KERKERS EN DRAKEN

Eindopdracht C++

EINDOPDRACHT C++: KERKERS EN DRAKEN

INLEIDING

Deze eindopdracht bestaat uit twee delen. In het eerste deel laat je zien dat je veilig geheugen kunt beheren door middel van het gebruik van raw pointers, en in het tweede deel maak je gebruik van de extra functionaliteit binnen C++ zoals STL containers en smart pointers.

Notabene: Tegenwoordig is het veel gebruikelijker om smart pointers te gebruiken in plaats van raw pointers. Die laatste zijn vaak lastiger in het gebruik en vereisen dat je veel beter oplet dat je niets fout doet. Hierdoor kunnen raw pointers geregeld een bron van (vervelende) bugs zijn. Echter is het doel van het vak C++ niet alleen om te leren programmeren in deze taal, maar ook om een beter inzicht te krijgen in hoe programmeertalen omgaan met het geheugen (ook al wordt dat bij managed talen zoals bijvoorbeeld Java en C# voor je gedaan). Raw pointers geven ons dat inzicht.

De hele eindopdracht zal een spel worden in de vorm van een combinatie van een text adventure (ook wel bekend als interactive fiction) met een dungeon crawler. Omdat we alles bij tekst houden kunnen we de applicatie bouwen als console app en hoeven we ons dan ook alleen maar bezig te houden met C++.

Voor de opdracht maak je een backend dat fungeert als een soort game engine waarin “de wereld” wordt opgeslagen (dwz alle locaties met inhoud die de speler kan aandoen). En een frontend dat zal dienen voor de selectie van de kaart en waarin alles met betrekking tot de speler wordt opgeslagen.

DEEL I: DE BACKEND MET ENKEL RAW POINTERS

De backend heeft de volgende requirements:

1. De spelwereld wordt door de backend in het geheugen opgeslagen.
2. De spelwereld bestaat uit locaties.
3. Iedere locatie heeft:
   - 3.1. Een unieke naam.
   - 3.2. Een algemene omschrijving. Het betreft hier een korte eerste indruk van de locatie. Alle vijanden worden ook genoemd.
   - 3.3. Een gedetailleerde omschrijving. Het betreft hier een uitgebreidere omschrijving dan de algemene beschrijving. Alle vijanden en zichtbare objecten worden ook genoemd.
   - 3.4. 0 of meer zichtbare objecten. Deze objecten worden genoemd in de gedetailleerde omschrijving. Hierop kan de speler acties ondernemen.
   - 3.5. 0 of meer verborgen objecten. Deze objecten ziet de speler niet. (Als de “zoek” actie wordt gegeven dan verplaatsen deze verborgen objecten zich naar de zichtbare objecten: Zie ook 10.2)
   - 3.6. 1 of meer uitgangen. Een uitgang is altijd één van deze richtingen: Noord, Zuid, Oost, of West. Deze uitgangen leiden naar nieuwe locaties.
   - 3.7. 0 of meer vijanden.
4. Een object heeft:
   - 4.1. Een unieke naam. Uiteraard mag je een cijfer achter een naam plakken om deze uniek te maken (*toverdrank1* en *toverdrank2* bijvoorbeeld).
   - 4.2. Een omschrijving
   - 4.3. Een type: wapen, wapenrusting, consumeerbaar, goudstukken, anders
   - 4.4. Als het om een goudstukken object gaat, dan is er een aantal mee gemoeid (het aantal goudstukken dat bij het object hoort). Dit aantal is 1 of meer en moet worden opgeslagen bij het object.

5. Een vijand heeft:
   5.1. Een unieke naam. Ook deze naam mag je uniek maken door er een cijfer achter te zetten (*ork1* en *ork2* bijvoorbeeld).
   5.2. Een omschrijving.
   5.3. 0 of meer verborgen objecten. Deze objecten ziet de speler niet. (Wanneer de vijand is verslagen kan de speler een "bekijk" actie doen op de vijand. Deze verborgen objecten verplaatsen zich dan naar de zichtbare objecten in de locatie: Zie ook 10.7.2)
   5.4. Een aantal levenspunten
   5.5. De kans dat een aanval lukt (bijvoorbeeld iedere ronde 30% kans dat de speler levenspunten verliest)
   5.6. Een willekeurig aantal levenspunten dat een speler verliest bij een succesvolle aanval (bijvoorbeeld een score van 3-10 doet minstens 3 levenspunten schade en maximaal 10 levenspunten).
6. De backend heeft een methode die ervoor zorgt dat alle vijanden zich (mogelijk) verplaatsen
   6.1. Elke vijand heeft 50% kans dat deze zich verplaatst.
   6.2. Eke vijand die zich verplaatst, verplaatst zich door middel van een random bepaalde uitgang naar een andere locatie.
   6.3. Een vijand verplaatst zich nooit door meer dan één uitgang per keer (in andere woorden: er zijn geen vijanden die twee of meer locaties per keer lopen)
   6.4. Een vijand die verslagen is (0 levenspunten heeft) zal uiteraard niet bewegen.

DEEL II: DE FRONTEND MET ENKEL SMART POINTERS

De frontend heeft de volgende requirements:

7. Bij het opstarten van het spel moet de speler een keuze kunnen maken tussen het laden van een kerker file, of het genereren van een random kerker.
   7.1. Bij deze opdracht krijg je de **kasteelruine.xml** file om te testen en om te zien hoe deze in elkaar zit. Deze file moet correct ingelezen en uitgevoerd kunnen worden. Let op: Bij het assessment zal er zomaar een andere file gebruikt kunnen worden dan deze test file. Deze assessment file krijg je niet van tevoren. Maak dus vooral ook je eigen test files om je code te testen.
   7.2. Een random kerker wordt gegenereerd door te vragen aan de speler hoeveel locaties deze moet hebben. Er wordt dan een kerker gegenereerd van dat formaat, waarbij:
      7.2.1. Elke locatie heeft een naam + beschrijving die je haalt uit de meegeven **kerkersendraken.db** SQLite3 database bij deze opdracht. Let op: elke locatie is uniek en de naam mag dus niet meer dan één keer voorkomen.
      7.2.2. Elke locatie heeft 0 tot 3 zichtbare objecten.
      7.2.3. Elke locatie heeft 0 tot 2 verborgen objecten.
      7.2.4. Elke locatie heeft 1 tot 4 uitgangen.
      7.2.5. Elke locatie moet bereikbaar zijn.
      7.2.6. Er is één vijand per 3 locaties in de wereld (afronden naar boven).
      7.2.7. Haal de objecten en vijanden uit de meegegeven database bij deze opdracht. Let op: Iedere objectnaam en iedere vijandnaam moet uniek zijn, maar je mag aan de naam van hetzelfde object steeds een nummer plaatsen (dus *toverdrank1*, *toverdrank2*, etc., of *ork1*, *ork2*, etc.)
8. Na het inladen/genereren van de kerker moet er een speler aangemaakt worden. Deze heeft de volgende eigenschappen:
   8.1. Een naam (de speler moet deze zelf in voeren)
   8.2. 20 levenspunten
   8.3. Een aanvalskans van 40%

8.4. Een *Dolk* als startwapen
8.5. Draagt geen wapenrusting
8.6. Heeft 0 goudstukken
8.7. Heeft geen objecten (anders dan een startwapen)

9. Alle speleracties moeten waar mogelijk worden uitgevoerd in de frontend. De speler mag dan ook niet meegegeven worden aan de backend. *Bijvoorbeeld: Als een speler een object oppakt, dan moet deze verwijderd worden uit de zichtbare objecten van de locatie. Dat gebeurt in de backend. Het toevoegen van dat object aan de speler moet dan echter wel geheel plaatsvinden in de frontend.*
10. Elke keer moet er gekozen worden om één van de volgende acties te ondernemen (zie ook hoofdstuk *Acties*):
    10.1. **Kijk** : Geeft de beschrijving van de locatie met alle zichtbare objecten, alle mogelijke uitgangen, en alle aanwezige vijanden.
    10.2. **Zoek*** : Verplaatst de verborgen objecten in de huidige locatie naar de zichtbare objecten in dezelfde locatie. Let op: De verborgen objecten die bij een vijand horen die zich in deze locatie bevindt blijven verborgen.
    10.3. **Ga <noord|zuid|oost|west>*** : Indien er inderdaad een uitgang is in de aangegeven richting, dan verplaatst de speler zich naar de aangegeven nieuwe locatie. Echter als er vijanden in de huidige locatie staan van de speler, dan doen ze ieder één aanval. Pas als de speler dat overleeft gaat deze naar de nieuwe locatie.
    10.4. **Pak <OBJECT>** : Verplaatst het genoemde zichtbare object van de locatie naar de speler (als het inderdaad aanwezig is). Echter gaat het hier om een object van het type “goudstukken”, dan moet het aantal goudstukken van het object opgeteld worden bij het aantal goudstukken van de speler. Er hoeft dan geen object verplaatst te worden, maar het goudstukken object zelf moet wel worden gedeletet.
    10.5. **Leg <OBJECT>** : Verplaats een object van de speler naar de zichtbare objecten in de huidige locatie.
    10.6. **Bekijk <OBJECT>** : Geef de beschrijving van het genoemde object, als deze zich onder de zichtbare objecten bevindt in de huidige locatie, of in het bezit van de speler is.
    10.7. **Bekijk <VIJAND>** :
        10.7.1. **Vijand heeft nog levenspunten**: Geef de beschrijving van de vijand (als deze in de huidige locatie aanwezig is).
        10.7.2. **Vijand heeft 0 levenspunten**: Geeft de beschrijving van de vijand (als deze in de huidige locatie aanwezig is) en verplaatst verborgen objecten die zich op de vijand bevinden naar de zichtbare objecten van de huidige locatie. Let op De verborgen objecten van de locatie blijven verborgen.
    10.8. **Bekijk Zelf** : Toont alle gegevens van de speler: Levenspunten, aanvalskans, goudstukken, gedragen wapen, gedragen wapenrusting, en de lijst met objecten die de speler bij zich heeft.
    10.9. **Sla <VIJAND>*** : Val de genoemde vijand aan met het wapen dat de speler vast heeft. Zie hoofdstuk Aanvallen.
    10.10. **Draag <OBJECT:WAPENRUSTING>*** : Als het genoemde object in het bezit is van de speler en van het type wapenrusting is, dan wordt dat geregistreerd als de gedragen wapenrusting van de speler. Een eventuele andere wapenrusting die de speler al draagt wordt automatisch uitgedaan, en wordt een zichtbaar object in de locatie waar de speler staat. In andere woorden: Een speler draagt nooit meer dan één wapenrusting (maar zou er wel meerdere in zijn bezit kunnen hebben).

10.11. **Draag <OBJECT:WAPEN>** : Als het genoemde object in het bezit is van de speler en van het type wapen is, dan wordt dat geregistreerd als het wapen dat de speler vast heeft. Een eventueel ander wapen dat de speler vast heeft wordt automatisch neergelegd, en wordt een zichtbaar object in de locatie waar de speler staat. Een speler heeft nooit meer dan één wapen vast (maar zou er wel meerdere in zijn bezit kunnen hebben).
10.12. **Wacht***: Je slaat de beurt over, en doet dus niets. (Maar monsters kunnen aanvallen en bewegen zoals genoemd bij punt 11)
10.13. **Consumeer <OBJECT:CONSUMEERBAAR>** : Afhankelijk van welke toverdrank wordt geconsumeerd gebeurt het volgende:
10.13.1. **Levenselixer**: De speler krijgt 5 tot 15 levenspunten erbij.
10.13.2. **Ervaringsdrank**: De aanvalskans van de speler wordt verhoogd met +10%, maar nooit naar meer dan 90%.
10.13.3. **Teleportatiedrank**: De speler wordt verplaatst naar een willekeurige locatie.
10.14. **Help**: Laat een lijst zien van de beschikbare acties waaruit een speler kan kiezen.
10.15. **Godmode**: Hiermee kan de de "godmode" in- en uitgeschakeld worden. Tijdens godmode kan de speler geen levenspunten verliezen en kan niet meer missen met een aanval.
10.16. **Quit**: Het spel wordt beëindigd en de applicatie wordt netjes afgesloten (dwz zodat eventuele memory leaks zichtbaar worden)

11. Elke keer *na* het uitvoeren van een actie genoemd bij punt 10 gemarkeerd met een ***** worden de volgende dingen gedaan:
11.1. Een vijand die zich in dezelfde locatie bevindt als de speler zal de speler aanvallen
11.2. Een vijand die verslagen is (0 levenspunten heeft) zal uiteraard niet aanvallen.
11.3. Als er geen vijanden zijn in de huidige locatie waar de speler zich bevindt: De backend wordt de opdracht gegeven om de vijanden te bewegen (zie punt 6 bij de backend).

ACTIES

De speler krijgt standaard ***geen*** lijst te zien met mogelijke acties. De speler ziet een prompt en typt dan een tekst in. Als deze overeenkomt met een mogelijke actie, dan wordt deze actie door het spel uitgevoerd. Is deze actie op dat moment niet mogelijk (of deze actie is niet bekend), dan brengt het spel de speler op de hoogte dat dit niet mogelijk is.

De speler kan met behulp van de *Help* actie een opsomming krijgen van bestaande acties. (Dus een lijst met in punt 10 aangegeven acties.)

AANVALLEN

Zoals bij punt 11 genoemd vallen vijanden aan als deze zich in de huidige locatie van de speler bevinden, en de speler heeft net een actie gedaan gemarkeerd met een *. Een speler valt aan als hij de actie *Sla <VIJAND>* uitvoert.

AANVAL VAN SPELER OP VIJAND

De aanvalskans is de kans dat een aanval slaagt, uitgedrukt als percentage. Als de aanval van de speler slaagt dan doet de speler een aantal levenspunten schade bij de vijand afhankelijk van het wapen dat je draagt.

Ieder wapen heeft een *minimum* schade en een *maximum* schade. Bepaal iedere geslaagde aanval random een getal dat ligt tussen deze twee waarden (inclusief de waarden zelf). Dit is het aantal levenspunten dat de vijand verliest.

Een vijand kan nooit minder dan 0 levenspunten hebben. Dus als je een vijand meer schade doet dan dat deze levenspunten heeft, dan houdt deze 0 levenspunten over. Zodra een vijand 0 levenspunten heeft, dan komt er achter zijn naam het suffix *[verslagen]* te staan. (Zie ook actie 10.7.2.)

AANVAL VAN VIJAND OP SPELER

Dit gaat net als bij een aanval van een speler op een vijand: Er is een aanvalskans. Als er random bepaald is dat deze raakt, dan doet de vijand een random bepaald aantal levenspunten schade bij de speler. Een vijand hoeft geen wapen te hebben om schade te kunnen doen.

Het aantal levenspunten schade dat een speler incasseert wordt verminderd met het aantal punten bescherming dat zijn wapenrusting tegenhoudt. Je kunt uiteraard nooit minder dan 0 punten schade krijgen. *Bijvoorbeeld: Je wordt gebeten door een wolf en krijgt 3 punten schade. Je hebt echter een zacht leren wapenrusting aan die 1 punt schade tegenhoudt. Hierdoor verlies je uiteindelijk slechts 2 levenspunten (=3-1).*

Als een speler 0 levenspunten heeft, dan is het spel voorbij. (Zie ook Einde Spel.)

EINDE SPEL

Als een speler op 0 levenspunten staat, dan is het spel voorbij. De score die de speler behaald heeft is gelijk aan het aantal goudstukken dat gevonden is.

Sla de score met de speler naam op in de Leaderboard tabel in de database, en laat een lijstje zien met de top tien van scores zoals die in de database staan geregistreerd.

De speler heeft nu de volgende twee keuzes:

1. **Opnieuw**: De speler begint weer bij punt 7 van hoofdstuk *Deel II: De frontend met enkel smart pointers*.
2. **Quit**: De applicatie wordt beëindigd en netjes afgesloten (dwz zodat eventuele memory leaks zichtbaar worden).

VOORBEELD

In dit hoofdstuk volgt er een voorbeeld van de output van het spel. Het gaat hier om een stuk in het midden; het begin en eind van het spel is weggelaten. De spelersinput is voor dit voorbeeld in blauw aangegeven.

```
...
Je staat in de locatie: Kasteelpoort
Wat is je commando?
> Bekijk Zelf
Je hebt 20 levenspunten.
Je hebt een aanvalskans van 40%
Je hebt het volgende wapen vast: Dolk (1 - 4 schade).
Je draagt de volgende wapenrusting: Geen
Je hebt 0 goudstukken
Je bezit bestaat uit de volgende objecten: Geen
> Kijk
```

Je staat bij de locatie Kasteelpoort.

De grauwe stenen zijn zwart uitgeslagen en op veel plekken bedekt met mos. De poort staat open en het valhek staat omhoog.

Zichtbare objecten: Geen

Uitgangen: Noord

Er zijn geen vijanden.

> **Zoek**

Je vindt:

Roestig Zwaard (2 - 5 schade).

:: De vijanden bewegen ::

Rat1 komt de huidige locatie binnen

> **Pak Roestig Zwaard**

Je voegt Roestig Zwaard toe aan je bezit

> **Zelf**

Je hebt 20 levenspunten.

Je hebt een aanvalskans van 40%

Je hebt het volgende wapen vast: Dolk (1 - 4 schade).

Je draagt de volgende wapenrusting: Geen

Je hebt 0 goudstukken

Je bezit bestaat uit de volgende objecten: Roestig Zwaard

> **Draag Roestig Zwaard**

Je laat Dolk vallen en pakt Roestig Zwaard vast.

> **Zelf**

Je hebt 20 levenspunten.

Je hebt een aanvalskans van 40%

Je hebt het volgende wapen vast: Roestig Zwaard (2 - 5 schade).

Je draagt de volgende wapenrusting: Geen

Je hebt 0 goudstukken

Je bezit bestaat uit de volgende objecten: Geen

> **Sls Rat**

Ik snap niet wat je bedoelt?

> **Sla Rat**

Er is hier geen vijand die Rat heet.

> **Sla Rat1**

Je slaat mis!

:: De vijand(en) in je locatie vallen aan! ::

Rat1 raakt en doet 1 punt schade.

Je hebt nog 19 levenspunten over.

> **Sla Rat1**

Je slaat raak en doet 4 punten schade.

Rat1 heeft 0 levenspunten over en is dood!

> **Kijk**

Je staat bij de locatie Kasteelpoort.

De grauwe stenen zijn zwart uitgeslagen en op veel plekken bedekt met mos. De poort staat open en het valhek staat omhoog.

Zichtbare objecten: Dolk (1 – 4 schade)

Uitgangen: Noord

Vijanden: Rat1[verslagen]

> **Bekijk Rat1**

Rat1 ziet eruit als een grote rat met priemende rooie oogjes. Rat1 heeft 0 levenspunten. Rat1 heeft geen verborgen objecten bij zich.

> _

FILE FORMAT XML

Bij deze eindopdracht krijg je de kasteelruine.xml file. Deze heeft de volgende opbouw:

Locaties: In de file staat een verzameling met **Locatie** items.

Locatie: Een locatie bevat de **naam** van de locatie en een **beschrijving** item. De Locatie heeft de volgende attributen:

- **Id**: Dit is een uniek nummer waarmee *binnen de xml file* makkelijk naar deze Locatie verwezen kan worden (mbt uitgangen)
- **noord/oost/zuid/west**: Deze zijn optioneel en worden alleen gegeven als deze Locatie een uitgang heeft in die richting. Dit attribuut verwijst dan naar het Id van de Locatie waar die uitgang naar leidt. *Bijvoorbeeld: oost=”3” betekent dat als je in deze Locatie de uitgang richting oost neemt, je bij de Locatie uitkomt met id=”3”.*
- **vijand**: De naam van een vijand die zich in deze locatie bevindt. Als er meerdere vijanden zijn dan worden de namen gescheiden door een ; De naam/namen komen precies overeen met de vijand namen in de SQLite3 database. Let op: Alle vijanden moet bij het instantiëren allemaal een unieke naam krijgen. (Zie ook 5.1)
- **objectenzichtbaar**: De naam van een object dat zichtbaar is in deze locatie. Als er meerdere objecten zichtbaar zijn worden ze gescheiden door een ; De objectnamen komen precies overeen met de object namen in de SQLite3 database. Ook deze namen moeten uniek zijn.
- **objectenverborgen**: Net als bij objectenzichtbaar, echter zit dit object/deze objecten in de verzameling die nog niet zichtbaar is en pas zichtbaar wordt als de speler zoekt. (Zie ook 10.2)

Beschrijving: Bevat één string: de beschrijving van de locatie.

DATABASE SCHEMA

Zie hieronder voor het database schema van de meegeleverde kerkersendraken.db file. Daarna een beschrijving van iedere tabel.

Nota bene: Je kunt de inhoud van de database ook bekijken met de SQLiteStudio die gratis te downloaden is.

SCHEMA

```
CREATE TABLE Locaties (
    naam         TEXT PRIMARY KEY,
    beschrijving TEXT NOT NULL
);

CREATE TABLE Objecttypen (
    naam TEXT PRIMARY KEY
);

CREATE TABLE Objecten (
    naam          TEXT    PRIMARY KEY,
    omschrijving  TEXT    NOT NULL,
    type                  REFERENCES Objecttypen (naam)
                          NOT NULL,
    minimumwaarde INTEGER,
    maximumwaarde INTEGER,
    bescherming   INTEGER
);

CREATE TABLE Vijanden (
    naam            TEXT    PRIMARY KEY,
    omschrijving    TEXT    NOT NULL,
    minimumobjecten INTEGER NOT NULL,
    maximumobjecten INTEGER NOT NULL,
    levenspunten    INTEGER NOT NULL,
    aanvalskans     INTEGER NOT NULL,
    minimumschade   INTEGER NOT NULL,
    maximumschade   INTEGER NOT NULL
);

CREATE TABLE Leaderboard (
    ID          INTEGER PRIMARY KEY,
    naam        TEXT    NOT NULL,
    goudstukken INTEGER NOT NULL
);
```

BESCHRIJVING

Locaties: Deze tabel bevat de namen en beschrijvingen voor random locaties. Uiteraard moet bij het genereren van een random kerker nog wel dingen als vijanden, objecten, en uitgangen worden gegenereerd bij iedere locatie.

Objecttypen: De verschillende soorten objecten die in het spel zijn. Notabene: Er zijn drie soorten consumeerbare objecten: levenselixers, ervaringsdranken, en teleportatiedranken.

Objecten: Hierin staan de namen, beschrijvingen, en typen van alle objecten. Voor wapenrusting is het *bescherming* veld ingevuld, voor de rest zijn de *minimumwaarde* en *maximumwaarde* ingevuld.

Vijanden: Hierin staan alle gegevens van de verschillende vijanden. Het aantal random verborgen objecten dat iedere random gegenereerde vijand bij zich heeft ligt tussen de *minimumobjecten* en *maximumobjecten*. Als de vijand raakslaat tijdens een aanval dan verliest de speler tussen de *minimumschade* en *maximumschade* aan levenspunten.

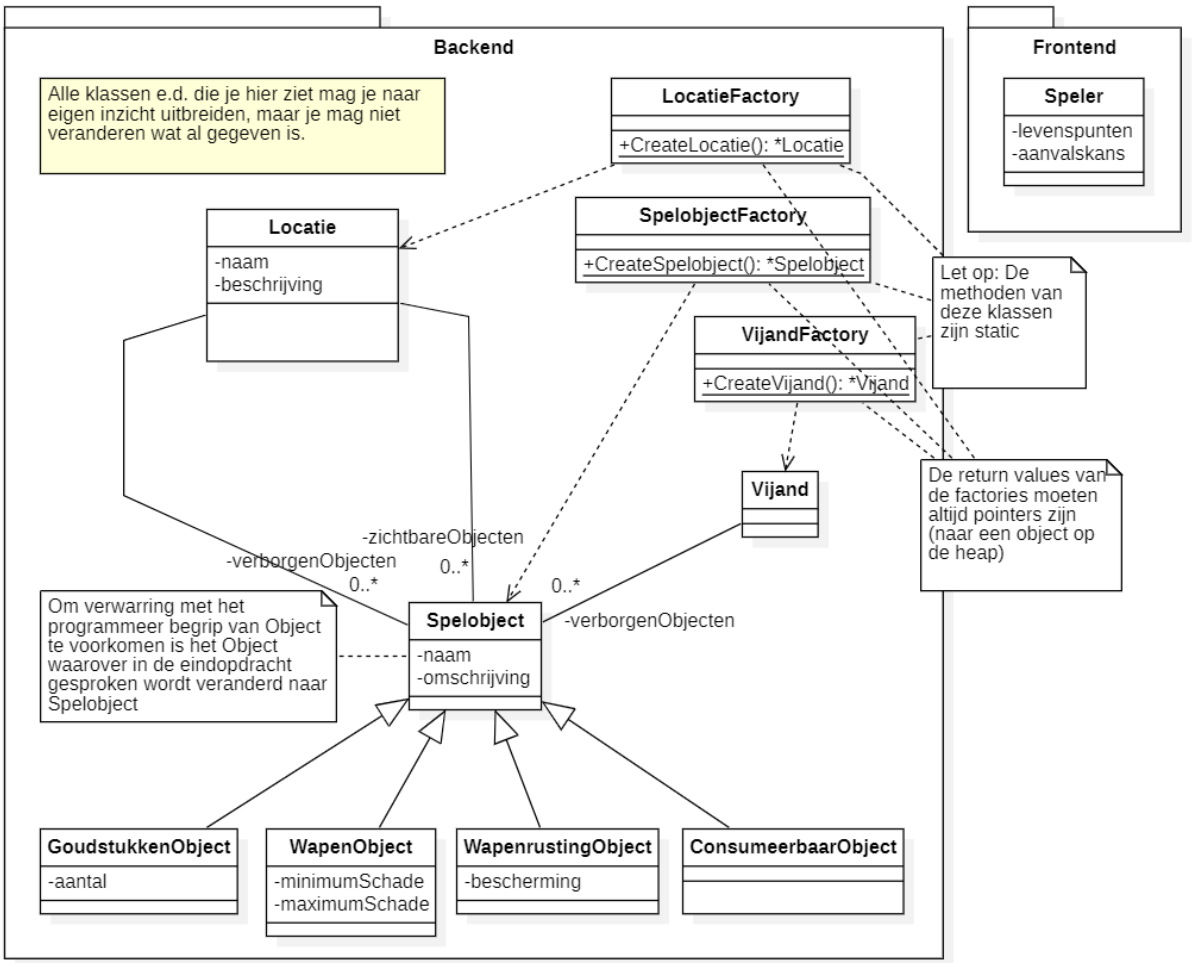
Leaderboard: Hier wordt de score van elke speler opgeslagen die het spel heeft uitgespeeld. (Zie ook *Einde Spel*.)

UML DIAGRAMMEN

Het volgende diagram moet worden gebruikt bij het implementeren van de eindopdracht. Voor de duidelijkheid: Ze mogen niet alleen maar voorkomen, ze moeten daadwerkelijk ook functioneel gebruikt worden door jouw code. Bijvoorbeeld: Elke locatie *moet* worden geïnstantieerd door de LocatieFactory.

De Factories die hier getoond worden mogen geen lijst bijhouden van alle geïnstantieerde objecten. Het eigenaarschap van een object hoort dus niet bij de Factories.

(Voor de duidelijkheid: Een lijst met objecten tbv een clone pattern mag wel, als deze lijst maar niet *alle* objecten bevat die de Factory instantieerde.)



ASSESSMENT

MINIMUM EISEN

De volgende eisen zijn **minimaal** verplicht om te implementeren. **Wanneer één of meer van deze eisen niet geïmplementeerd zijn krijg je automatisch een 1 voor de eindopdracht.**

1. Vóór de deadline verstreken is moet de laatste versie van de opdracht zijn ingeleverd op de inleverlink van Brightspace.
2. Een werkend spel, in de vorm van een C++ console application, inclusief alle dingen die in de voorgaande paragrafen beschreven staan. **Een werkend spel is een spel dat niet crasht!**
3. Er moet een vorm van *memory leak detection* beschikbaar zijn ten tijde van het assessment
4. Er mogen geen memory leaks optreden in het programma[1]
5. Elke quit optie die geïmplementeerd moet worden, moet ervoor zorgen dat de applicatie netjes beëindigd wordt, zodat memory leak detection mogelijk blijft. Let op: Als je klikt op het kruisje rechtsboven op de console, dan geldt dat ***niet*** als netjes afsluiten. De memory leak detection wordt dan namelijk ook afgebroken.
6. Als je casts gebruikt mag er ***alleen*** gebruik worden gemaakt van **C++ style casts** (static_cast, const_cast, dynamic_cast, en reinterpret_cast).
7. Voor het genereren van random nummers mag er *niet* gebruik worden gemaakt van rand(), maar moet er gebruik worden gemaakt van de **<random>** header. (Zie hoofdstuk “Random” voor voorbeelden hoe deze te gebruiken.)
8. Het is verplicht om de game data te lezen door van de SQLite3 database gebruik te maken vanuit jouw programma. Het is dus niet toegestaan die data met externe tools in een "handigere" vorm te gieten. (Zie hoofdstuk “SQLite3” voor voorbeelden hoe deze te gebruiken.)
9. Het moet mogelijk zijn om een transcript van een gamesessie vast te leggen in een "**session log**". Dat betekent dat zowel de gegevens die aan de speler worden getoond als wat de speler invoert aan die file worden toegevoegd.
   Het idee is dat je na afloop het hele spelverloop kunt nakijken door die session log te lezen. Het moet duidelijk zijn wat de speler voor acties ondernam, en wat de gevolgen waren, etc. (dus alles at de console output toont moet in de log terecht komen).
10. Voor de Backend gelden de volgende extra eisen:
    - Er mag alleen gebruik worden gemaakt van raw pointers.
    - Er mag niet gebruik worden gemaakt van de C++ Standard Template Library (let op: dit is wat anders dan de Standard Library die, op het STL deel na, wel mag worden gebruikt). Bijvoorbeeld: verzamelingen als Vector, Map, Set, etc. en de bijbehorende algoritmes mogen niet worden gebruikt.
    - Zichtbare Objecten (per locatie), Verborgen Objecten (per locatie), Verborgen Objecten (per vijand), en het bezit van de speler moeten allemaal een eigen verzameling hebben (moeten bijvoorbeeld allemaal hun eigen array hebben). Je mag dus bijvoorbeeld ***niet*** de zichtbare locatie objecten en verborgen locatie objecten in één array opslaan.
11. Voor de Frontend gelden de volgende extra eisen:
    - Er mag alleen gebruik worden gemaakt van smart pointers.
    - Je mag gewoon gebruik maken van de C++ Standard Template Library
12. Het gebruik van externe libraries is niet toegestaan, met uitzondering van:
    - SQLite3, want die heb je nodig voor het lezen van de game data.
    - Een XML library, voor het inlezen van xml files (zoals *kasteelruine.xml*)
    - Een unit test framework, zoals GTest (niet verplicht).
    - Eventuele libraries die nodig zijn voor je memory leak detection.
    - De C++ Standard Template Library mag wel, maar alleen voor de frontend

---

[1] Memory leaks waar je niets aan kan doen (mochten die in bijv. de Standard Library gesignaleerd worden) mogen wel voorkomen, maar dan moet je wel kunnen beredeneren dat deze niet veroorzaakt worden door je code. “False positives” (de memory leak detection tool geeft een leak aan, maar dit is niet correct) mogen ook voorkomen. In geval van twijfel vraag tijdig even de mening van je practicum docent.

13. De frontend en backend van je opdracht moeten *low coupling* zijn. Dit kan je bijvoorbeeld door middel van een facade pattern oplossen, maar je mag ook andere methoden van low coupling toepassen.
14. De gegeven UML diagrammen in deze opdracht moeten geïmplementeerd worden in je eindopdracht. Zie hoofdstuk UML Diagrammen voor additionele eisen.
15. Al het versiebeheer moet via *Github Classroom* gaan. De link hiernaar kun je vinden op Brightspace.

REPARATIE

Het is niet toegestaan om reparaties te doen *tijdens* het assessment. Ook is het niet toegestaan om een reparatie te doen en later op de dag het assessment te herhalen. In andere woorden: als je applicatie **crasht** of **memory leaks** vertoont, of op een andere manier niet voldoet aan de minimum eisen krijg je een onvoldoende voor de eindopdracht en zul je het assessment opnieuw moeten doen bij een herkansing. **Zorg dus dat je werk uitvoerig getest is voordat je bij het assessment verschijnt!**

Ben er ook van bewust dat de herkansing voor de eindopdracht die er volgende periode komt voor degenen die de eindopdracht de eerste keer niet gehaald hebben, niet dezelfde opdracht hoeft te zijn. Dus het repareren van de eindopdracht na de deadline is waarschijnlijk geen mogelijkheid.

BEOORDELING

De eindopdracht zal tijdens het assessment op de volgende manier beoordeeld worden:

	Eis # in pdf	punten	Commentaar
Knock-outs:			
Geen C++ console applicatie	ME.1	-	
Applicatie crasht	ME.2	-	
Memory leak detection is niet aanwezig	ME.3	-	
De twee nette quit opties zijn niet aanwezig. (als actie en als optie aan het einde van een spel)	ME.5	-	Een nette quit optie maakt het mogelijk dat er memory leak informatie te zien is (Het rode kruisje rechtsboven de console wordt dus niet als net beschouwd)
Geen gebruik gemaakt van SQLite3	ME.8	-	
Backend: Smart pointers gebruikt	ME.10	-	
Backend: STL gebruikt	ME.10	-	
Backend: niet ieder soort object heeft zijn eigen verzameling	ME.10	-	Dus een aparte verzameling voor: Zichtbare Objecten (per locatie), Verborgen Objecten (per locatie), Verborgen Objecten (per vijand), en het bezit van de speler
Third party libraries gebruikt die niet zijn toegestaan	ME.12	-	Wel toegestaan: SQLite3 library, een XML library, een unit test framework, memory leak detection libraries, C++ Standard Library, alleen voor de frontend: STL
Afgeweken van UML diagrammen uit eindopdracht	ME.14	-	
Het versiebeheer is niet via Github Classroom gegaan.	ME.15		De link hiernaar kun je vinden op Brightspace (Afronding->Eindopdracht)
Het is niet mogelijk een dungeon te genereren op basis van kasteelruine.xml	R.7.1	-	De file moet ingelezen en geparsed worden. Wat er in het spel gepresenteerd wordt moet kloppen met de xml file.

	Het is niet mogelijk een random dungeon te genereren op basis van de kerkersendraken.db	R.7.2	-	
	Grote delen van de eindopracht zijn afwezig	R.3 + R.4 + R.5 + R.7 + R.8 + R.11	-	Er moet zeker aanwezig zijn: Backend: Locaties+Objecten+Vijanden Frontend: Speler+Spel Keuze+Gevechten
	Je kunt niet **alles** in je code uitleggen.			ChatGPT, je slimme buurman, het internet, er zijn een hoop bronnen waar je uit kunt putten. Dat is allemaal prima, als je maar weet wat je aan het doen bent. Je moet daarom ook alles kunnen uitleggen op het assessment. "Dat weet ik niet meer, dat is weggezakt, want ik maakte deze opdracht 3 weken geleden" is dan ook geen geldig excuus.
Bonus:				
	(Unit) tests voor functionele requirements		+0.5	De meest significante delen van de code moeten getest worden.
	Template classes gemaakt en gebruikt		+0.5	
	IO-streams en manipulators goed toegepast		+0.5	
	Speltoestand kan worden opgeslagen dmv een extra beschikbare speler actie, en kan bij het starten van de applicatie ook weer helemaal correct worden ingelezen.		+1	
	De frontend en backend zijn low-coupling	ME.13	+0.5	Als er bijv. een facade pattern wordt gebruikt is dat prima, maar andere low-coupling manierer zijn ook goed. Je moet wel kunnen uitleggen waarom het low-coupling is.
Malus:				
	Niet overal (waar nodig) de rule of five toegepast		-0.5	
	Niet overal const correct		-0.5	
	Memory Leaks aanwezig	ME.4	-1 per leak	Wordt niet fout gerekend als het gaat om false positives.
	Casts gebruikt die niet C++ style casts zijn	ME.6	-1	
	Niet gebruik gemaakt van <random>	ME.7	-1	
	Geen session log geïmplementeerd	ME.9	-1	
	Frontend: Nergens smart pointers gebruikt	ME.11	-2	
	Frontend: Wel smart pointers gebruikt maar niet de juiste		-0.5	shared_ptrs gebruikt waar je unique_ptrs had moeten gebruiken

	Frontend: Nergens containers, iterators, of algoritmes gebruikt		-1	Al deze drie dingen moeten gebruikt zijn, anders een minpunt.
	Elke actie die niet geimplementeerd is	R.10	-1 per actie	
	Aan het eind van een spel is er geen optie om opnieuw te beginnen	Einde Spel	-1	
	Leaderboard ontbreekt bij het einde van het spel	Einde Spel	-1	
	Elk deel van de opdracht dat ontbreekt en hierboven nog niet genoemd is.		-1 per deel	
	Geen exception handling toegepast		-0.5	
	Code is niet exception safe		-0.5	
	Lelijke code		-1	

Je begint met het cijfer **7**.

Per onderdeel kunnen er punten worden afgetrokken of bijgeteld. Het maximum aantal punten dat er kan worden afgetrokken of bijgeteld per onderdeel staat in de kolom **punten**. Maar de docent mag beslissen om daar onder te gaan zitten (of zelfs 0 punten af te trekken of er bij te tellen).

Als er een **knock-out** niet gehaald is, dan is het cijfer een 1, en wordt er niet meer gekeken naar andere onderdelen.

Eis # in pdf kan refereren naar het ***Minimum Eisen*** hoofdstuk (ME), of naar de **requirements** hoofdstukken (R) in het eindopdracht document.

VOORBEELDCODE

RANDOM

De <random> header declareert een aantal classes voor het genereren van random getallen. Grofweg zit het zo in elkaar:

- Je zou één exemplaar van een random engine in je programma moeten hebben. Normaliter gebruik je de std::default_random_engine. Er zijn er meer, maar die gebruik je alleen wanneer je expert bent op dit gebied en speciale eisen hebt.
- Die engine initialiseer je met een std::random_device, die met hulp van je operating systeem een goede random seed kan leveren.
- Elke keer wanneer je een random getal nodig hebt, gebruik je een distribution object, dat als een soort adapter op de engine past. De ruwe random data uit de engine wordt op een technisch verantwoorde manier omgezet naar een gewenste verdeling.
  Meestal gebruik je een uniforme verdeling, met andere woorden: de kans op elk getal is even groot. Er zijn ook distribution classes voor normale verdeling en dergelijke. Die komen van pas wanneer je statistisch verantwoorde simulaties en analyses wil doen.
  Voor integers is er de std::uniform_int_distribution, een template class die je als template type het feitelijke integer type meegeeft, bijvoorbeeld “int”. Voor floating point getallen is er de std::uniform_real_distribution, die vergelijkbaar werkt. Je gebruikt dan float of double als template type.

Hieronder is een code voorbeeld van hoe je random dobbelsteenworp kunt genereren. Let op dat dit slechts een voorbeeld is, jij zult de verschillende delen uit deze code zelf een goede plek moeten geven in jouw code.

```
#include <iostream>
#include <random>

int main() {
    // Initialisatie. Gebruik slechts één engine in je programma!
    std::random_device device;
    std::default_random_engine engine {device()};
    // Een paar random integers
    std::uniform_int_distribution<int> dist {1, 6}; // dobbelsteen
    for (int i = 0; i < 20; ++i) {
        std::cout << dist(engine) << ' ';
    }
    std::cout << '\n';
}
```

SQLITE3

Vind en download de library zelf. Dit is een C-library, dus je zult de resources uit die library correct moeten managen. **Dat betekent gegarandeerd en correct opruimen of afsluiten!** Voor meer informatie over SQLite3-functies, zie de officiële documentatie: https://sqlite.org/c3ref/funclist.html

Hieronder is een code voorbeeld van hoe je informatie uit de database kan halen. Let op dat dit slechts een voorbeeld is, jij zult de verschillende delen uit deze code zelf een goede plek moeten geven in jouw code.

```
//Openen van een database file
sqlite3* db = nullptr;
if (sqlite3_open_v2("planetexpress.db", &db, SQLITE_OPEN_READONLY, nullptr) != SQLITE_OK) {
    sqlite3_close_v2(db);
    std::cerr << "error opening database file\n";
}


//Een SQL statement voorbereiden
sqlite3_stmt* stmt1 = nullptr;
const char* query1 = "SELECT ID,inhoud,bestemming FROM pakketjes";
if (sqlite3_prepare_v2(db, query1, -1, &stmt1, nullptr) != SQLITE_OK) {
    std::cerr << sqlite3_errmsg(db) << std::endl;
}


//Een SQL statement voorbereiden met gebruik van parameters in de query
// waar je variabelen aan kunt binden
sqlite3_stmt* stmt2 = nullptr;
const char* query2 = "SELECT inhoud,bestemming FROM pakketjes WHERE ID = ? ";
if (sqlite3_prepare_v2(db, query2, -1, &stmt2, nullptr) != SQLITE_OK) {
    std::cerr << sqlite3_errmsg(db) << std::endl;
}
// vul de waarde 17 in voor het eerste vraagteken
if (sqlite3_bind_int(stmt2, 1, 17) != SQLITE_OK) {
    std::cerr << sqlite3_errmsg(db) << std::endl;
}


//SQL-statement 1 uitvoeren
while (sqlite3_step(stmt1) == SQLITE_ROW) {
    int id = sqlite3_column_int(stmt1, 0);
    std::string inhoud = reinterpret_cast<const char*>(sqlite3_column_text(stmt1, 1));
    std::string bestemming = reinterpret_cast<const char*>(sqlite3_column_text(stmt1, 2));

    std::cout << id << ": " << inhoud << " : " << bestemming << std::endl;
}
std::cout << std::endl;

//SQL-statement 2 uitvoeren
while (sqlite3_step(stmt2) == SQLITE_ROW) {
    std::string inhoud = reinterpret_cast<const char*>(sqlite3_column_text(stmt2, 0));
    std::string bestemming = reinterpret_cast<const char*>(sqlite3_column_text(stmt2, 1));

    std::cout << "17: " << inhoud << " : " << bestemming << std::endl;
}


//Sluiten van de database file
sqlite3_close_v2(db);
```