Technisch Ontwerp

VNU Floor Manager Routeplanner & 3D map

2009

C.Floor

Evident Interactive

10-12-2009

**Inhoudsopgave**

[Inleiding 2](#_Toc248204528)

[Globale opzet applicatie 3](#_Toc248204529)

[Klasse diagrammen en klasse beschrijvingen 4](#_Toc248204530)

[Klasse diagram: Silverlight 3 applicatie 4](#_Toc248204531)

[Klasse beschrijvingen: Silverlight 3 applicatie 5](#_Toc248204532)

[Klasse diagram: Webservice 6](#_Toc248204533)

[Klasse beschrijvingen: Webservice 6](#_Toc248204534)

[Methoden beschrijvingen 7](#_Toc248204535)

[Silverlight 3 applicatie 7](#_Toc248204536)

[De Webservice 12](#_Toc248204537)

[De Routeplanner 14](#_Toc248204538)

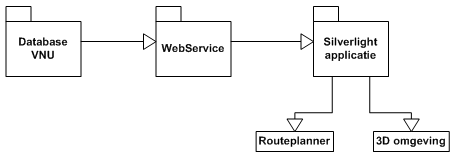
# Inleiding

In dit document wordt beschreven hoe het systeem in elkaar zit. Met andere woorden hierin wordt verteld hoe het systeem dat in het F.O. beschreven is technisch in elkaar zit.

Als eerst zal ik een overzicht geven van hoe het systeem globaal in elkaar zit. In de hoofdstukken erna zal ik dieper ingaan op de code en tot slot zal ik ingaan op delen van het systeem die meer toelichting nodig hebben. Met name hoe het plannen van een route achter de schermen in zijn werk gaat.

# Globale opzet applicatie

In dit hoofdstuk wordt weergeven hoe het systeem globaal in elkaar zit aan de hand van een afbeelding.



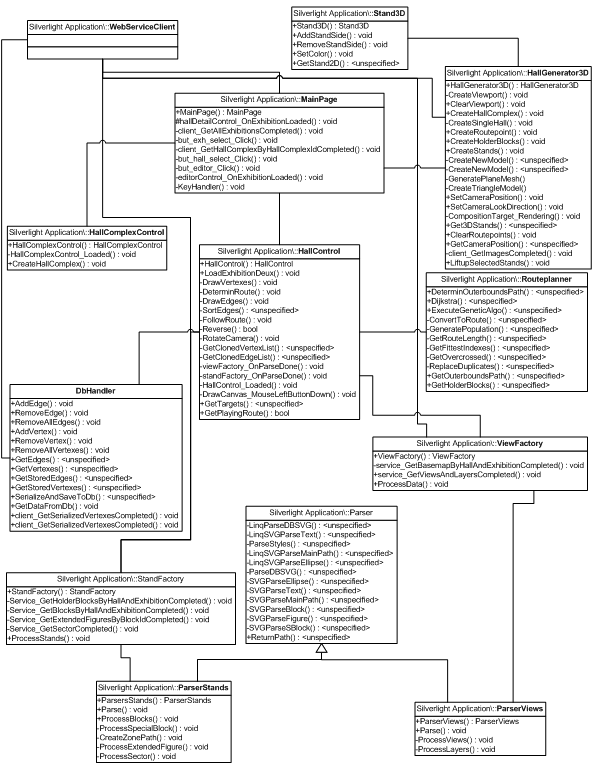
VNU (de klant) heeft een database met gegevens over beurzen. De Silverlight applicatie kan niet direct contact maken met de database. Daarom is er een webservice nodig. Wanneer de Silverlight applicatie data nodig heeft uit de database zal deze de webservice aanspreken. De webservice zal vervolgens de gewenste data ophalen uit de database en deze doorgeven aan de Silverlight applicatie.

De routeplanner en de 3D omgeving bevinden zich ook binnen de Silverlight applicatie. Maar dit zijn de twee functionaliteiten die ik moet toevoegen aan het systeem. Daarom heb ik ze weergegeven als aparte onderdelen.

# Klasse diagrammen en klasse beschrijvingen

In dit hoofdstuk zijn de klasse diagrammen te zien van het systeem. Per klasse zal ik ook een korte beschrijving geven van waar ze voor dienen ter verduidelijking van het klasse diagram.

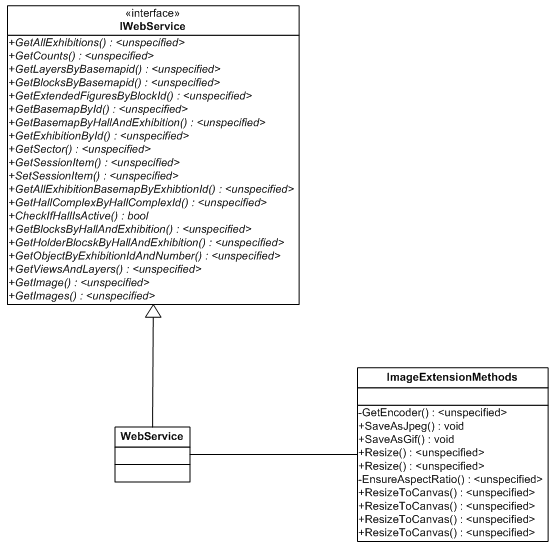
## Klasse diagram: Silverlight 3 applicatie



## Klasse beschrijvingen: Silverlight 3 applicatie

|  |  |
| --- | --- |
| **Klasse** | **Beschrijving** |
| MainPage | Dit is de hoofdklasse van de applicatie. Hieraan zit ook de opmaak verbonden die je ziet bij het opstarten van de applicatie |
| HallComplexControl | Dit is de klasse die verantwoordelijk is voor het weergeven van het juiste hallencomplex dat door de gebruiker is gekozen |
| HallControl | Dit is de klasse die verantwoordelijk is voor het weergeven van de juiste hal binnen een hallencomplex in zowel 3D als in 2D op een plattegrond |
| HallGenerator3D | Deze klasse zorgt voor het generen van 3D objecten zoals stands, terrein en hallencomplexen |
| DbHandler | Deze klasse dient als een tussenlaag tussen Silverlight en de database op het gebied van routenetwerken |
| Stand3D | Elke hall op de 3D kaart word gerepresenteerd door een Stand3D object |
| WebServiceClient | Dit is de client klasse die communiceert met de externe webservice. Zo is voor de silverlight applicatie duidelijk welke methoden van de webservice hij kan aanroepen en wat voor parameters hij moet meegeven. Ook is zo duidelijk wat voor type object elke webservice methode teruggeeft |
| Routeplanner | Deze klasse is verantwoordelijk voor alles wat met het routeplannen te maken heeft. Hierin bevinden zich dan ook de meeste wiskundige methoden. De twee algoritmen die in deze klasse worden gebruikt zijn het kortste pad algoritme van Dijkstra en een Genetisch algoritme. |
| StandFactory | Deze klasse roept functies van de webclient aan om de stands te verzamelen |
| ViewFactory | Deze klasse roept functies van de webclient aan om de views te verzamelen |
| Parser | De data uit de Database (VML en SVG) moet worden omgezet in tekeningen die in Silverlight weergegeven kunnen worden. Deze klasse is een interface waarin zich methoden bevinden die dit doen |
| ParserStands | Deze parser is een subklasse van Parser en zorgt voor het genereren van de stands binnen Silverlight |
| ParserViews | Deze parser is een subklasse van Parser en zorgt voor het genereren van de verschillende lagen die bestaan. Een getekende kaart binnen Silverlight bestaat uit verschillende lagen. |

## Klasse diagram: Webservice



## Klasse beschrijvingen: Webservice

|  |  |
| --- | --- |
| **Klasse** | **Beschrijving** |
| IWebService | Dit is de interface voor de webservice. Deze beschrijft welke methoden zich allemaal in de webservice bevinden. |
| WebService | Deze klasse bevat alle methoden die in de interface worden beschreven. Webclients kunnen deze methoden aanroepen. Het gaat hier voornamelijk om methoden die data uit de database halen en doorsturen naar de Silverlight applicatie. |
| ImageExtensionMethods | Deze statische klasse bevat functionaliteit voor het scalen van images. Dit is nodig omdat het intern, in de Silverlight applicatie zelf, niet dynamisch mogelijk is |

# Methoden beschrijvingen

In dit hoofdstuk zal ik per klasse alle noemenswaardige methoden uitleggen door middel van een korte beschrijving.

## Silverlight 3 applicatie

|  |  |
| --- | --- |
| **MainPage** | |
| **Methode naam** | **Beschrijving** |
| hallControl\_OnExhibitionLoaded | Deze methode wordt uitgevoerd wanneer een beurs is opgehaald en ingeladen in de applicatie. Het HallControl-object wordt dan zichtbaar gemaakt in de applicatie. Een 3D model van een beurshal is zichtbaar. |
| Client\_GetAllExhibitionsCompleted | Deze methode wordt uitgevoerd wanneer alle beurzen door de WebServiceClient zijn opgehaald. Deze worden dan aan de lijst van te kiezen beurzen toegevoegd. |
| But\_exh\_select\_Click | Deze methode wordt uitgevoerd wanneer er op de select knop wordt gedrukt bij de beurslijst. De WebServiceClient haalt dan een hallencomplex op. |
| Client\_GetHallComplexByHallComplexIdCompleted | Wanneer het hallencomplex is ingeladen wordt deze methode uitgevoerd. Er wordt een HallComplexControl-object zichtbaar gemaakt in de applicatie. Een 3D model van het hallcomplex is zichtbaar. |
| But\_hall\_select\_Click | Deze methode wordt uitgevoerd wanneer er op de select knop word gedrukt bij de hallenlijst. Er wordt dan een **HallControl**-object aangemaakt. En een hal wordt opgehaald door de WebServiceClient |
| But\_editor\_Click | Deze methode wordt uitgevoerd wanneer er op de map editor knop wordt gedrukt. De map editor wordt dan geladen |
| editorControl\_OnExhibitionLoaded | Maakt de map editor zichtbaar op het scherm wanneer deze ingeladen is |
| KeyHandler | Deze methode wordt aangeroepen bij het indrukken van een knop op het toetsenbord. Aan de hand van welke knop wordt ingedrukt worden een bepaalde acties ondernomen. |
| But\_save\_Click | Deze methode wordt uitgevoerd wanneer op de save button wordt gedrukt. Er wordt dan data in de database gesaved. |

|  |  |
| --- | --- |
| **HallComplexControl** | |
| **Methode naam** | **Beschrijving** |
| HallComplexControl\_Loaded | Deze methode wordt aangeroepen als het HallComplexControl is ingeladen. |
| CreateHallComplex | In deze methode wordt het genereren van een hallencomplex aangeroepen bij de HallGenerator3D |

|  |  |
| --- | --- |
| **HallControl** | |
| **Methode naam** | **Beschrijving** |
| LoadExhibitionDeux | Laad aan de hand van een hall id en exhibition id een beurs in. Een standFactory-object en een viewfactory-object worden aangemaakt. |
| DrawVertexes | Tekent een lijst van meegegeven vertexen (punten) op de 2D kaart |
| DeterminRoute | Bepaalt het pad tussen alle punten van een route dat het optimale pad benaderd. Hierbij komen het Dijkstra algoritme en een genetisch algoritme van pas. Wanneer de route is berekent wordt deze route in de 3D map met de camera gevolgd. Hiervoor wordt een aparte thread gestart. |
| DrawEdges | Tekent de lijnen van de berekende route op de 2D kaart |
| SortEdges | Sorteert de lijnen van een route. Op deze manier volgt de camera in de 3D map de goeie route. |
| FollowRoute | Deze methode wordt uitgevoerd in een aparte thread. De taak van deze methode is om de camera in de 3D map de route te laten volgen. |
| Reverse | Bepaalt of een meegegeven lijn omgedraaid moet worden of niet. Dus of het beginpunt en het eindpunt van de lijn moeten worden omgedraaid. Dit is van belang voor de camera bij het volgen van de route. Zo weet de camera naar welk punt hij moet draaien (de lookAt coördinaten) |
| RotateCamera | Draait de camera naar het juiste punt op basis van een meegegeven lijn. |
| GetClonedVertexList | Cloont een meegegeven vertex-lijst en retouneert deze |
| GetClonedEdgeList | Cloont een meegegeven edge-lijst en retouneert deze |
| viewFactory\_OnParseDone | Voert slechts een check uit |
| standFactory\_OnParseDone | Voegt de 2D kaart toe aan het tekencanvas wanneer het parsen van de stands klaar is. |
| HallControl\_Loaded | Zorgt dat de 2D kaart zichtbaar wordt in de applicatie en genereert de 3D beurshal en maakt deze zichtbaar zodra het HallControl-object ingeladen is. |
| DrawCanvas\_MouseLeftButtonDown | Wordt uitgevoerd wanneer de linker muisknop wordt ingedrukt op het DrawCanvas. Dit wordt gebruikt voor het selecteren van een stand op de 2D kaart. De routeplanner kan deze dan meenemen bij het berekenen van een korte route. |
| GetTargets | Geeft de lijst met doelen (vertexen) die in de route voor moeten komen terug |
| GetPlayingRoute | Geeft een bool terug die aangeeft of een route op het moment bezig is met afspelen of niet |

|  |  |
| --- | --- |
| **HallGenerator3D** | |
| **Methode naam** | **Beschrijving** |
| CreateViewport | Koppelt een camera object aan de viewport en voegt deze toe aan het grid dat aan de constructor is meegegeven |
| ClearViewport | Alle objecten die aan de viewport zijn toegevoegd worden verwijderd. |
| CreateHallComplex | Genereert een hallencomplex in 3D aan de hand van een lijst van Hall objecten die word meegegeven. |
| CreateSingleHall | Genereert een enkele hall die dan word toegevoegd aan het hallencomplex |
| CreateRoutepoint | Genereert een routepunt (een rode punt) in de 3D map |
| CreateHolderBlocks | Genereert de holderblocks (oranje vlakken) in een beurshall |
| CreateStands | Genereert de stands in een hal |
| CreateNewModel | Genereert een 3D model op de opgegeven coördinaten, met de opgegeven maten en kleur |
| CreateNewModel | Genereert een 3D model op de opgegeven coördinaten, met de opgegeven maten en texture |
| GeneratePlaneMesh | Genereert een mesh met de meegegeven afmetingen |
| CreateTriangleModel | Genereert een triangle model met de meegegeven afmetingen (3 punten) en kleur. De 3D modellen zijn opgebouwd uit driehoeken, zo werkt dit in de meeste 3D engines. |
| SetCameraPosition | Zet de camera op een meegegeven positie in de 3D ruimte |
| SetCameraLookDirection | Zet de kijkrichting van de camera. |
| CompositionTarget\_Rendering | Deze functie wordt uitgevoerd bij het refreshen van een frame. Hier worden dus alle updates uitgevoerd bij veranderingen in de 3D ruimte. |
| Get3DStands | Geeft een lijst terug met de Stand3D objecten |
| ClearRoutepoints | Verwijderd de gegenereerde routepunten van de 3D map |
| GetCameraPosition | Geeft de positie (x,y en z) van de camera terug |
| Client\_GetImagesCompleted | Wordt uitgevoerd wanneer alle geschaalde textures (bedrijfssymbolen) zijn opgehaald door de WebServiceClient. Vervolgens worden de stands toegevoegd aan de hal. |
| CreateCube | Genereert de kubus die boven stands verschijnt |
| MakeInfoCubeVisible | Laat de kubus zichtbaar worden of verdwijnen van het scherm aan de hand van een bool |

|  |  |
| --- | --- |
| **DbHandler** | |
| **Methode naam** | **Beschrijving** |
| AddEdge | Voegt een edge toe aan het te maken routenetwerk |
| RemoveEdge | Verwijderd een edge van het te maken routenetwerk |
| RemoveAllEdges | Verwijderd alle edges uit het te maken routenetwerk |
| AddVertex | Voegt een vertex toe aan het te maken routenetwerk |
| RemoveVertex | Verwijderd een vertex van het te maken routenetwerk |
| RemoveAllVertexes | Verwijderd alle vertexen uit het te maken routenetwerk |
| GetEdges | Geeft alle edges van het te maken routenetwerk |
| GetVertexes | Geeft alle vertexen van het te maken routenetwerk |
| GetStoredEdges | Geeft alle opgeslagen edges met het meegegeven hall id en exhibition id |
| GetStoredVertexes | Geeft alle opgeslagen vertexen met het meegegeven hall id en exhibition id |
| SerializeAndSaveToDb | Slaat alle vertexen en edges op in de database |
| GetDataFromDb | Geeft alle opgeslagen vertexen en edges met het meegegeven hall id en exhibition id |

|  |  |
| --- | --- |
| **Stand3D** | |
| **Methode naam** | **Beschrijving** |
| AddStandSide | Voegt een zijde toe aan het 3D model van een stand |
| RemoveStandSide | Verwijderd een zijde van het 3D model van een stand |
| SetColor | Zet de kleur van een stand op de meegegeven kleur |
| GetStand2D | Geeft de 2D stand terug (de stand op de 2D kaart) |

|  |
| --- |
| **WebServiceClient** |
| Deze klasse dient als contactpunt tussen de webservice en de Silverlight applicatie. Voor een beschrijving van de methodes die deze klasse kan aanroepen, raadpleeg de klasse ***WebService*** in het WebService package. |

|  |  |
| --- | --- |
| **Routeplanner** | |
| **Methode naam** | **Beschrijving** |
| DeterminOuterboundsPath | Bepaalt de grenzen van een beurshal |
| ExecuteDijkstra | Voert het Dijkstra algoritme uit op een verzameling van vertexen. Je geeft een source vertex op. Voor elke andere vertex wordt dan de kortste route berekent vanaf deze source. |
| ExecuteGeneticAlgo | Voert een genetisch algoritme uit op een lijst van vertexen. Hier komt uiteindelijk een korte route uit die langs alle gekozen stands gaat. (Lees de eindscriptie van dit project voor meer info over het genetische algoritme) |
| ConvertToRoute | Maakt een route van een lijst van vertexen |
| GeneratePopulation | Genereert een populatie van willekeurige routes. Deze populatie dient als basis voor het genetische algoritme |
| GetRouteLength | Berekent de lengte van een route en geeft deze lengte terug. |
| GetFittestIndexes | Bepaalt uit de populatie (lijst) de locaties van de kortste routes. |
| GetOvercrossed | Kruist de kortste paden. |
| ReplaceDuplicates | Verwijderd en vervangt punten die twee keer voorkomen in een route veroorzaakt door overkruising. |
| GetOuterboundsPath | Geeft de grenzen van een beurshal terug |
| GetHolderBlocks | Geeft een lijst met de holderblocks terug. |

|  |  |
| --- | --- |
| **StandFactory** | |
| **Methode naam** | **Beschrijving** |
| Service\_GetHolderBlocksByHallAndExhibitionCompleted | Wordt uitgevoerd wanneer de holderblocks zijn opgehaald door de WebServiceClient. |
| Service\_GetBlocksByHallAndExhibitionCompleted | Wordt uitgevoerd wanneer blokken uit verschillende lagen zijn opgehaald door de WebServiceClient. |
| Service\_GetExtendedFiguresByBlockIdCompleted | Wordt uitgevoerd wanneer extended figures zijn opgehaald door de WebServiceClient. |
| Service\_GetSectorCompleted | Wordt uitgevoerd wanneer een bepaalde sector is opgehaald door de WebServiceClient. |
| ProcessStands | Verwerkt de opgehaalde stands. |

|  |  |
| --- | --- |
| **ViewFactory** | |
| **Methode naam** | **Beschrijving** |
| Service\_GetBasemapByHallAndExhibitionCompleted | Wordt uitgevoerd wanneer de basismap (de hoofdlaag) van een hal is opgehaald |
| Service\_GetViewsAndLayersCompleted | Wordt uitgevoerd wanneer alle lagen voor een map zijn opgehaald. |
| ProcessData | Verwerkt de opgehaalde data. |

|  |  |
| --- | --- |
| **Parser** | |
| **Methode naam** | **Beschrijving** |
| LinqParseDBSVG | Zet SVG Data afkomstig uit de database om in objecten die in Silverlight kunnen worden weergegeven. (LINQ versie) |
| LinqSVGParseText | Filtert tekst uit SVG data afkomstig uit de database en maakt dit klaar voor gebruik binnen Silverlight. (LINQ versie) |
| ParseStyles | - |
| LinqSVGParseMainPath | Geeft de buitenste lijnen van een hal in de vorm van een path object terug. (LINQ versie) |
| LinqSVGParseEllipse | Zet een cirkel om naar een object dat in Silverlight gebruikt kan worden. |
| ParseDBSVG | Zet SVG Data afkomstig uit de database om in objecten die in Silverlight kunnen worden weergegeven. |
| SVGParseEllipse | Zet een cirkel om naar een object dat in Silverlight gebruikt kan worden. |
| SVGParseText | Filtert tekst uit SVG data afkomstig uit de database en maakt dit klaar voor gebruik binnen Silverlight. |
| SVGParseMainPath | Geeft de buitenste lijnen van een hal in de vorm van een Path opbject terug. |
| SVGParseBlock | Zet een rechthoek/blok om in een object geschikt voor Silverlight en geeft dit object terug |
| SVGParseFigure | Zet een figuur om in een Path-object dat geschikt is voor Silveright en geeft dit object terug. |
| SVGParseSBlock | Zet een speciaal blok (niet rechthoekig) om in een object geschikt voor Silverlight en geeft dit object terug. |
| ReturnPath | Zet VML data om naar een Path object en geeft dit object terug. |

|  |  |
| --- | --- |
| **ParserStands : Parser** | |
| **Methode naam** | **Beschrijving** |
| Parse | Start het omzetten van blokken. Roept ProcessBlocks aan. |
| ProcessBlocks | Zet blocks om in Holderblocks. |
| ProcessSpecialBlock | Zet een special Block om in Holderblock in de vorm van een Path-object. |
| CreateZonePath | Genereert een bepaalde zone/gebied. |
| ProcessExtendedFigure | Zet een ExtendedFigure object om in een Path object. |
| ProcessSector | Genereert een bepaalde sector. |

|  |  |
| --- | --- |
| **ParserViews : Parser** | |
| **Methode naam** | **Beschrijving** |
| Parse | Start het omzetten van blokken. Roept ProcessLayers aan. |
| ProcessViews | Zet blocks om in lagen. |
| ProcessLayers | Bepaalt welke lagen wel en niet te zien zullen zijn. |

## De Webservice

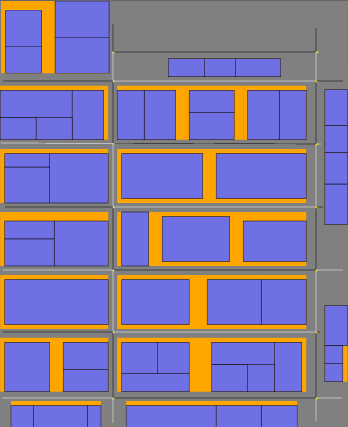
|  |  |
| --- | --- |
| **WebService : IWebService** | |
| **Methode naam** | **Beschrijving** |
| GetAllExhibtions | Haalt alle beursen op uit de database |
| GetCounts |  |
| GetLayersByBasemapid | Haalt alle lagen van een bepaalde hal op, op basis van basemap id |
| GetBlocksByBasemapid | Haalt alle blokken op van een bepaalde hal op basis van basemap id |
| GetExtendedFiguresByBlockId | Haalt alle ExtendeFigure objecten op uit de database voor een blok op basis van het Block id. |
| GetBasemapById | Haalt de basemap op, op basis van het basemap id |
| GetBasemapByHallAndExhibition | Haalt de basemap op uit de database, op basis van het hall id en exhibition id. |
| GetExhibitionById | Haalt een beurs op uit de database, op basis van het exhibition id. |
| GetSector | Haalt een sector op uit de database, op basis van een sector id. |
| GetSessionItem | Geeft het huidige session item terug. |
| SetSessionItem | Elke keer als er een beroep wordt gedaan op de webservice word er een nieuwe session item aangemaakt. |
| GetAllExhibitionBasemapByExhibitionId | Haalt alle basemaps op van elke hal voor een bepaalde beurs op basis van een exhibition id |
| GetHallComplexByHallComplexId | Haalt een hallencomplex op uit de database op basis van een hallComplex id. |
| CheckIfHallsActive | Controleert of een bepaalde hall word gebruikt voor een beurs, op basis van een exhibition id en een hall id. |
| GetBlocksByHallAndExhibition | Haalt alle blokken SVG data op voor een bepaalde beurs op basis van hall id en exhibition id |
| GetHolderBlocksByHallAndExhibition | Haalt alle holderBlocks op, op basis van hall id en exhibition id |
| GetObjectByExhibitionIdAndNumber | Haalt een bepaalt object op, op basis van exhibition id, object nummer en object type. |
| GetViewsAndLayers | Haalt de verschillende lagen van een basemap op, op basis van basemap id |
| GetImage | Haalt een bepaalde afbeelding op, op basis van een serverpath en schaalt deze naar de meegegeven breedte en hoogte. |
| GetImages | Haalt een lijst van afbeeldingen op en schaalt deze naar de meegegeven afmetingen. |

|  |  |
| --- | --- |
| **ImageExtensionMethods** | |
| **Methode naam** | **Beschrijving** |
| GetEncoder | Geeft een object met informatie terug over de image encoder. |
| SaveAsJpeg | Slaat Image object op als Jpeg afbeelding naar een opgegeven pad met een opgegeven kwaliteit |
| SaveAsGif | Slaat Image object op als gif afbeelding naar een opgegeven pad met een opgegeven kwaliteit |
| Resize | Schaalt een afbeelding naar de meegegeven afmetingen (breedte, hoogte) en met een opgegeven interpolatie mode |
| Resize | Schaalt een afbeelding naar de meegegeven afmetingen (breedte, hoogte). |
| EnsureAspectRatio | Schaalt een afbeelding naar de meegegeven afmetingen (breedte, hoogte) waarbij de breedte en hoogte verhouding hetzelfde blijft (de aspect ratio). |

# De Routeplanner

In dit hoofdstuk wordt uitgelegd hoe de routeplanner precies werkt. Enkel het commentaar in de code en de uitleg van de verschillende methoden in dit document is hierbij niet voldoende om de werking te begrijpen. Vandaar dat hier nog extra aandacht aan wordt besteed.

Op de afbeelding hieronder is een wegennetwerk te zien. Dit netwerk bestaat uit lijnen en punten. De punten zijn de uiteinden van de lijnen en de kruispunten tussen de lijnen. Deze punten en lijnen zijn nodig bij het kortste pad algoritme van Dijkstra.



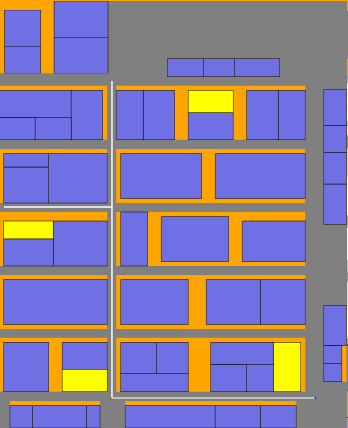
Dit algoritme berekent de kortste afstand tussen alle punten. Hierdoor is dus bekent wat de kortste route is tussen elk tweetal punten. Op deze manier is er dus al bekend wat de kortste weg is van het punt rechts onder in de map naar het punt links boven. Deze kortste wegen bestaan uit een deelverzameling van lijnen en punten uit het bovenstaande routenetwerk.

Op dat moment weet je dus enkel de kortste wegen tussen elk tweetal punten, maar als we stands selecteren zal je dus een combinatie moeten maken van deze kortste wegen om een route te krijgen langs alle gekozen stands.

Hierbij komt het handelsreizigersprobleem (<http://nl.wikipedia.org/wiki/Handelsreizigersprobleem>) kijken. Kort gezegd houdt dit probleem in:  
  
“Als er *n* steden gegeven zijn die een handelsreiziger moet bezoeken, samen met de afstand tussen ieder paar van deze steden, vindt dan de kortste weg die kan worden gebruikt, waarbij iedere stad precies eenmaal wordt bezocht.”

Nu hebben wij natuurlijk niet te maken met steden en handelsreizigers. Vervang steden door beursstands en handelsreiziger door beursbezoeker en je hebt dan het probleem waar wij mee te maken hebben.

In onderstaande afbeelding is een route te zien. Deze route is een deel van het totale routenetwerk zoals in de vorige afbeelding te zien is.



De stappen die geleid hebben tot deze route zal ik nu uitleggen. Wanneer het berekenen van de route wordt aangeroepen zal gekeken worden welke stands geselecteerd zijn (de gele blokken). Per stand word er gekeken wat het middelpunt is van die stand.

Een simpele berekeningen van standhoogte / 2 en standbreedte / 2 is al genoeg. Vervolgens wordt gekeken welk punt/vertex uit het totale routenetwerk het dichts bij dit middelpunt ligt. Dit punt/vertex wordt dan meegenomen in de punten die in te berekenen route moeten zitten.

Wanneer dit voor elke stand die geselecteerd is gedaan is kan het berekenen van de route echt beginnen. Met het Dijkstra algoritme zijn dus alle kortste wegen tussen elk tweetal punten berekent. Nu moet er een complete route worden berekent die alle punten bevat die aan de hand van de geselecteerde routes zijn bepaald.

Nu word het genetische algoritme uitgevoerd. Een genetisch algoritme is een algoritme dat gebaseerd is op de evolutie theorie (<http://nl.wikipedia.org/wiki/Genetisch_algoritme>). Dit algoritme doorloopt drie stappen. Op de volgende bladzijde word aangegeven welke stappen dit zijn en welke methoden in de applicatie hierbij een rol spelen.

* Initialisatie
  + Routeplanner.GeneratePopulation
* Selectie
  + Routeplanner.GetFittestIndexes
* Reproductie
  + Routeplanner.GetOvercrossed
  + Routeplanner.ReplaceDuplicates

Bij de initialisatie wordt een basispopulatie gegenereerd. Bij ons houdt dit in dat er een verzameling van 500 willekeurige routes word gemaakt.

De selectie houdt in dat de sterkste uit de populatie worden geselecteerd. De sterkte hangt in ons geval af van de lengte van een route. Hoe korter een route hoe sterker hij is. De 10 kortste routes worden bepaald.

Vervolgens vindt de reproductie plaats. De 10 korte routes worden opgedeeld in 5 paren van 2. Een paar bestaat dus uit 2 korte (sterke) routes. Deze 2 routes noemen we de ouderparen. Die ouderparen splitsen we elk in 2. Vervolgens kruisen we de ouders en zo krijgen we de kinderparen. Onderstaande reeks moet dit verduidelijken.

* Ouder 1 = B A D E G F H J I C
* Ouder 2 = G I D F H E B A C J
* Kind 1 = B A D E G E B A C J
* Kind 2 = G I D F H F H J I C

Zo krijg je twee nieuwe routes. Dit wordt voor elke van de 5 paren gedaan. Zo krijg je 10 nieuwe korte routes.

Een probleem dat zich echter voor kan doen is dat een route twee keer hetzelfde punt bevat en dat mag niet. Elk punt hoeft tenslotte maar 1 keer bezocht te worden. Wanneer een punt dus twee keer in een route voorkomt zal de tweede keer dat dit punt voorkomt worden vervangen door een ander willekeurig punt dat niet in de route voorkomt. Zo komt er in kind 1 twee keer het punt A voor. Dit punt zal worden vervangen door een ander punt dat in de route mag zitten.

Wanneer dit is gedaan worden de ouders uit de populatie vervangen door de kinderen. Op deze manier krijg je een nieuwe populatie. Dan begint het proces van selectie en reproductie weer van voor af aan. Dit word 200 keer opnieuw gedaan.

Uiteindelijk krijg je dan een korte route. Deze route is misschien niet de kortst mogelijke route. Maar hij zal hem wel benaderen. De kortste route vinden is geen mogelijkheid in dit geval aangezien dit teveel tijd in beslag gaat nemen wanneer er veel stands worden geselecteerd. Voor meer informatie hierover verwijs ik naar de eindscriptie. En dan naar het onderzoek over het traveling salesman problem (handelsreizigersprobleem).