CSD2d - Keuzeopdrachten

Hieronder een beknopt overzicht. Uitgebreide omschrijvingen staan verderop. Je kunt kiezen om het hele blok (6 weken) aan één opdracht te werken of twee opdrachten te kiezen waarbij je drie weken aan elke opdracht werkt.

Opdracht-titel	**Tags**
ADSR envelope (C++)	C++, synthese, envelope, systeemontwerp, UML class diagram, design patterns, observer pattern
Playful muzikale web/mobile-app/native	web/mobile/native, muziek generatie, prototype, interactie ontwerp, playful interactie
BASH script voor klankinstallatie, beschrijving op https://csd.hku.nl	UNIX/BASH shell scripting, sox, automatisering, muziekgeneratie a.d.h.v. samples
Stand-alone looper	Raspberry Pi/Bela, muzikale loops, één-knops-interactie
Plugin m.b.v. JUCE	C++, dsp/synthese/analyse, plugin, JUCE, werken met een framework, interactie-ontwerp
Web game met pitch-aansturing	web, webaudio, werken met een library, onderzoek en verslaglegging, PoC[1]/prototype, interactie ontwerp, gameplay ontwerp
Realtime visuals bij muziek	realtime visuals generatie, toepassing ontwerp, openFrameworks/openCV, MIR libraries, mapping, m.v.p. PoC/prototype
Draaiorgel met C++ Standard Template Library (STL)	Draaiorgel, STL, list, map, queue, modulair werken, niet-lineaire compositie, datastructuren en algoritmen

[1] PoC: Proof of Concept

Keuze-opdrachten (uitgebreide omschrijving)

ADSR Envelope

Omschrijving

In deze opdracht maak je een eenvoudige synthesizer die een Envelope class bevat en waarbij tonen aan- en uitgezet kunnen worden. De nadruk ligt hierbij op modulair werken in C++ waarbij je goed nadenkt over welke onderdelen waar en op welke wijze moeten worden ondergebracht. Zo kun je bijvoorbeeld de ADSR waarden los toevoegen aan de Envelope class, maar een struct hiervoor gebruiken is netter.

Stappen

1. Maak een class diagram in UML met daarin i.i.g. de volgende classes:
    - Generator, bevat (o.a.) de method getSample() en de virtual method tick()
    - Oscillator - erft van generator
    - Sine - erft van oscillator (of andere oscillatoren zoals Saw en Square)
    - Envelope - erft van generator, bevat ADSR functionaliteit
    - Synthesizer - erft van generator, bevat minimaal één Sine/Saw/Square/… en een envelope voor de amplitude. De synthesizer is aan te sturen met noteOn(...) en noteOff(...) functies. Deze Synthesizer class vormt een monofone synth. Je mag deze polyfoon maken door de class Voice toe te voegen, die de Envelope en Oscillator subclass beheert.

2. Implementeer alle classes uit je class diagram in C++.
   (Hierbij kun je starten from scratch, maar je mag ook delen van jouw uitwerking voor project 2b gebruiken als startpunt.)

3. Je gaat nu het Observer pattern toepassen in je project. Bestudeer eerst dit design pattern voordat je verder gaat.
4. Maak een Clock class die een lijst van observers kan bijhouden, deze class vormt de *subject* van het Observer pattern. Voeg aan deze class een tick method toe, waarin de tick method van alle observers die zijn geregistreerd (de observer objecten in de observer lijst van de Clock class, zie Observer pattern) worden aangeroepen.
   Aan de constructor van de Generator class (dus ook aan elke andere classes die daarvan erven) geef je een Clock object mee. Een Generator registreert zich vervolgens bij de Clock class als ‘observer’. De Generator vormt nu de *observer* van het Observer pattern.
   In je project maak je één object aan van de Clock class die je aan alle generatoren (objecten van subclasses van Generator) meegeeft. Vervolgens kun je de clock.tick() method aanroepen in plaats van de tick method van al je Generator subclass objecten los aan te roepen.
   Tot slot, herzie je UML class diagram.

5. Maak een beknopte demo (command-line) app om de werking van je Synthesizer te tonen.

Voorwaarden

- uitwerking in C++
- het systeem is gebouwd volgens de OOP richtlijnen

- er is een UML class diagram gemaakt van alle classes
- de Envelope class is uitgewerkt als een ADSR envelope
- het Observer pattern is geïmplementeerd

Deliverables
- code op github
- een videopresentatie waarin de werking van de demo applicatie wordt getoond (maximaal 3 minuten).
- een document met daarin:
    - UML class diagram
    - Omschrijving jouw implementatie van de Envelope class; welke ontwerpkeuzes heb je gemaakt?
    - Uitleg van het observer pattern in eigen woorden.
    - Een beschrijving van de wijze waarop het observer pattern in het project is verwerkt.

Tags
c++, synthese, envelope, systeemontwerp, UML class diagram, design patterns, observer pattern

Playful muzikale web/mobile/native-app maken

Ontwerp een ‘playful’ muzikale toepassing die de gebruiker op een speelse manier uitnodigt om muziek te maken. Dit ontwerp vertaal je vervolgens naar een prototype. Welke elementen zijn essentieel en kunnen in een prototype getest worden?

Vaste knoppen op het scherm waarmee je samples kunt bespelen zijn niet perse een playful interactie, maar wat gebeurt er wanneer de samples (aangeduid met een kleur van de knop) van knop wisselen? Of hoe verandert de interactie wanneer de knoppen bewegen over het scherm?

Tijdens de ontwerpfase van deze opdracht zijn onder andere de volgende vragen relevant.
- Wanneer is een muzikale interactie playful/speels?
- Hoe blijft een interactie playful én uitdagend op de lange termijn?
- Wordt de muziek direct getriggerd / aangepast, of gebeurt dit geleidelijk en minder opvallend?
- Voor wie is de app bedoeld?
- Wanneer is het klinkende resultaat muzikaal interessant (denk hierbij ook aan je doelgroep)?

Na het ontwerpen van de interactie van de app, selecteer je de essentiële onderdelen die je vervolgens uitwerkt in een eenvoudig prototype. Hierbij ga je pragmatisch te werk, zo is visuele vormgeving voor jouw prototype misschien minder relevant dan de audio output. Je mag gebruik maken van bestaande libraries of omgevingen, zoals bijvoorbeeld openFrameworks, P5js of Unity.

Voorwaarden
- Er is bewust gekozen voor één platform (web/mobile/native).
- Het ontwerp is afgesteld op het gekozen platform (bijvoorbeeld touch of tilt interactie in een mobile app).
- Het interactie ontwerp richt zich op een playful interactie.

- De essentie van het interactie ontwerp is verwerkt tot een speelbaar én klinkend prototype.

Deliverables
- code van het prototype op github
- een videopresentatie waarin, na een korte beschrijving van het ontwerp, de werking van het prototype wordt getoond (maximaal 3 minuten)
- een document met daarin:
    - het interactie ontwerp in de vorm van een beschrijving waarin onder andere de antwoorden op de bovenstaande vragen zijn verwerkt, aangevuld met visueel materiaal ter ondersteuning
    - een omschrijving van de essentie van de interactie die is uitgewerkt in het prototype

Tags
web/mobile/native, muziek generatie, prototype, interactie ontwerp, playful interactie

Stand-alone looper

Omschrijving
De Raspberry Pi en Bela zijn bij uitstek geschikt om handzame stand-alone apparaatjes te maken. Bij deze opdracht wordt gevraagd een stand-alone looper te maken. Dit is behoorlijk vrij interpreteerbaar. Het geluid kan afkomstig zijn van de Pi/Bela of een externe bron (met Pi/Bela als aansturing) en mag gemaakt worden met synthese, samples of bewerking van input-geluid.

Mogelijke richtingen:
- [audio/MIDI/OSC] irregular beat generator die zelf ritmes ter lengte van één maat produceert en deze aan elkaar plakt tot een loop van meerdere maten. Wanneer de gebruiker op een knop drukt wordt een andere permutatie van maten ge-loopt.
- [audio] loop station: gebruiker speelt/zingt/neuriet een stukje, wat vervolgens ge-loopt wordt. De gebruiker krijgt één knop ter beschikking die het begin en eind van een loop aangeeft

Voorwaarden
- produceert muzikale loops
- de software draait op een Raspberry Pi of Bela
- de gebruiker kan het programma besturen met één knop
- het produceert muzikale loops
- de software is door jou zelf gemaakt
- programmeertalen naar keuze

Deliverables
- stand-alone looper
- video van het gebruik of live-demo via Jitsi
- documentatie: concept, gedetailleerd ontwerp, systeem-diagram, audio flow diagram
- code

Plugin m.b.v. JUCE

Omschrijving
Maak een plugin in C++ met behulp van JUCE. Het plugin type mag je zelf bepalen, hierbij valt te denken aan synthese, een audio effect en een analytische plugin.

Ontwerp de plugin zodanig dat deze uniek is; met jouw plugin kun je 'iets' wat niet kan met bestaande plugins. Durf *out of the Box* te denken, bijvoorbeeld een delay met een continue veranderende delaytijd op basis van de BPM, of een synthesizer waarbij de detune per frequentie verschillend is en bovendien afhangt van de ingestelde toonsoort, een delay waarbij de feedback automatisch groter is wanneer het input signaal 'meer hoog' bevat.

Voorwaarden
- in C++
- met JUCE
- GUI bevat minimaal 3 manieren om de plugin in te stellen.
- een vernieuwend ontwerp

Deliverables
- code op github
- een videopresentatie waarin, na een korte beschrijving van het ontwerp, de werking van de plugin binnen een host (bv DAW) wordt getoond (maximaal 3 minuten)
- een document met daarin:
    - een korte beschrijving van de plugin, waar in ieder geval het ontwerp en de werking aan bod komt (tussen de 200 en 300 woorden)
    - een audio flow diagram
    - een systeem diagram

Tags
c++, dsp/synthese/analyse, plugin, Juce, werken met een library, interactie ontwerp

Web game met pitch aansturing

Omschrijving
Maak een eenvoudige web game waarbij de toonhoogte van een 'live geluid' (zoals zang of gitaar) de (hoofd)aansturing vormt.

Allereerst ga je op zoek naar web audio libraries die je kunt gebruiken voor pitch detectie in de browser. Je legt de gevonden libraries vast in een document waarbij je per library de plus- en minpunten op een rij zet. Aan het einde van dit document verantwoord je je keuze. Eventueel kun je deze verantwoording onderbouwen door twee eenvoudige *proof of concepts* te maken met de twee libraries die het beste lijken (een *proof of concept* (PoC) is een test/prototype met als doel bewijzen dat 'iets' werkt).

Wanneer je een library hebt gekozen gebruik je deze in het tweede deel van deze opdracht. Je ontwerpt en ontwikkelt een prototype van een eenvoudige game in de browser waarbij de toonhoogte van een 'live geluid' (zoals zang of gitaar) de (hoofd)aansturing vormt. Dit prototype is meer dan enkel een

technische PoC, de gameplay in het prototype is bewust ontworpen en geïmplementeerd met als doel om langere tijd de speler te kunnen uitdagen.

Voorwaarden

- met html, css, javascript en webaudio
- gebruik een bestaande library voor pitch detectie
- een web-game met een ontworpen gameplay
- besturing van de game met (hoofdzakelijk) pitch detectie

Deliverables

- code van het prototype op github
- een videopresentatie waarin, na een korte beschrijving van het ontwerp, de werking van het prototype wordt getoond (maximaal 3 minuten)
- een document met daarin:
    - overzicht van de gevonden web audio pitch detectie libraries, per library:
        - een beknopte omschrijving (aantal zinnen)
        - pluspunten
        - minpunten
    - verantwoording van de gekozen library
    - een omschrijving van de game en interactie die is uitgewerkt in het prototype

Tags

web, werken met een library, onderzoek en verslaglegging, PoC/prototype, interactie ontwerp, ontwerp van gameplay

Realtime visuals bij muziek

Omschrijving

In deze opdracht ontwerp je een toepassing waarin realtime visuals worden gegenereerd op basis van muzikale input. De context van de toepassing mag je breed interpreteren, zoals een applicatie, een kunstinstallatie, een live performance en een game. Ook de muzikale input is vrij te kiezen, zoals een muziekstuk dat wordt afgespeeld, realtime input van een instrument en realtime omgevingsgeluid. Vervolgens werk je aan een PoC en eventueel een prototype met daarin de essentie van je ontwerp.

Stappen

1. Allereerst ontwerp je de toepassing die je voor ogen hebt. Durf groot te denken en even los van de technische mogelijkheden. Leg dit ontwerp vast in een document. Omschrijf hierbij het concept van de toepassing en op wat voor soort beeld wordt gegenereerd aan de hand van de input.

2. Welke realtime mapping van muziek naar visuals is nodig in de toepassing die je voor ogen hebt? Stel een lijst op van elk mapping onderdeel (voorbeeld van één onderdeel: “De power van de audio input wordt omgezet in de kleur intensiteit van de visuals”) en voeg deze toe aan je documentatie.

3. Welke mapping is je m.v.p.? Wanneer je dit ontwerp uit zou werken tijdens een langdurig project, welke elementen zouden dan absoluut uitgewerkt moeten worden? Voeg deze afbakening en een beknopte onderbouwing hiervan toe aan je documentatie.

4. Werk de mapping onderdeel/onderdelen in je m.v.p. uit in een proof of concept (PoC) in een platform naar keuze (applicatie, native, web, …). Gebruik bestaande libraries voor de muzikale analyse (MIR) en visuele generatie. Eventueel kun je van de PoC een prototype van je toepassing maken. (PoC = technologisch 'bewijs dat het werkt', een prototype omvat meer dan een PoC, daarin is bijvoorbeeld al een interactie ontwerp verwerkt.)

5. Voeg aan je documentatie een reflectie op je PoC (en eventueel op je prototype) toe. Bijvoorbeeld een PMI reflectie (Pluspunten: Wat werkt goed? Minpunten: Wat werkt niet goed? Interessant: Wat is interessant?). Verwerk aanvullend in deze reflectie de technologische obstakels/beperkingen m.b.t. jouw initiële ontwerp.

<u>Voorwaarden</u>

- M.b.t. het ontwerp:
    - De context van de toepassing is bewust gekozen.
    - De muzikale input is bewust gekozen.
    - De visuals worden in realtime gegenereerd.
    - De mapping onderdelen van het gehele ontwerp zijn verzameld.
    - De m.v.p. afbakening is onderbouwd.
- M.b.t. de PoC:
    - voor zowel de muzikale analyse als de visuele generatie zijn bestaande libraries gebruikt
    - er worden realtime visuals gegenereerd op basis van muzikale input

<u>Deliverables</u>

- code van de PoC/het prototype op github
- een videopresentatie waarin, na een korte beschrijving van de toepassing inclusief context én muzikale input, wordt de werking van de PoC/het prototype getoond (maximaal 3 minuten)
- een document met daarin:
    - het ontwerp, inclusief het concept en de beoogde visuals
    - een lijst van de realtime mapping van muziek naar visuals
    - de m.v.p. van de mapping onderdelen, inclusief beknopte onderbouwing
    - reflectie op de PoC/het prototype met daarin in ieder geval een reflectie op de technologische obstakels/beperkingen m.b.t. het initiële ontwerp

<u>Tags</u>

realtime visuals generatie, toepassing ontwerp, openFrameworks, MIR libraries, mapping, m.v.p. PoC/prototype

STL draaiorgel

Benodigde kennis: C++ Standard Template Library

- https://www.dinkum.nl/software/cplusplus/doc/stl

Opdracht

Uitgangspunt is een draaiorgel met een flexibel boek. De muziek wordt live aangepast op aanwijzingen van de gebruiker en op constante snelheid afgespeeld. De aanpassingen zijn vooraf bedachte modificaties van een basismelodie.

Aanpak

- maak m.b.v. een STL-map een bibliotheek met melodieën die ieder één maat vullen
- elke melodie is een STL-list
- genereer de melodieën door vanuit een basis-melodie op willekeurige plaatsen één noot tussen te voegen en één noot te vervangen
- wanneer het programma draait kan de gebruiker door namen van de melodieën te typen het verloop van het afspelen bepalen
- zet melodieën stuk voor stuk in een STL-queue door ze aan de hand van hun naam uit de map op te halen
- aan de andere kant van de queue wordt elke melodie afgespeeld
- als de queue leeg is wordt de vorige melodie blijvend herhaald tot er weer iets in de queue staat

Aanvullende voorwaarden

- Programmeer in C++
- Werk modulair: maak een ontwerp en class-diagram waarin je de classes voor de onderdelen aangeeft en hun onderlinge relaties
- Gebruik STL containers, iterators en algoritmes
- Integreer Git in je workflow: zet bij elke significante wijziging je code in git met zinvolle commit message

Deliverables

- Robuust werkend programma
- Beschrijving van de opbouw (ontwerp-doc) met class-diagram
- Code via Git

Optioneel

- Geef de gebruiker de mogelijkheid om aan te geven om een melodie achterstevoren of een andere permutatie te gebruiken
- Laat net als bij een draaiorgel kleine variaties in afspeelsnelheid voorkomen