Ontwerp

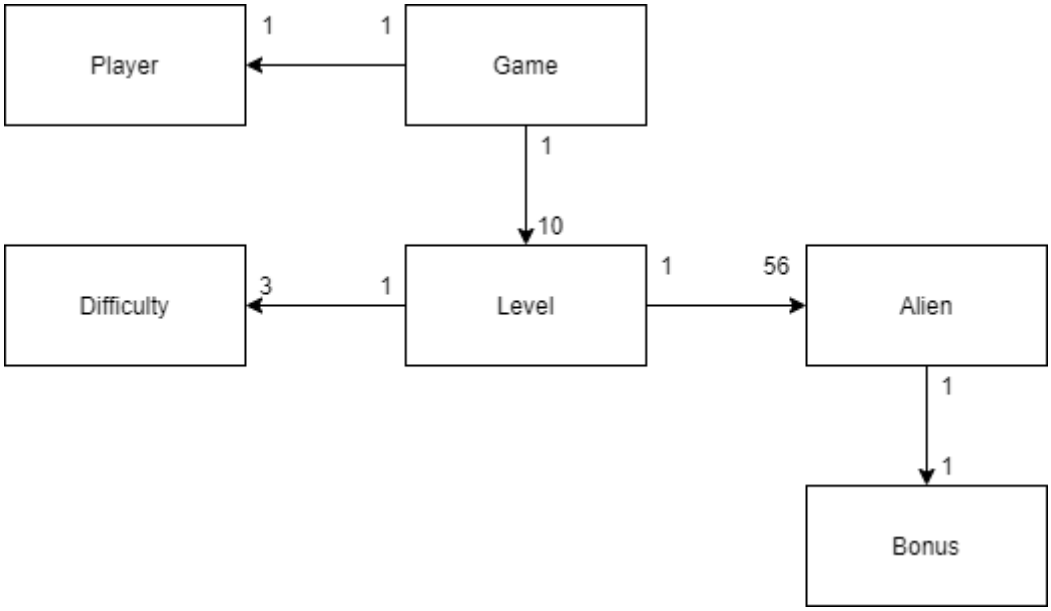
Voor het vorige semester heb ik voor mijn programma Space Invaders heb ik eerst een design moeten maken voordat ik verder mocht gaan met programmeren.
Dit viel zeer tegen, maar achteraf was ik blij dat ik eerst een ontwerp heb gemaakt.

Door eerst goed over zaken na te denken en een ontwerp te maken is het programmeren een stuk gemakkelijker worden. De keuzes die zijn gemaakt, zijn ook onderbouwd en hoeven niet meer over nagedacht te worden.
Als je achteloos gelijk gaat programmeren kom je al gauw in de problemen of later in het programma kom je erachter dat je veel moet gaan herschrijven omdat je in het begin een verkeerde keuze hebt gemaakt.

Voor mijn programma Space Invaders wilde ik graag een variant op het originele Space Invaders maken. De afwijking hierop is dat elke 3de level een eindbaas genereerde. Deze eindbaas heeft een 30-voud levens dat gekoppeld zit in hoe vaak de eindbaas als is geweest, of beter gezegd, modulus 3.
Wanneer de eindbaas nog maar 20 levens heeft verandert hij van kleur en dit gebeurt nog een keer bij 10 levens. De eindbaas is veel groter als een normale alien en schiet met 3 kogels tegelijkertijd.

Om de speler een kans te geven kan hij zijn schip verstoppen achter een 3-tal bunkers. Elke bunker bestaat uit 3 horizontale lagen die elk 3 keer geraakt moet worden voordat die laag verdwijnt. Dit kan zowel door de speler gebeuren als door de aliens.

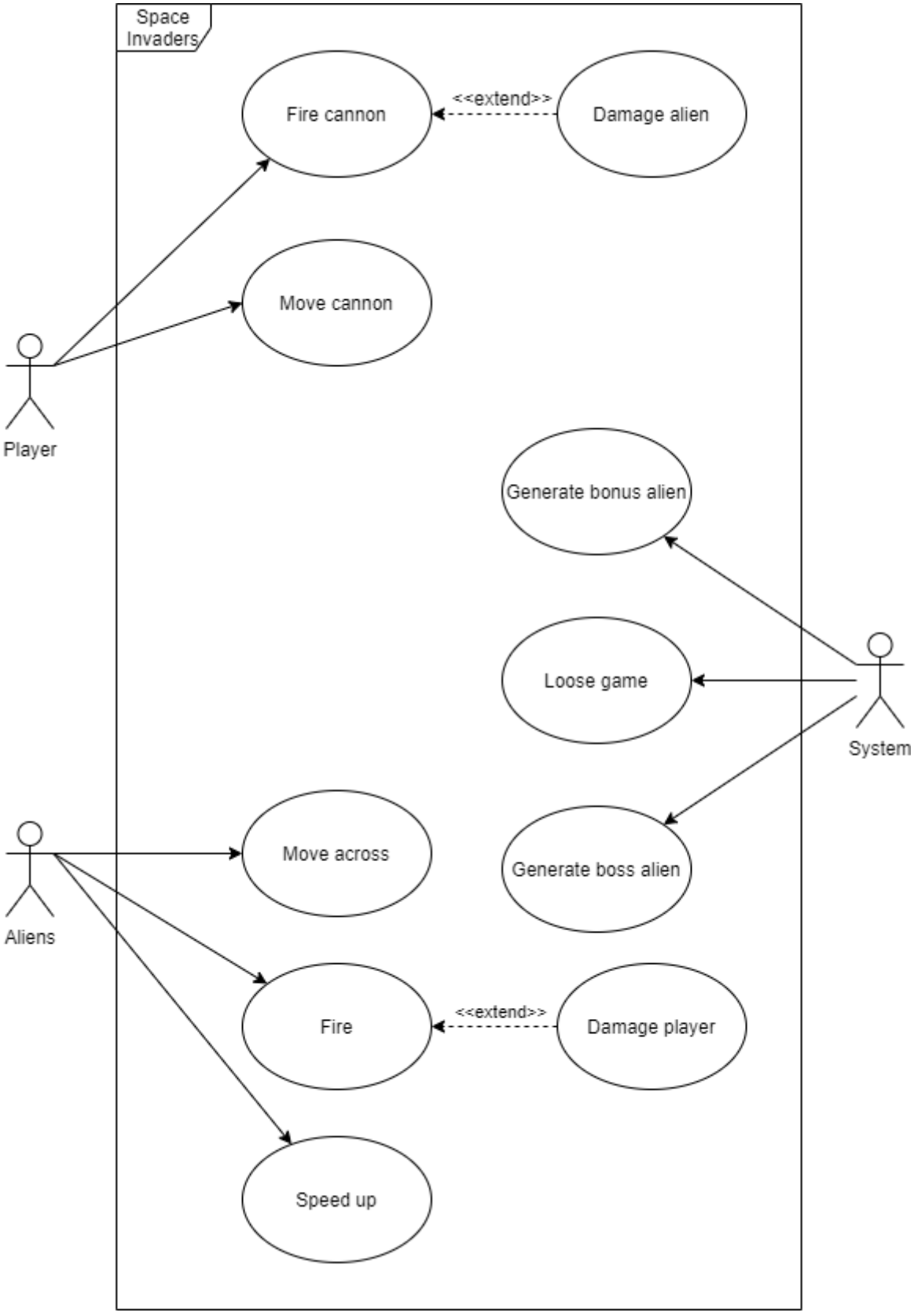
Als eerste stap is het goed om een conceptueel diagram te maken.



Uit het diagram kunnen het volgende opgemaakt worden:

- Het spel(Game) kan maar 1 speler hebben
- Het spel heeft 10 levels
- Elke level kent 3 verschillende moeilijksheidsniveaus
- Elk level heeft 56 aliens
- Elke alien kan een bonus bevatten.

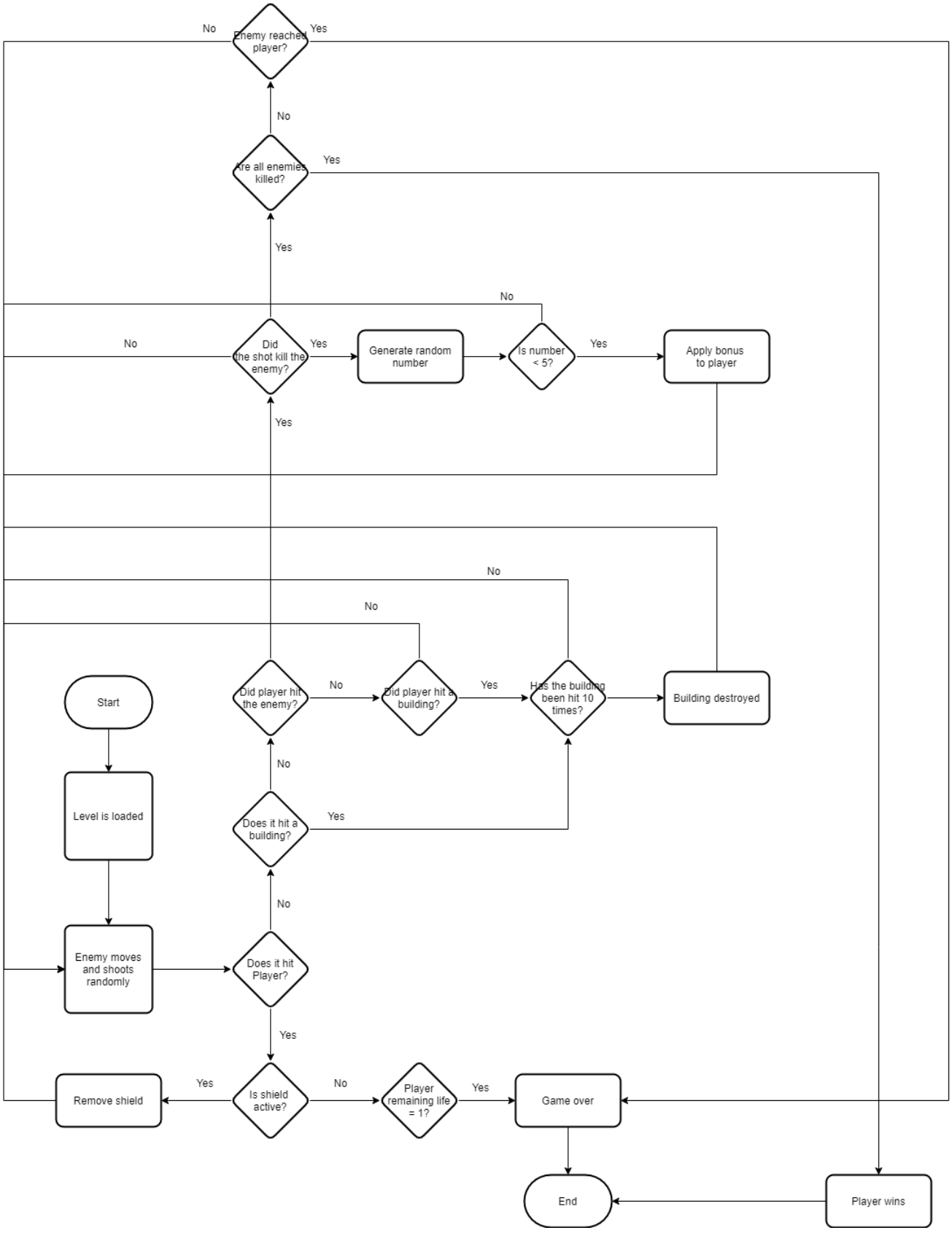
Nu dat er meer bekend is kan er een use-case diagram gemaakt worden:



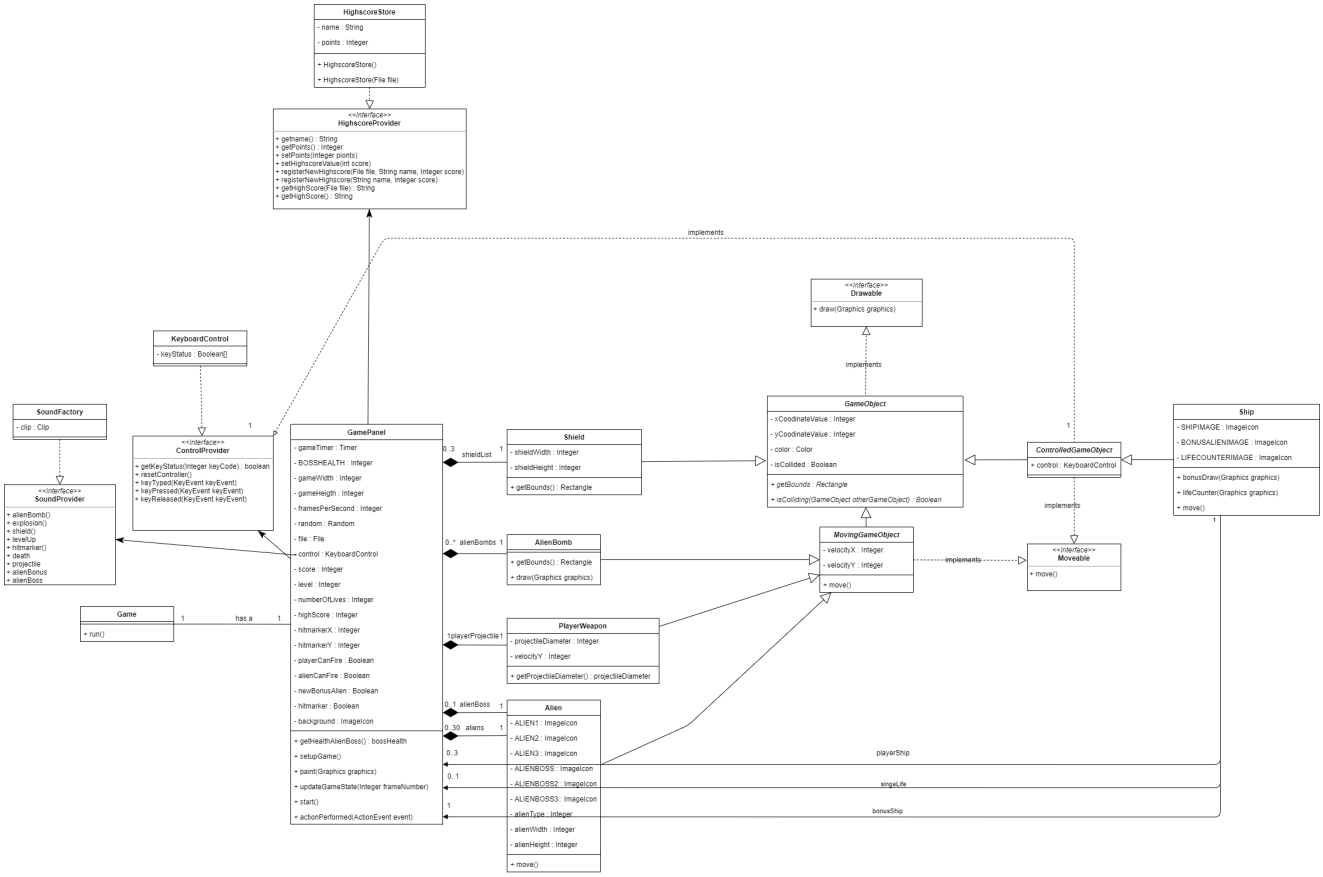
Het interessante van dit diagram is dat het 3 actuators kent, Player, Aliens en System.
System bepaald dan ook wanneer een spel bijvoorbeeld voorbij is of wanneer er een bonus of eindbaas alien moet komen.

Voor een spel als dit kan het fijn zijn om een flow chart te maken. Dit maakt het zichtbaar welke beslissingen er gemaakt moeten worden wanneer er bepaalde situaties voordoen en in welke volgorde zij

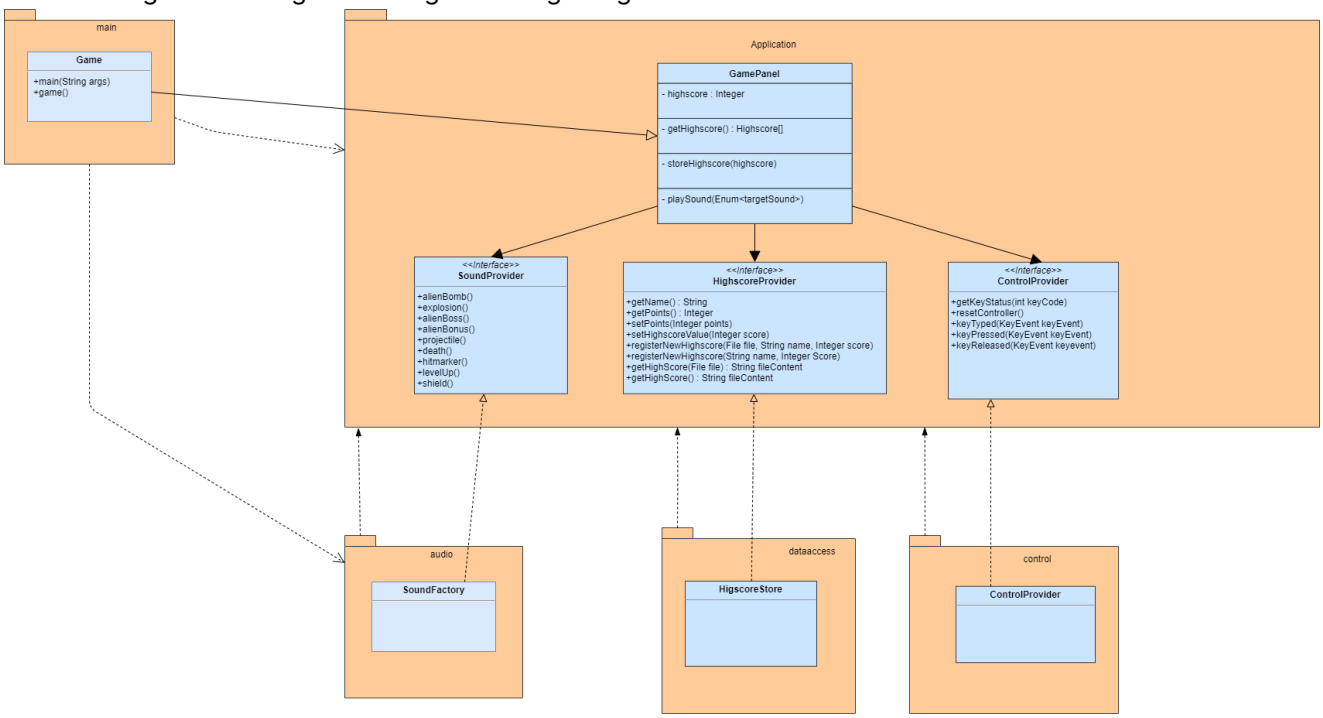
kunnen lopen.



Nu de concepten en de use-cases bekend zijn kan er een klassendiagram gemaakt worden en worden de onderliggende relaties in kaart gebracht.



Voor een lagenscheiding is het volgende diagram gemaakt.



C4-model

Voor het tweede semester heb ik een level 1 en level 2 diagram gemaakt volgens het C4-model. Na het volgen van de video Uitleg C4 model op C4model.com heb ik een goed begrip gekregen van het C4-model.

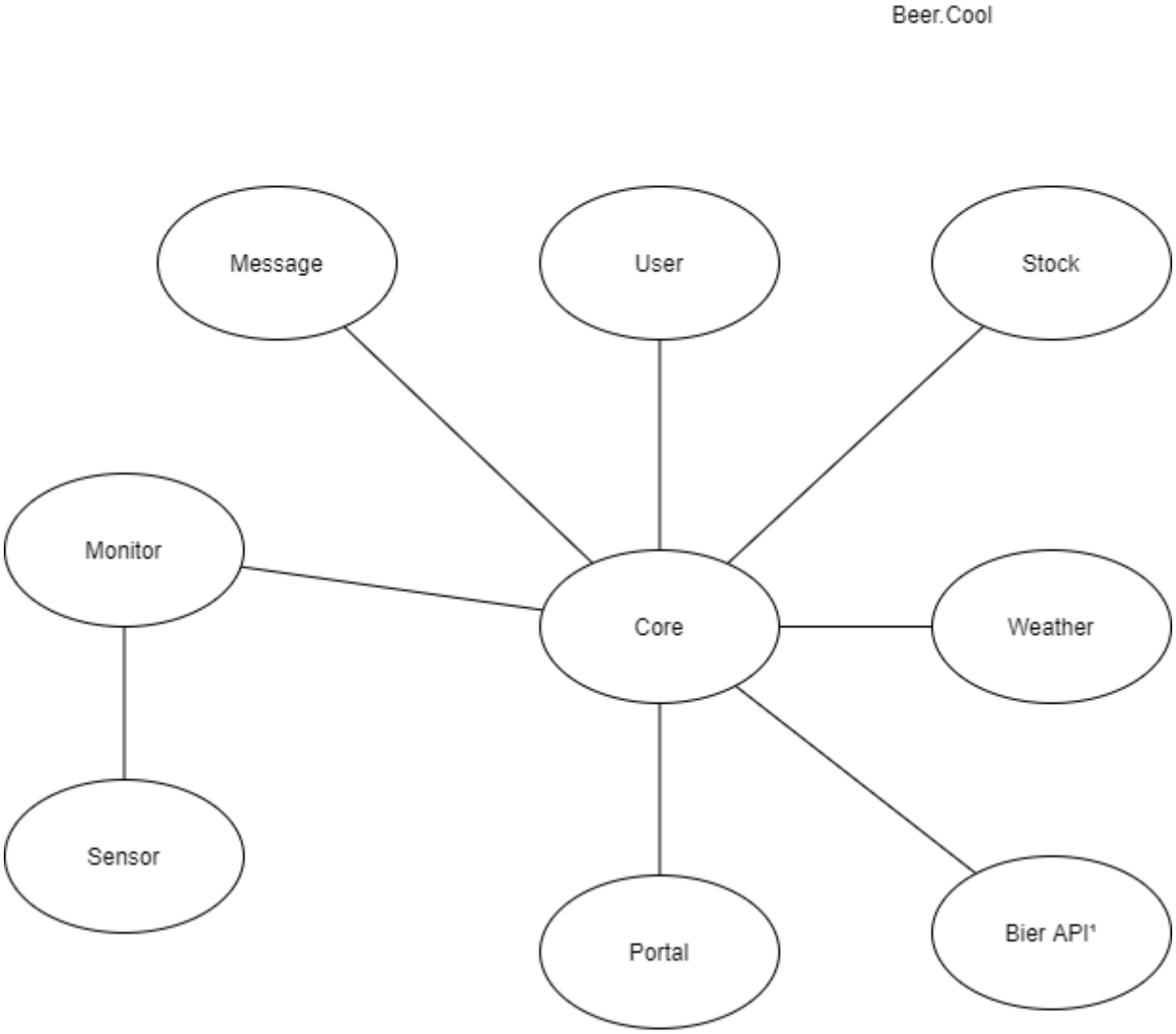
Zoals hierboven beschreven voor mijn show-case is het heel erg lastig om veranderingen bij te houden, daar gaat veel tijd inzitten.

Het mooie aan het C4-model is dat het verschillende zaken weergeeft zoals het opbreken van een systeem in onderdelen en componenten, de relaties tussen deze elementen.

Het C4-model is verdeeld in 4 lagen.

- level 1 Context diagram, geeft weer wat het bereik is van de applicatie en welke relaties er tussen gebruikers en andere systemen zijn.
- level 2 container diagram, geeft een systeem weer dat onderverdeelt is in containers. Een container vertegenwoordigt een applicatie of data opslag.
- level 3, geeft een container weer verdeelt in componenten en hoe de relaties tussen deze en andere systemen/componenten zijn
- level 4, klassendiagram

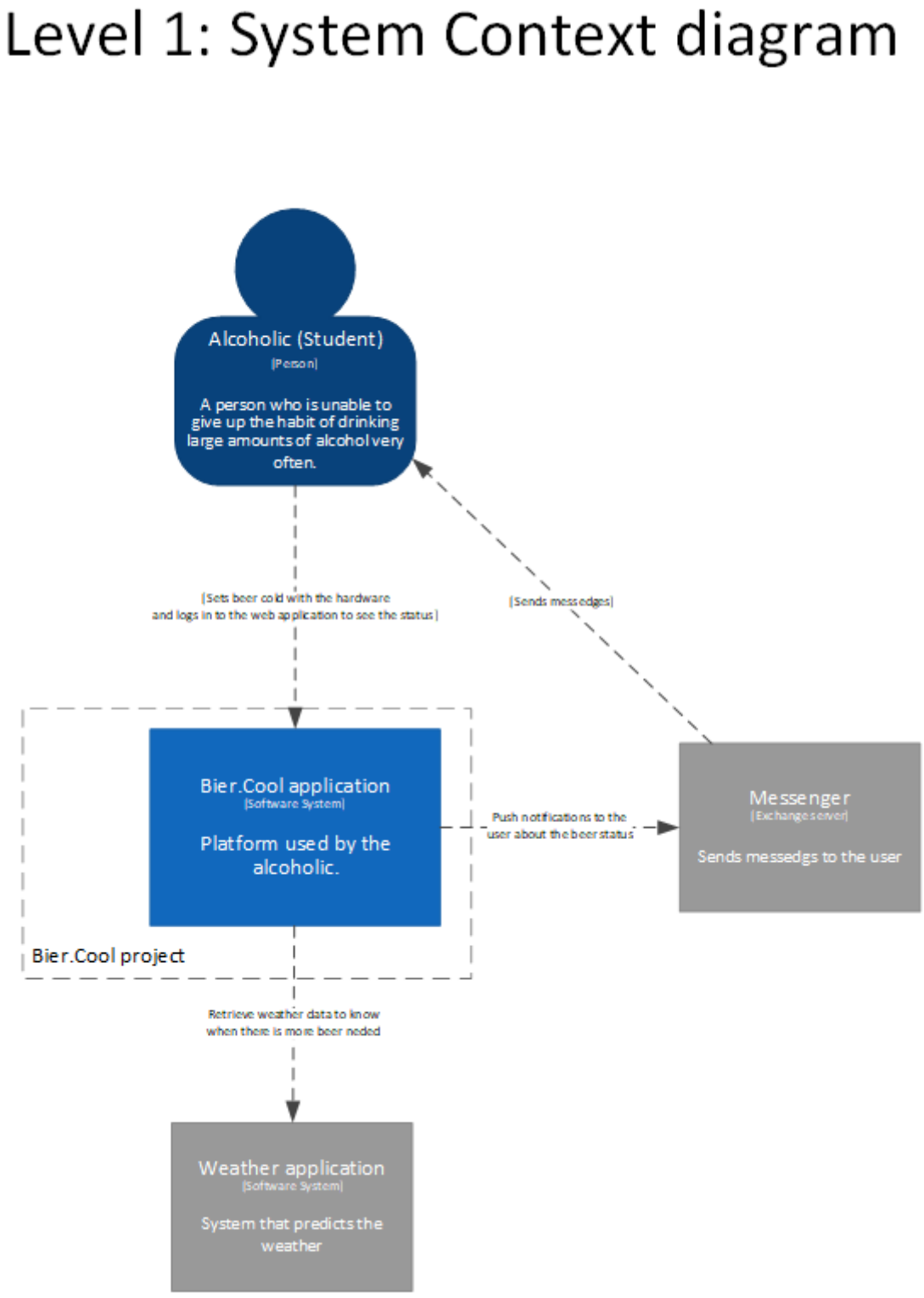
We hebben samen met de groep gezeten en hebben een domein getekend.



¹ 18 Beer APIs (2021) | ProgrammableWeb

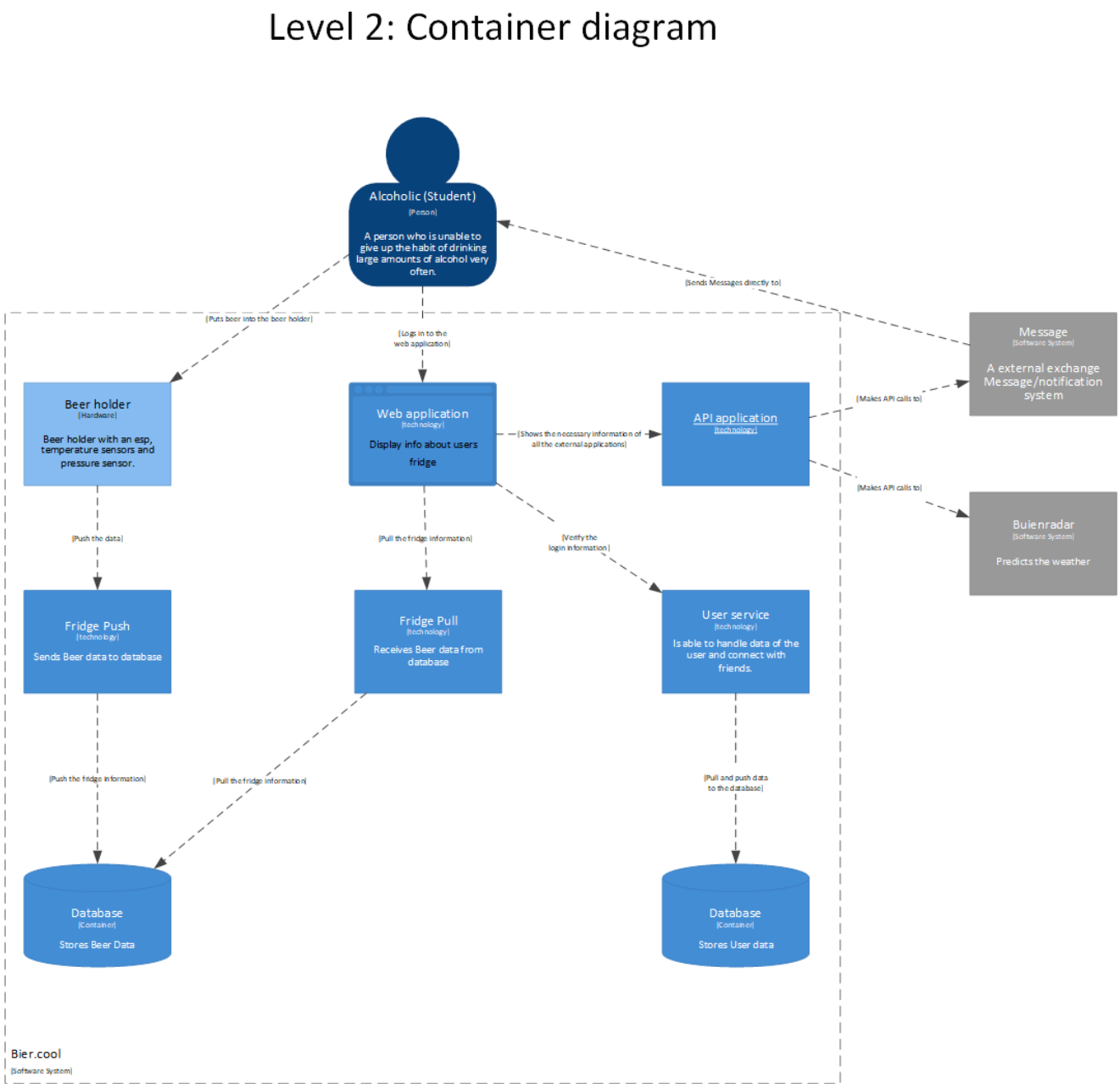
Example: DSDLink Global Brands

Vervolgens heb ik volgens het C4 model een level 1 diagram gemaakt.



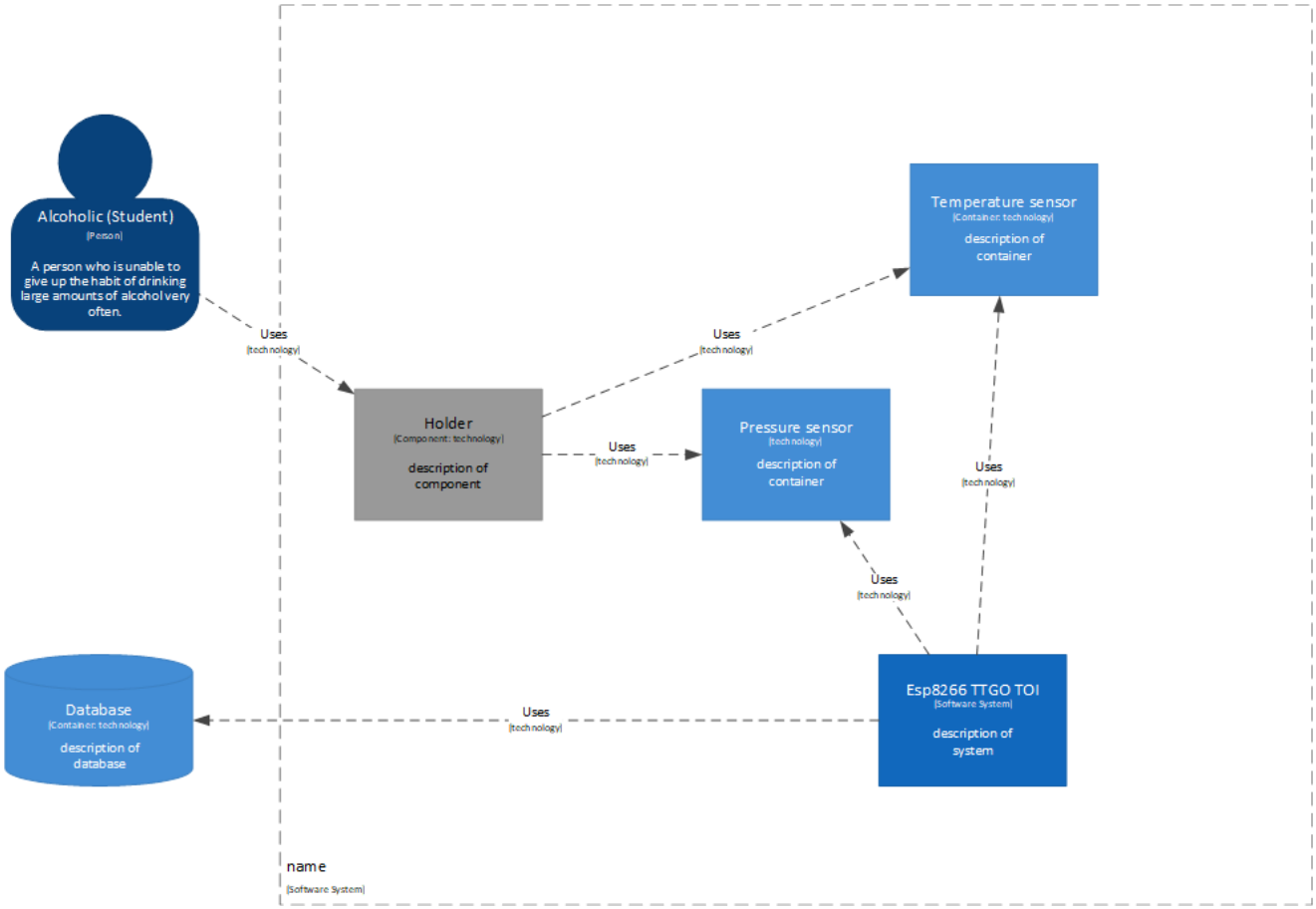
In dit ontwerp is de webapplicatie van Beer.Cool weergegeven in een level 2 diagram volgens het C4 model. De container geeft een software systeem weer die binnen de applicatie bestaat. De pijlen tussen

de containers geeft de communicatie van en naar de containers weer.



> **Scope**: A single container.
>
> **Primary elements**: Components within the container in scope.
> **Supporting elements**: Containers (within the software system in scope) plus people and software systems directly connected to the components.
>
> **Intended audience**: Software architects and developers.
>
> **Recommended for most teams**: No, only create component diagrams if you feel they add value, and consider automating their creation for long-lived documentation.

(**Bron:** *Level 2: Container diagram*)



Na een eerste feedback met de docent bleek deze niet helemaal juist te zijn.
Na het bespreken in de groep en aanpassing door te voeren zijn we uitgekomen op de volgende versie:

Level 3: Component diagram