Project Grafische Technieken 2004

Benjamin De Leeuw

Departement Zuivere Wiskunde en Computer Algebra
Galglaan 2, B-9000 Gent
bdeleeuw@cage.ugent.be
Gent University, Belgium

March 10, 2004

1 Regelingen

woe	10 maart 2004	: Inleidende les over de projectopgave
di	30 maart 2004	: Registratie van de gekozen opties per groep op de website
woe	31 maart 2004	: Bijkomende les voor vragen over de projectopgave
ma	3 mei 2004	: Deadline voor het indienen van het project
di	4 mei 2004	: Begin mondelinge verdedigingen projecten
vrij	14 mei 2004	: Einde mondelinge verdedigingen projecten

Het project wordt gemaakt in groepjes van minimaal twee, en maximaal drie personen. De de administratie van de groepen gebeurt via Claroline.

2 Inleiding

De doelstelling van dit project is tweeledig: in eerste instantie moet een verdere uitwerking en implementatie worden gemaakt in java van de theorie uit de cursus Grafische Technieken. In tweede instantie wordt met dit project een inleiding gegeven op softwarespecificatie in UML (Unified Modeling Language).

De cursus Grafische Technieken bespreekt een aantal wiskundige algoritmen om krommen te tekenen op een computerscherm aan de hand gebruikersinput in één of andere vorm. De gebruiker kan volgende informatie aan de algoritmen doorgeven of manipuleren:

datapunten	punten waardoor de kromme moet lopen
controlepunten	punten die de vorm van de kromme sturen
knopenvector	geeft extra informatie omtrent de vorm van de kromme
raaklijnvectoren	eerste afgeleide in een datapunt van de kromme
kromming	tweede afgeleide in een datapunt van de kromme
graad/orde	de gewenste graad van de kromme
segmenten	het aantal segmenten in de kromme

Het is de bedoeling dat de gebruiker op een intuitieve manier krommen kan tekenen op een computerscherm, gebruikmakend van deze inputgegevens. Een kromme is een koppel

$$\mathbf{P}(t) = (x(t), y(t)) \qquad t \in \mathbb{R}$$

We beschouwen nu de belangrijkste manieren, besproken in de theoriecursus, om een tweedimensionale kromme op een scherm te tekenen. Het zijn de achterliggende algoritmen en de gebruikersinterface voor de inputgegevens, die geïmplementeerd zullen worden. Niet elk algoritme gebruikt alles van de bovengenoemde informatie:

Neville	datapunten, raaklijnvectoren, knopenvector
B(asis)-spline	controlepunten, knopenvector, orde
Bézier	controlepunten, knopenvector, orde

Alle algoritmen, behalve Neville, gebruiken mengfuncties en een triangulaire berekening voor afgeleiden. Het gebruik van generieke code voor al deze elementen is dus aangewezen in een object georiënteerd ontwerp.

De algoritmen zullen gaandeweg aan bod komen in de theoriecursus. Het is de bedoeling dat men zo vroeg mogelijk in het project de algemene structuur aanlegt, en later, wanneer de benodigde stof is gezien in de theoriecursus, de algoritmische functionaliteit verder aanvult.

De projectopgave is deels in de taal UML geformuleerd. In UML-diagrammen wordt aangegeven hoe de klassen van een systeem er moeten uitzien en wat hun verbanden zijn met andere klassen. Een deel van de opgave bestaat erin om de opgegeven specificatie in UML te gebruiken als geraamte voor de implementatie. Een inleiding op UML kan bekomen worden op Claroline.

3 Specificatie

We bespreken nu de specificatie van de projectopgave. Om de specificatie in UML te begrijpen, zal eerst de semantiek van de verschillende diagramelementen begrepen moeten worden. Er is een inleidend document op Claroline beschikbaar dat de meest courante elementen van de UML taal bespreekt. Dit document geeft voldoende informatie om de semantiek van de specificatie te begrijpen.

We stellen als vereiste voor een geldig project dat de hieronder besproken klassenstructuur terug te vinden is in de javacode. Dit impliceert dat aanpassingen aan de specificatie enkel toevoegingen of verfijningen mogen zijn. We veronderstellen ook dat dezelfde namen voor klassen gebruikt worden, zodanig dat de overeenkomsten tussen de javacode en de UML specificatie duidelijk zijn, zonder bijkomende uitleg (in een verslag).

Uitzondering op deze regel zijn de klassen die als stereotype *java* hebben. Dit zijn typisch klassen die gebruikt worden bij het maken van grafische interfaces. Welke klassen van java hiervoor gebruikt worden mag vrij gekozen worden, zolang ze maar voldoen aan de betekenis die deze met *java* aangeduide klassen hebben in de diagrammen. Ook kunnen deze klassen samengesteld worden uit meerdere javaklassen. Zelfs de naamgeving hoeft niet overeen te komen, zolang het verband duidelijk blijft.

In- en output voor het systeem $ProjectGT$ worden aangegeven met de stereotypes *Input* en *Output*. Het stereotype CPU geeft aan waar de hoofdrekeneenheid van het systeem zich bevindt.

Wanneer een zelfde naam wordt gebruikt voor klassen in verschillende diagrammen, of in hetzelfde diagram, duiden deze dezelfde klasse aan.

3.1 UML-diagrammen

3.1.1 Hoofdklasse

In figuur 1 zien we een klasse $ProjectGT$, die een aantal deelklassen bevat. De klasse $ProjectGT$ is de hoofdklasse en het besturingssysteem zal deze klasse moeten opstarten (de $main()$-methode

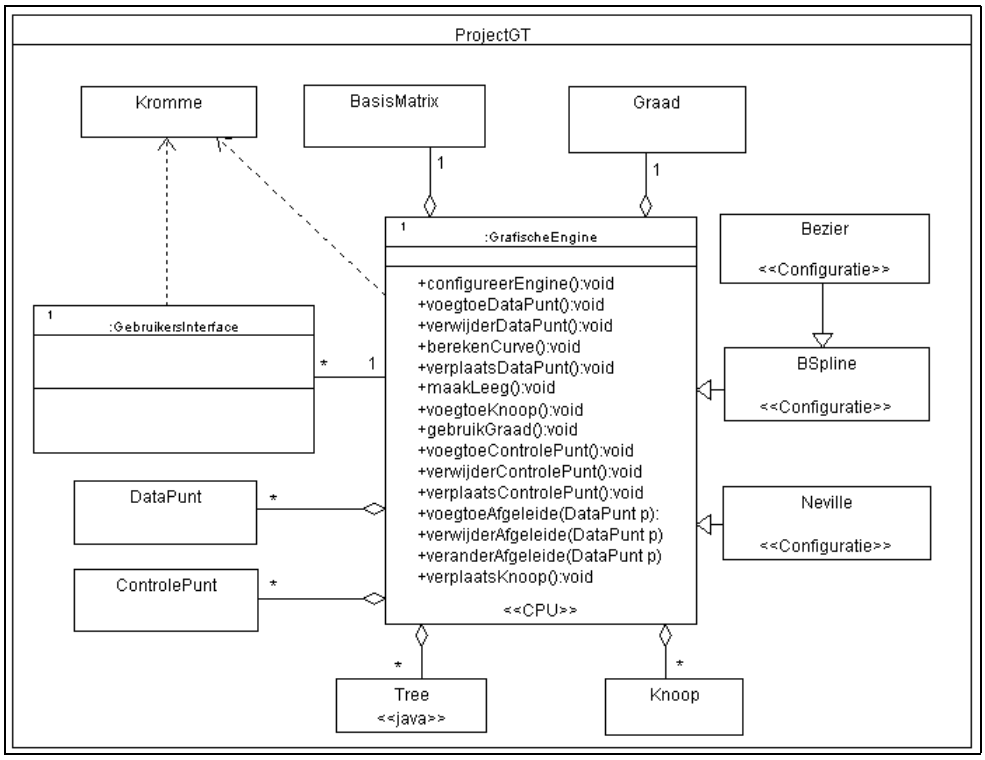

Figure 1: UML-diagram van de Hoofdklasse

kan zich in *ProjectGT* bevinden, maar wanneer de automatisch gegenereerde code gebruikt wordt, kan de klasse *GTComponent* gebruikt worden als *main*()-klasse).

De grafische algoritmen uit de cursus worden ondergebracht in één van de afgeleide klassen van $GrafischeEngine$. Deze afgeleide klassen berekenen de juiste kromme op basis van de door de gebruiker opgegeven input en het respectieve algoritme uit de cursus. We bekijken de $GrafischeEngine$ als functionerend als één van deze configuraties (of afgeleide klassen). Hoe het systeem bijhoudt in welke configuratie de $GrafischeEngine$ functioneert mag vrij gekozen worden. De klasse $GrafischeEngine$ zelf implementeert enkel de behandeling van input (toevoegen, verwijderen, veranderen, ...), en kan eventueel een naïeve implementatie bevatten van *berekenCurve*(). De afgeleide klassen overschrijven vnl de methode *berekenCurve*(). De attributen *Graad*, *ControlePunt*, *DataPunt* worden gebruikt indien nodig voor het algoritme.

De klasse $GebruikersInterface$ roept methodes op van de klasse $GrafischeEngine$, of andersom. De antwoorden (de gegevens om een kromme op het scherm te zetten, klasse *Kromme*) worden door de $GebruikersInterface$ gebruikt om in het $OutputVenster$ de juiste kromme te tonen. Er kan vrij gekozen worden welke van de klassen ($GebruikersInterface$ of $GrafischeEngine$) de communicatie initiëert. De informatie die de gebruiker heeft ingegeven in $GebruikersInterface$ en de informatie waarop de $GrafischeEngine$ zijn berekeningen baseert moeten wel overeenstemmen op elk moment dat een krommeberekening wordt uitgevoerd. Een $GrafischeEngine$ kan in contact staan met meerdere $GebruikersInterface$-objecten.

Let erop dat bij het verplaatsen van data- of controlepunten de gewichtsfuncties *niet* herberekend worden. Let er ook op dat bij het toevoegen van knopen en data- of controlepunten, de oorspronkelijke vorm van de kromme zo veel mogelijk bewaard blijft. Maak de besturing logisch en intuitief voor de gebruiker.

3.1.2 Gebruikersinterface

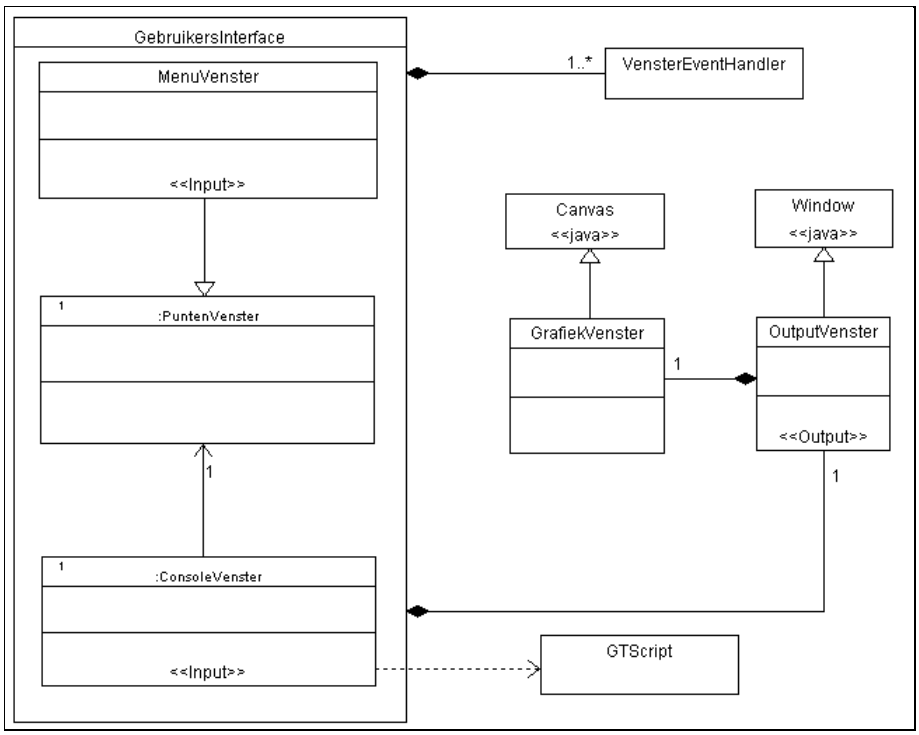

Figure 2: UML-diagram van de Gebruikersinterface en Verbanden

In figuur 2 zien we de deelklassen die tot de $GebruikersInterface$-klasse behoren. Het centrale element hier is het $PuntenVenster$. Deze structuur verzamelt informatie van de gebruiker. Deze gebruiker heeft de mogelijkheid om hiervoor zowel een GUI (met menu) of een console (tekstgebaseerd) te gebruiken.

In figuur 2 zien we ook een klasse $VensterEventHandler$. Dit is een $Listener$-object samengesteld uit javaklassen, dat zorgt voor de aansturing van de vensters van de $GebruikersInterface$. Er mag vrij gekozen worden voor een aanstuurmechanisme, zolang de resulterende klasse maar de aansturing levert voor de objecten die in het diagram worden aangeduid als $Window$. Elk venster dat wordt gebruikt moet vergroot, verkleind, geminimaliseerd, en gesloten kunnen worden. Let ook op de naamgeving. Men mag vrij kiezen wat er gebeurt met het $OutputVenster$ wanneer één van de grafische interface vensters wordt afgesloten.

We zien in figuur 2 een dependency-relatie tussen $ConsoleVenster$ en $GTScript$. Dit betekent dat $ConsoleVenster$ moet weten wat $GTScript$-objecten zijn, en ermee moet kunnen werken. $GTScript$-objecten zijn sequenties van consolecommando's.

3.1.3 Consolevenster

In figuur 3 zien we hoe het consolevenster moet functioneren. De commando's die het consolevenster zou moeten ondersteunen zijn gegeven als methodes van de klasse $ConsoleVenster$. Er worden geen veronderstellingen gemaakt over de signatuur van deze methodes. Deze signatuur mag dus vrij gekozen worden, en zal oa afhankelijk zijn van de gebruikte datastructuren.

In het algemeen zullen deze methodes, die worden opgeroepen door de gebruiker, door het gepaste commando in de console in te geven, ook als menu-opties in het $MenuVenster$ aanwezig zijn. De betekenis van de verschillende methodes wordt gegeven in tabel 2.

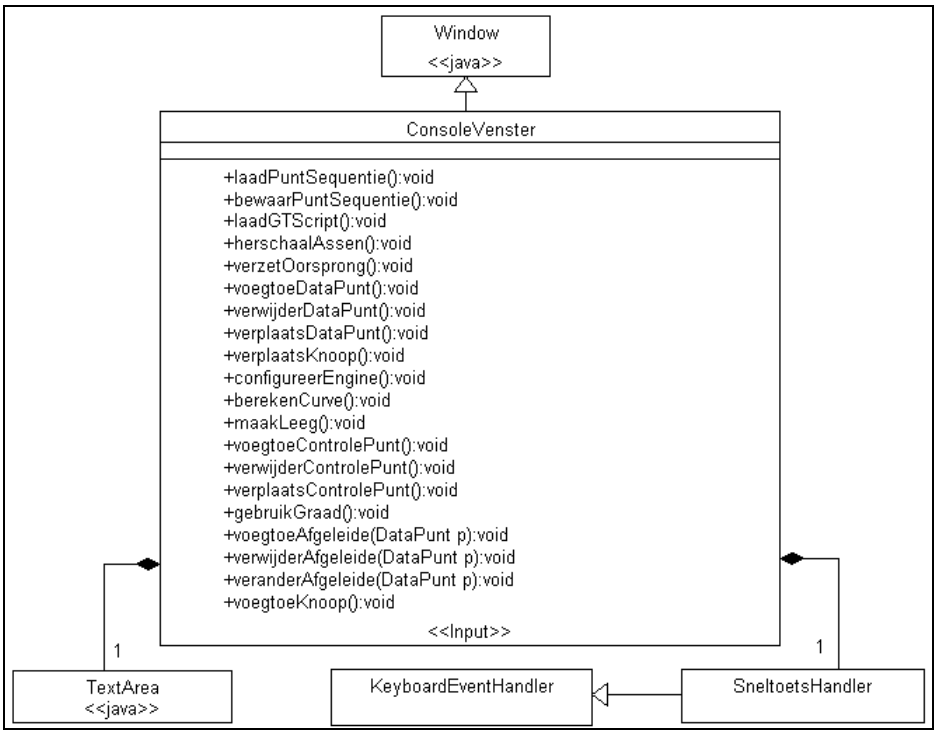

Figure 3: UML-diagram van de Gebruikersinterface en Verbanden

De commando's worden ingegeven in een *TextArea*-klasse. Er wordt ook een soort *Listener* gebruikt voor deze *TextArea*-klasse, en deze *Listener* moet ook eventuele sneltoetsen voor commando's aanvaarden. De sneltoetsen worden gegeven in tabel 1. In het consolevenster zal een geldige toestencombinatie tot gevolg hebben dat het commando automatisch verschijnt in de console. Extra informatie (zoals bestandsnamen) kan nadien bijgetypt worden. Wanneer op *enter* wordt gedrukt zal het commando uitgevoerd worden.

3.1.4 GUI-venster

De klasse $MenuVenster$ is afgeleid van $PuntenVenster$. Het is in dit $PuntenVenster$ dat alle informatie over posities en toestanden van datapunten en controlepunten wordt verzameld.

De datapunten, controlepunten en de lijnen die de datapunten verbinden, worden getoond in het $TekenVenster$, dat een deel is van $PuntenVenster$. Het $TekenVenster$ heeft ook een assenstelsel dat vnl gebruikt zal worden voor vergroting en verplaatsing in het $TekenVenster$ en $OutputVenster$. Indien meerdere datapunten, controlepunten of knopen op dezelfde locatie liggen, moet dit worden aangegeven in het $TekenVenster$, respectievelijk $KnopenManipulatieVenster$. Duid datapunten en controlepunten aan op een verschillende manier.

Er moet ook een menu geïmplementeerd worden in de klasse $MenuVenster$ die minstens dezelfde commando's ondersteunt als de $ConsoleVenster$-klasse, behalve de knopenmanipulatiecommando's, welke ondersteund worden in het $KnopenManipulatieVenster$. De commando's die met de muis worden gegeven staan in tabel 3. De sneltoesten moeten ook ondersteund worden in de $MenuVenster$-klasse en zijn dezelfde als in tabel 1. Extra informatie (zoals bestandsnamen) kan opgevraagd worden met extra dialoogvensters. De functies die met de puntplaatsing te maken hebben kunnen de huidige muispositie gebruiken (dus *tab* komt ongeveer overeen met linker muisknop).

laadPuntsequentie	crtl + l
bewaarPuntsequentie	ctrl + s
laadGTScript	ctrl + g
herschaalAssen	alt + z
verzetOorsprong	alt + o
Punt = Controle of Data	capslock
voegtoeDataPunt	tab
verwijderDataPunt	ctrl + tab
verplaatsDataPunt	shift + tab
voegtoeControlePunt	tab
verwijderControlePunt	ctrl + tab
verplaatsControlePunt	shift + tab
voegtoeAfgeleide	a
verwijderAfgeleide	ctrl + a
veplaatsAfgeleide	shift + a
voegtoeKnoop	k
verplaatsKnoop	shift + k
gebruikGraad	ctrl + d
configureerEngine	F1, F2, F3, F4, ...
berekenCurve	F11
maakLeeg	F12

Table 1: Tabel van Sneltoetsen voor de Console

3.2 Extra Opties

We bespreken nu een aantal uitbreidingen op de specificatie zoals hierboven gegeven. De functie van deze uitbreidingen is tweeledig: enerzijds kan men door opties te implementeren extra punten en verdienen, anderzijds zal de combinatie van opties de authenticiteit van het project bepalen.

Er zijn 17 extra opties, de speciale opties (SPECx) niet meegerekend. Hieruit moeten vier opties gekozen worden. Vermits

$$C_4^{17} >> |projecten|$$

verwachten we geen twee projecten die exact dezelfde extra opties implementeren, hoewel we ons bewust zijn van het feit dat bepaalde opties frequenter gekozen zullen worden dan andere. Indien zulk een identieke copie van opties toch voorkomt wordt de authenticiteit van *beide* projecten in twijfel getrokken. We gaan dan na welke SPECx opties nog werden geïmplementeerd. Indien nog steeds geen verschil te bemerken is, krijgt men het aantal punten voor één van beide projecten, gedeeld door twee (of meer naargelang het aantal identieke projecten). Projecten die een deelverzameling van opties implementeren van een ander project worden ook in twijfel getrokken qua authenticiteit. Er zal dan weer gekeken worden naar de SPECx opties. Indien geen verschil kan worden vastgesteld, zal het project voor maximaal de helft van de score van het andere project gequoteerd worden. Om verrassingen te vermijden wordt vóór de tweede les die bij dit project hoort, via Claroline een lijst opgesteld van $(groep, opties)$ paren. Projecten met dezelfde optiekeuzes kunnen dan in de blijkomende les over dit project aangepast en twisten besproken worden.

laadPuntsequentie	laad een voordien ingegeven input en eventueel configuratie in
bewaarPuntsequentie	sla een voordien ingegeven input en eventueel configuratie op
laadGTScript	laad een sequentie van commando's in die dan in volgorde worden uitgevoerd
herschaalAssen	herschaal het assenstelstel van het *PuntenVenster* en *OutputVenster* zodanig dat een vergroting (zoom) wordt bekomen
verzetOorsprong	verplaats de figuur in het *PuntenVenster* en *OutputVenster*
voegtoeDataPunt	voeg een datapunt toe aan de huidige input
verwijderDataPunt	verwijder een datapunt uit de huidige input
verplaatsDataPunt	verplaats een datapunt in de huidige input
voegtoeControlePunt	voeg een controlepunt toe aan de huidige input
verwijderControlePunt	verwijder een controlepunt uit de huidige input
verplaatsControlePunt	verplaats een controlepunt in de huidige input
voegtoeAfgeleide	voeg een afgeleide of raaklijnvector toe aan het meegegeven datapunt
verwijderAfgeleide	verwijder de afgeleide of raaklijnvector van het meegegeven datapunt
verplaatsAfgeleide	verander de grootte van een raaklijnvector voor het meegegeven datapunt
voegtoeKnoop	voeg een knoop toe aan de huidige input
verplaatsKnoop	verplaats een knoop in de huidige input
gebruikGraad	geef de graad of orde op, gebruikt in de berekening bij B-splines
configureerEngine	verander van configuratie
berekenCurve	geef het commando om te beginnen berekenen door aan de *GrafischeEngine*
maakLeeg	verwijder alle punten en begin opnieuw

Table 2: Tabel van Commando's

3.2.1 Uitbreidingen op de GUI (GUIx)

1. Definieer een extra venster in $OutputVenster$ dat de gewichtsfuncties toont. Deze functionaliteit moet aan- en afgezet kunnen worden, en gewichtsfuncties mogen niet berekend worden indien de optie afstaat.

2. Realiseer een omzetting van het $OutputVenster$ naar een of meerdere figuurbestandstypes (zoals .jpg, .gif, . tiff, .eps, ...). Deze optie moet zowel in $MenuVenster$ als $ConsoleVenster$ toegankelijk zijn.

3. Voorzie een recorder functionaliteit, waarbij de bewegingen en de commando's van de gebruiken worden onthouden als $GTScript$.

4. Bedenk en implementeer een manier om $GTScripts$ af te spelen als animatie en realiseer een conversie van $GTScript$-bestanden naar een videobestandstype (zoals .avi, .mpg, .ram, .mp4, ...). Denk in deze context vooral aan de door java ondersteunde QuickTime libraries.

5. Voorzie een mogelijkheid om het $OutputVenster$ te configuren: positie, kleur, grootte,

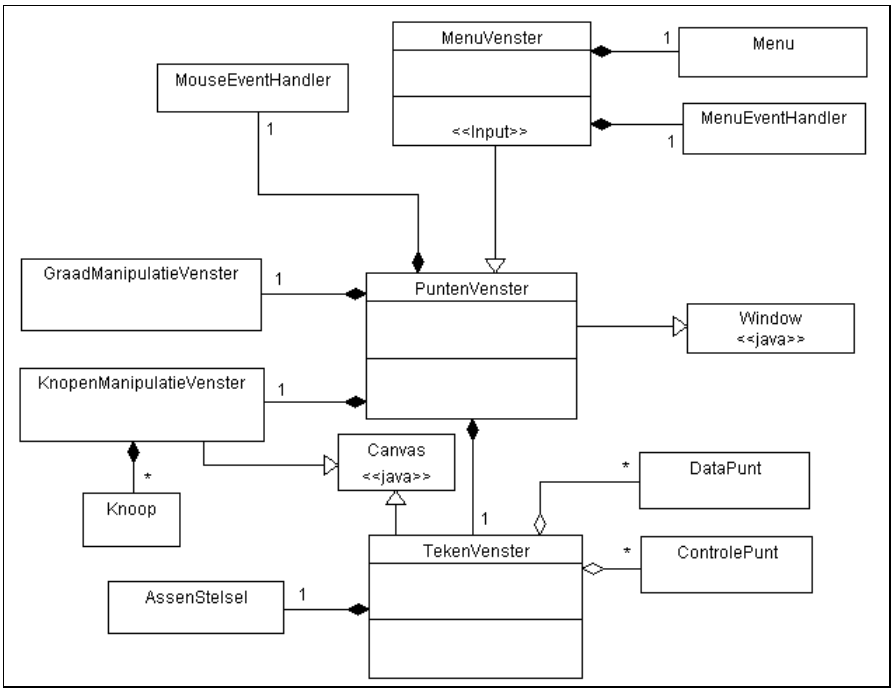

Figure 4: UML-diagram van het GUIVenster

Punt = Controle of Data	capslock
voegtoePunt	linker muisknop
verwijderPunt	rechter muisknop
verzetPunt	shift + linker muisknop

Table 3: Tabel van Muiscommando's

inhoud, aanwezigheid van hulplijnen, data- en controlepunten, ... Voorzie ook een manier om dit op te slaan als voorkeursinstellingen.

6. Voorzie een mogelijkheid om de $GebruikersInterface$ aan te passen: sneltoetsen instelling, muisbesturing, kleur, grootte, kwaliteit van de output bij slepen, ...

7. Geef de gebruiker de mogelijkheid om in het $ConsoleVenster$ de berekeningen te bekijken die worden uitgevoerd door de $GrafischeEngine$. Deze mogelijkheid moet ook kunnen worden afgezet.

3.2.2 Uitbreidingen op de GTScripttaal (GTSx)

1. Bedenk hoe men nuttig variabelen zou kunnen gebruiken in een $GTScript$ en implementeer het concept 'variabele' in $GTScript$. Implementeer lusconstructies in het scriptformaat $GTScript$. Dit wil zeggen dat een $GTScript$, naast sequenties van acties, ook lussen kan bevatten rond die acties, en dat acties in de lus gebruik kunnen maken van de lusteller. Denk in deze context ook aan de BNF formulering van je scripttaal (zie cursus Formele Logica).

2. Bedenk hoe men nuttig variabelen zou kunnen gebruiken in een $GTScript$ en implementeer

het concept 'variabele' in *GTScript*. Implementeer ook if-then-else contructies, met condities op deze variabelen. Maak het niet te ingewikkeld, want dit is veruit de moeilijkste optie. Denk in deze context ook aan de BNF formulering van je scripttaal (zie cursus Formele Logica).

3.2.3 Uitbreidingen op de GrafischeEngine (GENx)

1. Voorzie een AND-configuratie waarmee $GrafischeEngine$-configuraties tegelijk actief kunnen zijn. Het $OutputVenster$ zal dan ook verschillende krommen tekenen op één scherm en met één datapuntensequentie of één controlepuntensequentie. Duid de snijpunten tussen de verschillende krommen aan indien deze er zijn. Verklaar in het verslag de bekomen resultaten.

2. Voorzie een mogelijkheid om de gewichtenmatrix, ook basismatrix genoemd, te tonen op het scherm. Deze functionaliteit moet aan- en afgezet kunnen worden. Maak hiervoor ook een extra commando voor de console.

3. Voorzie de mogelijkheid om verschillende krommen (met verschillende inputsequenties) op één scherm te tekenen, gebruikmakend van dezelfde configuratie van $GrafischeEngine$. Bedenk dus ook een manier om meerdere inputsequenties in te voeren op één scherm. Duid de snijpunten tussen de verschillende krommen aan indien deze er zijn. Verklaar in het verslag de bekomen resultaten.

4. Voorzie de mogelijkheid om verschillende krommen (met verschillende inputsequenties) op één scherm te tekenen, gebruikmakend van verschillende configuraties van $GrafischeEngine$. Duid de snijpunten tussen de verschillende krommen aan indien deze er zijn. Verklaar in het verslag de bekomen resultaten.

5. Bedenk aan de hand van de theoriecursus wat er nog kan gedaan worden met een invoer van sequenties van datapunten, raaklijnvectoren en controlepunten. Implementeer dit als een extra configuratie voor $GrafischeEngine$.

3.2.4 Algemene Uitbreidingen (AUIx)

1. Voorzie een commandline versie van het programma, dat een sequentie data- of controlepunten als parameters meekrijgt, en als output een venster op het scherm toont met de relevante kromme. Dit mag geen apart programma zijn, en het is de aan- of afwezigheid van argumenten bij opstarten dat bepaalt welke versie er wordt opgestart. Geef duidelijk aan wat de syntaxis van deze commandline versie is in het verslag.

2. Spit de java grafische routines verder uit en zoek of er bruikbare methodes of klassen zijn, die beter zouden kunnen gebruikt worden in dit project, in de plaats van de eigen methodes van $GrafischeEngine$. Bespreek moeilijkheden en mogelijkheden in het verslag.

3. Voorzie een professioneel installatiebestand dat nagaat welke resources er reeds aanwezig zijn op de PC (Java Virtual Machine bvb) en al het nodige installeert, op een locatie die de gebruiker kan specificeren. Let wel, *alle* resources die niet aanwezig zijn moeten automatisch geïnstalleerd worden.

3.2.5 Speciale Uitbreidingen (SPECx)

1. Voorzie de mogelijkheid om naast de afgeleiden in een datapunt, ook de kromming (tweede afgeleide) weer te geven en te manipuleren. Voorzie hiervoor drie nieuwe functies voor $GrafischeEngine$:

   voegtoeTweedeAfgeleide(DataPunt p)
   verwijderTweedeAfgeleide(DataPunt p)
   verplaatsTweedeAfgeleide(DataPunt p)

   Maak overeenkomstige commando's voor de console en de menu en voorzie eventueel sneltoetsen.

2. Bedenk en implementeer een manier om met de Bézierrepresentatie van B-Splines te werken in het programma.

4 Implementatie

De UML-specificatie wordt geïmplementeerd in de programmeertaal java. Er wordt een java klassenskelet beschikbaar gesteld, gegroepeerd in een package $GrafischeTechnieken$, dat automatisch werd gegenereerd ahv de besproken UML-diagrammen. De opstartklasse is in dit geval $GTComponent$. Dit skelet hoeft echter niet gebruikt te worden. Er wordt wel verondersteld dat de klassenamen en klassenstructuur overeenstemt met de opgegeven specificatie.

De code mag hier en daar becommentariëerd worden, maar men hoeft geen ontwerpbeslissingen, of uitleg over ontwerpbeslissingen in de code toe te voegen. De commentaar in de code wordt bij voorkeur gebruikt om te refereren naar de UML-diagrammen in het verslag. In dit verslag kunnen mogelijke onduidelijkheden worden uiteengezet.

5 Verslag

Het verslag begint met een duidelijke vermelding van de groepsleden en de gekozen opties.

In het verslag worden vervolgens de volledige UML-diagrammen weergegeven (attributen, methodes met volledige signatuur, verbanden met andere klassen, sterkte van deze verbanden, ...), die overeenkomen met het geïmplementeerde project. Verfijningen en toevoegingen worden ook getoond ahv UML-diagrammen. Er mogen geen nieuwe diagrammen worden bijgemaakt, elke toevoeging of verfijning moet kunnen aangegeven worden in de opgegeven vier diagrammen. Een bruikbaar programma om deze diagrammen aan te maken is DIA.

Bijkomende verfijningen voor de opties worden ook in de UML-diagrammen aangegeven. Bespreek kort de belangrijkste ontwerpbeslissingen omtrent deze opties.

Indien de overeenkomsten tussen UML-diagrammen en javacode niet duidelijk genoeg lijken, mag men ook hier een woordje uitleg over geven.

6 Presentatie

Op de presentatie krijgt men maximaal tien minuten de tijd om het project voor te stellen én te demonstreren. Een presentatie met slides mag, maar is niet verplicht. Een werkend programma tonen is wel verplicht. Er wordt ook aangegeven wie wat gedaan heeft voor het project.

De overige tien minuten zullen vragen worden gesteld aan de leden van de groep over het project en de gekozen opties. Dit zijn individuele vragen: degene die de vraag krijgt, beantwoordt de vraag, er wordt niet overlegd met groepsleden. Door de vragen goed of slecht te

Basisdiagrammen 1-4 met Neville	6 ...
... en B-splines	+4
... en B-splines en Bézier	+6
Opties SPECx	...
... 1 optie	+2
... 2 opties	+4
Opties GUIx GTSx GENx AUIx	...
... 1 optie	+0.5
... 2 opties	+1.5
... 3 opties	+2.7
... 4 opties	+4
Mondelinge verdediging	individueel ...
... niet opdagen	-3
... vlotte presentatie	+1
... per goed beantwoorde vraag (max 3)	+1

Table 4: Tabel voor Quotering

beantwoorden kan de individuele eindscore worden aangepast, niet het totaalresultaat van de groep.

7 Quotering

We geven in tabel 4 weer welke punten er te verdienen en te verliezen zijn. Er zijn geen andere mogelijkheden om punten te verdienen. Voor andere combinaties van configuraties zal de grootste deelverzameling uit tabel 4 genomen worden, die aanleiding geeft tot het hoogste punt. Niemand kan meer dan 20 punten halen, een eventuele overschot gaat verloren. De punten worden direct na de presentatie bekendgemaakt.