieder beest in de collectie de methode ‘loop’ aangeroepen. Als beesten uit het water gaan dan komen deze in de collectie afgevallenBeesten terug.

```
public ArrayList<Beest> simulatieStap()
{
    ArrayList<Beest> afgevallenBeesten = new ArrayList<>();

    for(Beest beest: beesten)
    {
            ……
    }
    return afgevallenBeesten;
}
```

5.13 Wereldview

De wereldview is verantwoordelijk voor het tekenen van de wereld. De wereldview implementeert de interface van Observer zodat deze geüpdatet wordt vanuit de Wereld (Oservable). Wereldview overerft van JPanel.

5.13.1 Diagram



5.13.2 Functie

Verantwoordelijk voor het tekenen van de wereld.

5.13.3 Constructor(s)

De constructor heeft geen parameters.

```
public WereldView()
{
	initComponents();
	this.setBackground(Color.blue);
}
```

5.13.4 Methoden

De methode update wordt vanuit de wereld aangeroepen.

```
@Override
public void update(Observable o, Object arg)
{
	wereldModel = (WereldModel)o;
	this.repaint();
}
```

De methode tekenBeest is verantwoordelijk voor het grafisch tekenen van de beesten.

```
private void tekenBeest(Graphics g, Color color, int posX, int posY)
{
	g.setColor(color);
	g.fillRect(posX, posY, 5, 5);
}
```

De methode tekenPlant is verantwoordelijk voor het grafisch tekenen van planten

```
private void tekenPlant(Graphics g, int posX, int posY)
{
	g.setColor(Color.GREEN);
	g.fillRect(posX, posY, 5, 5);
}
```

De methode tekenObstakel is verantwoordelijk voor het grafisch tekenen van obstakels

```
private void tekenObstakel(Graphics g, int posX, int posY)
{
	g.setColor(Color.BLACK);
	g.fillRect(posX, posY, 5, 5);
}
```

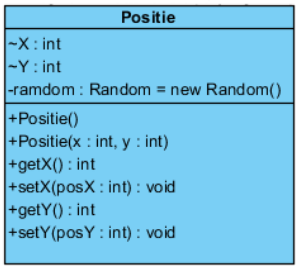
Deze methode is verantwoordelijk voor alle tekenbewerkingen. paintComponent wordt in deze klasse overschreven (Overrride). Derhalve wordt de methode van de superKlasse als eerste aangeroepen.

```
@Override
protected void paintComponent(Graphics g)
{
	super.paintComponent(g);
	.
	.
	if(wereldModel != null)
	{
		int i = 0;
		for(Leefgebied leefgebied: wereldModel.getLeefgebieden())
		{
			.
			.
		}
	}
}
```

5.14 Positie

De positie is een algemene klasse voor het onthouden van een X en een Y positie. De positie heeft twee constructors. Bij een constructor kan een gewenste X en Y worden meeegegeven terwijl bij de andere constructor de X en Y random worden bepaald. De klassen wordt onder andere gebruikt door beesten, planten en obstakels

5.14.1 Diagram



5.14.2 Functie

Het onthouden van een X en Y positie.

5.14.3 Constructors(s)

De eerste constructor zorgt voor een initiële random X en Y positie. Deze posities worden intern opgeslagen.

```
public Positie()
{
	X = ramdom.nextInt(100);
	Y = ramdom.nextInt(200);
}
```

Bij de tweede constructor kan een gewenste initiële X en Y positie worden opgegeven. Deze posities worden intern opgeslagen.

```
public Positie(int x, int y)
{
	X = x;
	Y = y;
}
```

5.14.4 Methoden

Getters en setter voor het opvragen en instellen van interne data.

```
public int getX()
{
	return X;
}
```

```
public void setX(int posX)
{
	this.X = posX;
}
```

```
public int getY()
{
	return Y;
}
```

```
public void setY(int posY)
{
	this.Y = posY;
}
```

6 Het lezen van de XML file met instellingen

Om de instellingen die gebruikt worden, op te kunnen slaan en te kunnen wijzigen, wordt er tijdens het aan maken van een nieuwe simulatie instellingen geladen vanuit een XML bestand (LifeSettings.xml). Op deze manier worden de laatste instellingen bewaard.

Dit wordt gedaan door de methode “readXMLfile ()”.

Het in lezen van de XML file wordt gedaan met behulp van een DOM XML parser. De gegevens per type worden in een array geladen.

```
typeBeest[s]          = (String)typeBeestNodeList.item(0).getNodeValue().trim();
aantalPoten[s]        =
Integer.valueOf(aantalPotenNodeList.item(0).getNodeValue());
maxGewicht[s]  =  Integer.valueOf(maxGewichtNodeList.item(0).getNodeValue());
maxEnergie[s]   =  Integer.valueOf(maxEnergieNodeList.item(0).getNodeValue());
```

Vervolgens worden de gegevens aan de setters van het beest type gekoppeld

```
instellingenCarnivoor.setAantalPoten( aantalPoten[0]);
instellingenCarnivoor.setMaxGewicht( maxGewicht[0]);
instellingenCarnivoor.setMaxEnergie( maxEnergie[0]);
```

Bovenstaande wordt voor elk type beest herhaald.

7 Beschrijving van de klasse database bewerkingen.

Methode:

De methode LaadSqlDatabase(), het is de bedoeling dat deze methode de database met simulatie gegevens van de SQL database server in leest als er en connectie met de server is

```
public void laadSqlDatabase()
    {
        Connection con = openSQLConnection();
        if (con != null)
        {
       // Code om de gegevens in te lezen
        }
    }
```

Wanneer men een simulatie wil opslaan, dan wordt er eerst gekeken of er al een database bestaat met dezelfde naam. Indien de naam al bestaat krijgt de gebruiker de keuze om de bestande database te verwijderen. Indien de naam niet bestaat maakt deze methode een database aan met de bijbehorende tabellen.

```
public void maakSqlDatabase(String naam)
{
…
…
}
```

Deze methode is gemaakt om een database van de server te verwijderen indien deze bestaat.

```
public void DropDatabase(String naamDatabase)
{
…..
….
}
```

Deze method wordt gebruikt om te kijken of een bepaalde database al aanwezig is op de sql server.
Indien de database met opgegeven naam al aanwezig is op de server dan retourneert de methode true anders retourneert de methode false

```
public boolean checkDBExists(String dbName)
{
……
….
}
```

8 Requirements traceability matrix

Requirement	Implementatie in klasse
N001	Wereldview
N002	DatabaseBewerkingen
F001	Wereldmodel, Wereldview
F002	Wereldview
F003	Wereldmodel
F004	Positie
F005	Leefgebied
F006	Wereldmodel
F007	Wereldmodel
F008	Wereldcontroller, Wereldmodel, Leefgebied, Beest
F009	Wereldcontroller, Knoppenpaneel
F010	Gedrag
F011	DatabaseBewerkingen
F012	DatabaseBewerkingen
F013	SimulatiegegevensDialog
F014	
F015	
F016	
F017	
F018	
F019	
F020	
F021	
F022	
F023	
F024	
F025	
F026	
F027	
F028	
F029	
F030	

9 Bevindingen

9.1 Bart

Terugkijkende op het semester denk ik dat ik wel kan stellen dat er een behoorlijke race gelopen is. Gedurende de race is er veel geleerd. In ieder geval kan ik zeggen dat ik inmiddels een redelijke hoeveelheid kennis he opgedaan over Java.

Wat betreft software engineering denk ik dat ik iets heb kunnen leren over methoden die toegepast kunnen worden tijdens het ontwerpproces. Wel denk ik dat de casus te groot was waardoor ik in mindere mate heb geleerd hoe de methoden goed te gebruiken om een samenhangend ontwerp te realiseren.

9.2 Patrick

Als ik terug kijk naar de afgelopen paar maanden, dan kan ik voor mezelf zeker zeggen dat ik veel, heel veel geleerd heb. Het waren geen makkelijke maanden en jammer genoeg was het niet altijd even duidelijk.

Het hele software ontwerp proces is in een zeer hoge sneltrein vaart lang me voor bij geschoten. De tijd is veel te kort om het ontwerp proces goed te begrijpen. De kracht van diagrammen en beschrijvingen is nu wel erg duidelijk geworden.

Het Java stuk vond ik erg leuk, daar ben ik ook flink in gegroeid. En ik heb er ook veel aan. Hulp tooltjes op mijn werk maak ik steeds vaker in Java.

De casus was echt een goede leerschool. Alles bij elkaar een hele wijze les. Wat me erg frustreert is dat het te groot was. Het zit nu eenmaal in je om iets perfect werkend af te willen leveren. En dat lukte niet.