Reflectie CSD2b SynthSong eindopdracht – Fabian Dekker

Inhoud

Wat ik heb geleerd

1.1 Werken in C++

Ik ben ondertussen zeer comfortabel geworden met werken in C++. Ik kan goed object oriented programming toepassen en ik kan gemakkelijk pseudo code vertalen naar een werkende C++ code. Ook heb ik een goed inzicht verkregen in inheritance en composition. Ik kan goed error messages interpreteren en mijn code debuggen. Ik weet hoe ik voor mezelf overzicht houdt bij projecten met veel verschillende bestanden. Ik heb veel kennis vergaard over hoe de standard library in elkaar steekt en ik heb aardig door hoe typecasting werkt. Ik kan bad-practice voorbeelden aardig herkennen en bedenken wat er misgaat. Ik ben goed geworden in code die geschreven is door andere mensen uitpluizen, doorgronden en toepassen. Ik weet wat ik moet opzoeken om ik kennis die ik mis te vinden. Ik ben goed geworden in documentatie lezen en het lukt aardig om kennis op te doen uit de cppreference. (Het heeft een tijd geduurd voordat dit lukte, voor mij was de cppreference in het begin erg verwarrend).

1.2 Git integratie

Een takeaway die ik had uit het vorige blok was dat ik github meer wilde integreren in mijn workflow. Dit is deels gelukt. Ik ben nog niet bezig geweest met development branches aanmaken, en ik denk dat ik nog niet helemaal het voordeel hiervan begrijp. Ik heb wel het pullen van de CSD2 repository opgenomen in mijn workflow. Dit was heel fijn want zo had ik good-practice voorbeelden altijd bij de hand en kon ik voorbeeld code gemakkelijk aanpassen, compilen en testen.

1.3 Inzicht werkwijze

Ik begon met het grondig uitwerken van de opdrachten in de sessies. Ik merkte alleen halverwege sessie twee dat ik veel tijd verloor aan het formuleren van met antwoorden op de vragen die werden gesteld in de opdracht en dat het voor mij beter werkte om te beginnen met de eindopdracht en dan voor mezelf veel tests te doen, en dan datgeen waar ik tegenaan liep zelf uit te. Verder verschilde mijn werkwijze bij het leren van de syntax behoorlijk ten opzichte van vorig blok. In python kon ik op goed verstand veel dingen zelf uitvogelen (80% schrijven, 20% lezen). Bij C++ probeerde ik dit in eerste instantie ook, maar ik liep al snel vast. Gedurende het blok groeide ik uit mezelf toe naar een drastisch andere werkwijze (20% schrijven, 80% lezen). Ik denk dat hier bij mij twee duidelijke redenen voor zijn aan te wijzen:

1) Voor mij voelt C++ een stuk preciezer/ specifieker dan Python. Ik had tijdens het programmeren veel meer het gevoel dat ik controle had over interne processen (bijvoorbeeld memory management). Hierdoor wilde ik deze processen op een nette manier laten verlopen en efficiënte code voelde veel belangrijker dan bij Python. Dit maakte dat ik meer geneigd was om op te zoeken en te leren van hoe andere mensen omgingen met bepaalde problemen.

2) Over het algemeen ben ik tijdens het programmeren het meeste bezig met voorspellen wat bepaalde code gaat doen. Hierbij maak ik aannames over hoe onderliggende systemen werken. Ik maak deze aannames op basis van kennis die ik vergaard heb, en simpelweg op basis van wat me logisch lijkt. Bij Python was dit altijd wel genoeg, en code die ik had geschreven functioneerde vaak zoals ik had ingeschat, maar bij C++ ging dit absoluut niet op. Het frustreerde me dat ik niet begreep waarom bepaalde code wel of niet werkte. Dit maakte dat ik dit tot op de bodem wilde uitzoeken. Pas toen ik enigszins doorhad hoe het onderliggende systeem werkte had ik het gevoel verder te kunnen met de opdracht.

1.4 Vaardigheid in het opslaan van informatie

Ik had mijn werkwijze dit blok een enorme hoeveelheid kennis om te verwerken, op te slaan en toe te passen. (Ik had op een gegeven moment 228 tabbladen open). Hierdoor heb ik, eigenlijk vanuit noodzaak, veel tijd besteed aan hoe ik kennis die ik vergaar opsla en toegankelijk maak voor mezelf. Ik gebruikte hier eerst een sources.txt bestand voor, maar ging later over naar een soort tree-view overzicht in Obsidian.

Waar ik tegenaan liep

2.1 Reflectie op leerdoel afbakenen scope

Het eerste leerdoel dat ik had gekozen was:
"Het beoogde ontwerp afstellen op de eigen capaciteit, zowel wbt het afbakenen ter voorkoming van overbelasting als opschalen ter voorkoming van onder belasting."
De manier hoe ik dit leerdoel wilde bereiken was eerst het absolute minimale doen voor de opdracht, en vanuit daar, als ik tijd overhad, functies toevoegen en de scope uitbreiden. Dit lukte goed tijdens het ontwerpen, maar ik merkte al snel dat door oppervlakkig te blijven in mijn uitwerking:
- Ik moeite had met begrijpen wat ik aan het doen was.
- Motivatie verloor
- Het simpelweg niet meer zo leuk vond om aan de opdracht te werken.

Pas toen ik me helemaal ging verdiepen in de taal vond ik mijn motivatie terug en heb ik heel veel voortgang geboekt en geleerd in een relatief korte tijd.

2.2 Reflectie op leerdoel timeboxing

Het tweede leerdoel dat ik had gekozen was:
"De time-management techniek timeboxing toepassen bij het onderzoeken / leren van nieuwe kennis en vaardigheden."
Timeboxen is in de eerste helft van het blok goed gelukt, later toen ik de deadline niet haalde en ging doorwerken in de vakantie liet ik het timeboxen los om de opdracht sneller te kunnen afronden. Ik heb in de vakantie meerdere dagen veel te lang achter elkaar gewerkt. Ondanks dat het weer mis ging in de vakantie viel het me op dat het timeboxen me eigenlijk gemakkelijk afging.
Ik denk dat dit aan de ene kant kwam doordat ik eerst ook maar weinig tijd beschikbaar had om aan zelfstudie te doen dit blok. Doordeweeks kon ik vanwege persoonlijke redenen eigenlijk niet aan school werken, en de tijd die overbleef werd al snel opgevuld door deadlines van andere vakken. Uiteindelijk kwam ik maar toe aan 4 uur zelfstudie per week in plaats van 10.
Aan de andere kant denk ik dat het kiezen van dit leerdoel gebaseerd was op een misvatting die ik had aan het begin van dit blok over mijn leerproces en wat ik daarbij nodig heb. Ik dacht dat ik moeite had met mijn tijd inkaderen en dat ik het moeilijk zou vinden om bewuster om te gaan met de tijd die ik spendeerde aan de opdracht, maar wat ik eigenlijk moeilijk vond was genoeg diepgang vinden in de beperkte tijd die er beschikbaar was om te werken aan de opdracht.

2.3 Conclusie leerdoelen

Doordat ik de opdracht voor mezelf had ingekaderd qua scope ging ik automatisch naar diepgang zoeken op een andere plek. In dit geval was dat de beste manier vinden om dingen te doen binnen de kaders die ik had gesteld: generieke herbruikbare templatefuncties, dynamische allocatie, mooie duidelijke user interface, etc.
Zelfstudie die ik moest doen om dit te realiseren zag ik als iets dat ik *'gewoon leuk vond om te doen'*, en beschouwde dit als *'vrije tijd'*. Op die manier had ik voor mezelf een soort loophole gevonden en heb ik mezelf uiteindelijk (ondanks het timeboxen) weer opnieuw overwerkt.
Ik denk dat de twee leerdoelen die ik had gesteld niet samengaan op de manier hoe ik ze heb geprobeerd uit te voeren dit blok. *Kiss principe* **+** *maximaal 4 uur per dag* **=** Ik verzin een loophole en zoek uit noodzaak naar een andere manier van diepgang binnen de opdracht. Een vaste hoeveelheid uur per dag lukt op zich wel, maar als ik de opdracht niet heb ingekaderd op een manier die voor mij werkt, haal ik de deadline niet.

2.4 Werken met aangeleverde code

Ik heb dit blok gemerkt dat ik het lastig vond om te werken met al aangeleverde code toen ik de syntax van de codeertaal nog niet goed doorhad. Ik denk dat dit te maken heeft met datgeen waar ik bij ***'1.3 inzicht werkwijze'*** al enigszins op heb gereflecteerd. Vooral bij het werken met de custom audiocallback liep ik vast doordat er zo veel achterliggende kennis was die ik nog niet begreep. Toen ik wat achtergrondkennis vergaard had en nadat ik Daan een aantal vragen had kunnen stellen over de callback ging het weer heel goed. Volgend blok zal ik eerder naar Daan stappen zodra ik merk dat ik vastloop bij het werken met aangeleverde code.

2.5 Koppigheid

Ik merkte dat ik soms moeilijk een bepaald idee kon loslaten bij het coderen. Ik had gelezen over template conversiefuncties en generieke type-onafhankelijke classes. Ik was hier enthousiast over en ik vond hier precies de diepgang waar ik naar zocht. Ik had constant het gevoel dat ik heel dicht bij het werkend krijgen van de code was, en elke keer dat het dan toch niet werkte voelde ik me uitgedaagd om het dan *'toch wel'* werkend te krijgen.
Dit was de extra werkdruk die het opleverde echt niet waard en ik heb de type-onafhankelijkheid uiteindelijk niet eens echt gebruikt. Ik denk dat ik vanwege het 'loophole' dat ik vond geen duidelijk kader meer had en mezelf hierdoor overwerkte.

Takeaways

3.1 Ingekaderde diepgang

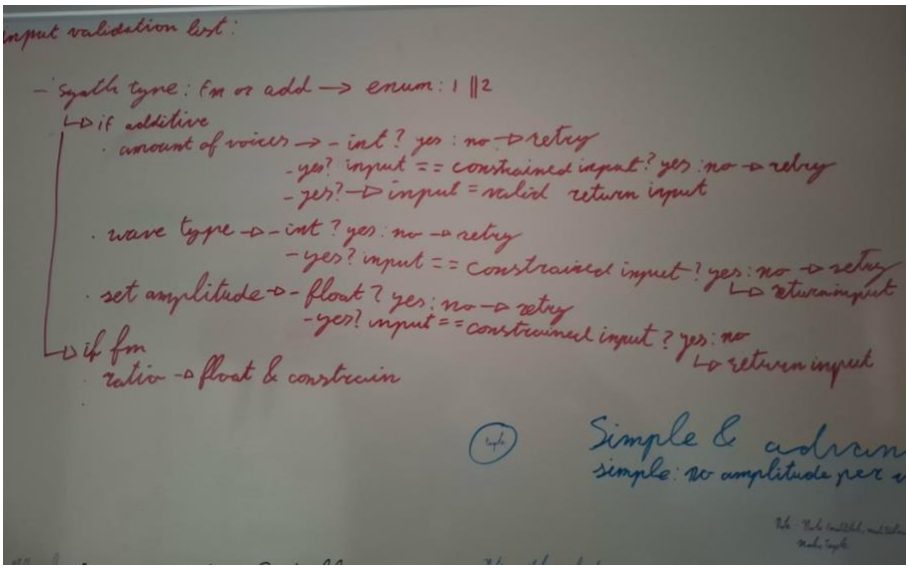
Ik ben erachter dat ik diepgang echt nodig heb om de opdracht te kunnen maken. Als ik niet doorgrond wat ik aan het doen ben en oppervlakkig blijf bij het werken aan de opdracht snap ik het niet goed, verlies ik motivatie en loop ik vast. Om dit te voorkomen zal ik komend blok heel specifiek kiezen waar ik diepgang zoek en ook waar ik dat niet doe. Zo houd ik de opdracht interessant voor mezelf en voorkom ik dat ik loopholes ga bedenken en alsnog mezelf overwerk.

3.2 Hoe om te gaan met opdrachten in sessies

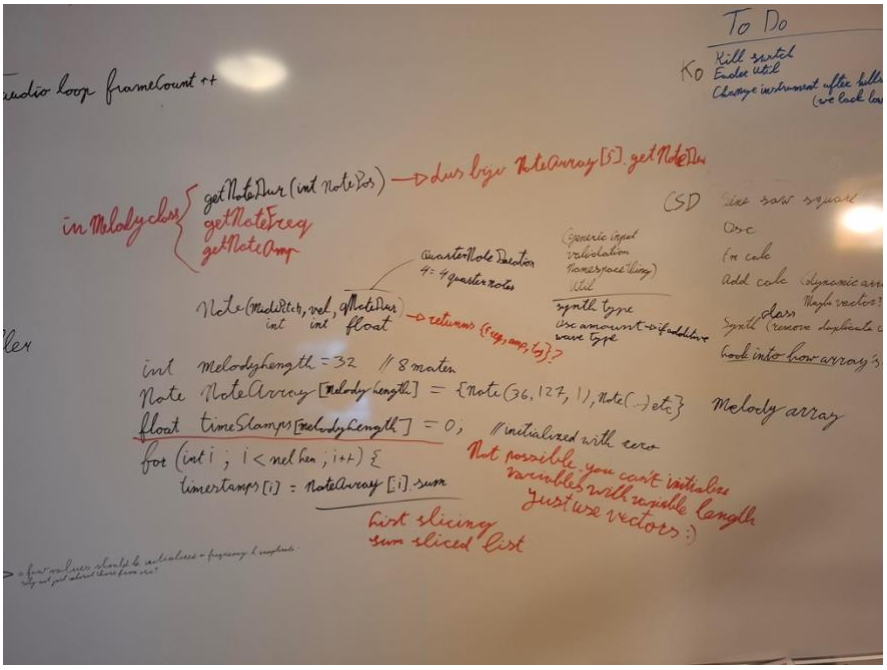
Ik denk dat ik veel tijd verlies bij het heel precies willen uitwerken van de opdrachten die we krijgen bij de sessies en dat het voor mij het beste werkt om meteen aan de slag te gaan met de eindopdracht. Echter, de opdrachten helemaal overslaan lijkt me ook niet verstandig omdat het kan dat ik een belangrijk onderdeel mis, of tijdens de eindopdracht het wiel opnieuw ga uitvinden. Ik wil in blok 2c de sessie-opdrachten die we krijgen nauwkeurig doorlezen en hieruit halen wat het leerdoel van de sessie is; Dit noteren, maar vervolgens niet de opdrachten maken. (Deliverables doe ik natuurlijk wel)

3.3 Horizontaal in plaats van verticaal werken

Dit blok heb ik veruit de meeste tijd besteed aan het lezen van code en documentatie. Uiteindelijk bij het doen van tests en het schrijven van code voor de eindopdracht ben ik als een soort printer te werk gegaan (verticale werkwijze) en heb ik eerst elk onderdeel heel precies uitgewerkt en afgemaakt voordat ik naar het volgende ging. Ik was hierbij al heel veel bezig met de netheid, documentatie en efficiëntie van de code voordat ik überhaupt een werkend programma had. Ik denk dat ik veel tijd had kunnen besparen als ik eerst had gefocust op het programma werkend krijgen en daarna pas op de code opschonen. Volgend blok wil ik eerst zo snel mogelijk naar een werkend programma en daarna pas bezig gaan met de code net en leesbaar maken.



Input validation pseudo code op whiteboard



Work in progress photo melody en note class