I-SEB 18/19 S2 Deeltijd

Beroepsproduct 2: Game ontwikkelen

BEROEPSPRODUCT 2

COURSE naam	SEB
Studiejaar	2018-2019
Opleiding	HBO-ICT deeltijd

Korte inleiding

In de afgelopen weken heb je de belangrijkste principes rond objectgeorienteerd programmeren geleerd en toegepast in een flink aantal kleine en middelgrote opdrachten. Nu kun je deze principes toepassen in een grotere opdracht: een game. Je gaat daarbij gebruik maken van een aangeboden “game engine”: een hoeveelheid code die je helpt om met relatief weinig werk een interessant spel in elkaar te zetten.

Hier staat beschreven wat de bedoeling van de opdracht is, hoe je deze geacht wordt aan te pakken, en aan de hand van welke criteria het resultaat beoordeeld zal worden.

Aanpak en op te leveren producten

Het beroepsproduct is het product van de samenwerking tussen **twee personen** (bij een oneven aantal personen in de klas kan de docent toestemming geven voor een samenwerking met drie personen, waarbij de omvang en/of complexiteit van het totale project (uiteraard) wat groter moet zijn).

Alle teamleden worden geacht een evenwichtige bijdrage te leveren in alle fases van de opdracht (analyse, ontwerp en implementatie). **Het is dus nadrukkelijk *niet* de bedoeling om taken zo te verdelen dat de het ene teamlid de ontwerpen maakt en de ander de code schrijft!**

Analyse (functioneel ontwerp)

Zoals elk softwareproject dat wat groter is, begin je met een analyse van wat je wil gaan maken. In eerste instantie bedenk je in heel grote lijnen wat voor spel je wil gaan maken. Indien gewenst bespreek je je ideeen met je docent, zodat deze je eventueel kan helpen in te schatten of het een realistisch idee is.

Als je het idee hebt dat het in principe haalbaar zou kunnen zijn maak je een *functioneel ontwerp (FO)*. In dit document verwachten we minimaal de volgende zaken:

- De naam van je spel.
- Als je het spel baseert op een bestaand spel: de naam van dat spel en een verwijzing naar een bron waarin dit spel beschreven wordt.
- Het doel van het spel.
- Een korte omschrijving van de wereld waarin het spel speelt.
- Het perspectief waarin de speler de wereld ziet (van boven of van opzij: andere perspectieven zijn met de gegeven engine erg lastig te maken).
- Een beschrijving van de acties die de speler kan uitvoeren en via welke besturing dit wordt gedaan (toetsen, muis, ...).
- Een beschrijving van de objecten en obstakels die in het spel voorkomen, inclusief het gedrag van deze objecten (het gaat hier dus zowel om andere actieve figuren als om bijvoorbeeld powerups).
- Een beschrijving van de start en het eind van het spel.
- Een beschrijving van de overige elementen (formulieren, dashboard, menu's) die in het spel een rol spelen.
- Schermschetsjes inclusief viewport (indien je er een gebruikt). Je mag het woord “schetjes” hier heel letterlijk nemen: een foto van een potloodschets kan voldoen, maar een volledig uitgewerkte Photoshopafbeelding ook. De keuze is aan jullie.
- Welke delen van het spel je in elk geval wil gaan maken, en welke delen optioneel zijn (die kun je maken als je voldoende tijd blijkt te hebben).

Een functioneel ontwerp zou in het ideale geval door twee verschillende teams opgepakt kunnen worden en functioneel tot exact dezelfde game leiden. Dit ideale geval is eigenlijk niet haalbaar, maar we verwachten van je dat je in elk geval flink wat zaken duidelijk maakt in je FO, zodat je tijdens het maken van je technisch ontwerp en de implementatie zo weinig mogelijk functionele beslissingen meer hoeft te nemen.

Uiteraard is het FO een professioneel document met voorblad, paginanummers, hoofdstuknummering en een inhoudsopgave, een heldere structuur en zonder taalfouten. Het is bovendien een enkel document met alle bijlagen (zoals schermschetsen) erin opgenomen, en dus geen verzameling van allemaal losse bestanden.

Je kunt het FO al voor de deadline voorleggen aan de docent. Dit is aan te raden. Deze geeft een van de volgende oordelen:

1. Het is onvoldoende (docent geeft aan wat je nog moet doen ter verbetering).
2. Het is onvoldoende, maar biedt voldoende basis om naar de volgende fase te gaan. Dus wel verder gaan maar nog wel wat zaken verbeteren aan het FO.
3. Het is in deze vorm voldoende: je weet dat je hier niets meer aan hoeft te verbeteren voor een voldoende. Als je achteraf nog wijzigingen aanbrengt, kan dat. Maar zorg ervoor dat de kwaliteit daardoor niet achteruit gaat.

Mocht je nu weinig inspiratie hebben om zelf een spel te bedenken. Kijk dan eens naar de afbeelding VoorbeeldGame bij bestanden. Wellicht brengt je dat op ideeen.

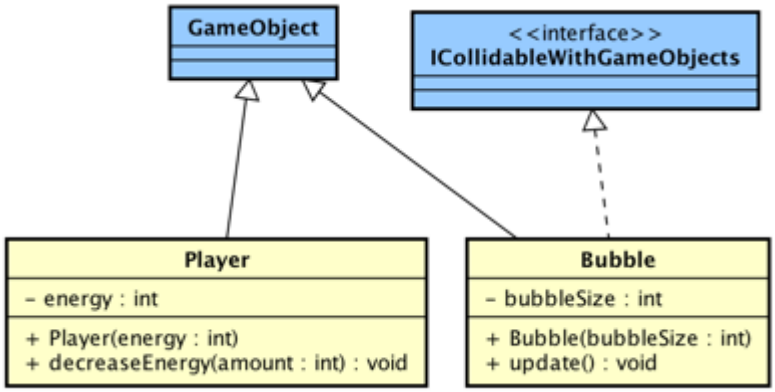
Ontwerp (technisch ontwerp)

Nadat je FO goed is, maak je het technisch ontwerp. Dit bestaat uit:

Een klassendiagram

Een UML-klassendiagram waarin alle klassen en interfaces die je zelf maakt zijn opgenomen. Van elke klasse geeft je aan welke instantievariabelen en methodes je nodig denkt te hebben en geef je aan of ze *public,private* of *protected* zijn. Uiteraard geef je alle overerving en implementatie van interfaces aan.

Daarnaast neem je in het klassendiagram op van welke in de game engine aanwezige klassen je overerving toepast en welke interfaces uit de Game Engine je implementeert. Het is niet nodig de klassen en interfaces uit de game engine volledig in je diagram over te nemen: alleen de naam overnemen is voldoende (geef wel aan dat deze uit de Engine komen, door bijvoorbeeld een andere kleur te gebruiken). Een fictief en klein voorbeeld:



Een korte toelichting

Beschrijf in een toelichting kort waar elke klasse en interface voor dient en hoe ze met elkaar samenhangen. Beschrijf ook (indien relevant) welke klassen en interface je in elk geval gaat bouwen en welke optioneel zijn (als je voldoende tijd blijkt te hebben).

Tip: een veelgemaakte fout in het klassendiagram is dat er van uit wordt gegaan dat objectenlijsten van andere objecten bij moeten houden: soms is dit inderdaad nodig, maar vaak niet: een player kan bijvoorbeeld kogels-objecten maken, maar hoeft ze niet in een ArrayList bij te houden, tenzij de player achteraf nog iets met de kogels moet kunnen doen. Ook hoef je over het algemeen in je wereld*-klasse geen lijst van de objecten bij te houden: dat doet de klasse GameEngine namelijk al.*

Ook het TO kun je voorleggen aan de docent. Deze geeft dan een van de volgende oordelen:

1. Het is onvoldoende (docent geeft aan wat je nog moet doen ter verbetering).
2. Het is onvoldoende, maar biedt voldoende basis om naar de volgende fase te kunnen. Dit betekent concreet dat je wel alvast naar de volgende fase kan, maar nog wel wat zaken moet verbeteren aan het TO.

3. Het is in deze vorm voldoende: je weet dat je hier niets meer aan hoeft te verbeteren voor een voldoende. Als je achteraf nog wijzigingen aanbrengt, kan dat. Maar zorg ervoor dat de kwaliteit daardoor niet achteruit gaat.

Uiteraard is het TO een professioneel document met voorblad, paginanummers, hoofdstuknummering en een inhoudsopgave, een heldere structuur en zonder taalfouten. Het is bovendien een enkel document met alle bijlagen (zoals het klassendiagram) erin opgenomen, en dus geen verzameling van allemaal losse bestanden.

Mogelijk kom je tijdens het maken je TO tot de conclusie dat je wijzigingen wil maken in je FO (bijvoorbeeld omdat je bepaalde functionaliteiten toch anders wil dan je dacht) kan dat natuurlijk. Maar waak over de kwaliteit.

Implementatie

Nadat je het FO en TO gemaakt hebt kun je beginnen met programmeren!

Mogelijk kom je tijdens de implementatie tot de conclusie dat je wijzigingen wil maken in je FO (bijvoorbeeld omdat je bepaalde functionaliteiten toch anders wil dan je dacht) en/of TO (als je bepaalde zaken anders wil implementeren dat je ontworpen hebt). Dat kan natuurlijk. Maar waak over de kwaliteit. Overleg zo nodig met de docent.

Benodigdheden

Op deze plaatst in onderwijs online zijn de benodigdheden voor de uitvoering opgenomen bij bestanden:

De Engine: Engine-1617.zip. Wanneer je deze uitpakt ontstaat er een directory-structuur beginnend met *'OOPG-ZetHierNaamVanJeSpel'*

De volledige API-documentatie en een klassendiagram van Engine

ICA-controlekaart (die ken je waarschijnlijk al, maar voor de volledigheid toch ook hier opgenomen).

Beoordeling en beoordelingsformulier

Het functioneel ontwerp en technisch ontwerp kunnen al eerder worden beoordeeld als je dat wenst (de docent kan op dat moment al aangeven dat ze van voldoende niveau zijn). De implementatie en eventueel nog niet beoordeelde FO en TO, wordt tijdens een assessment beoordeeld. In principe krijgen de teamleden hetzelfde cijfer, tenzij de docent een reden ziet om hier van af te wijken (bijvoorbeeld als de hoeveelheid of complexiteit van het werk niet evenredig verdeeld is). De docent hanteert de onderstaande minimumeisen om tot een cijfer te kunnen komen.

Aanpassingen aan de Engine

Soms is het voor het spel dat je probeert te maken handig of noodzakelijk om wijzigingen in de Engine te maken. Dit is onder voorwaarden toegestaan, maar het verdient altijd de voorkeur om na te gaan of je de gewenste wijzigingen niet kunt aanbrengen in je eigen code (dus zonder ze rechtstreeks in de engine aan te brengen, bijvoorbeeld door van een klasse die je wil aanpassen te erven en daar de aanpassingen te doen). Vraag je docent eventueel om hulp.

Als het nodig is om zaken in de engine te wijzigen, dan gelden daarvoor de volgende regels:

- Je moet elke wijziging bijhouden in een document dat je inlevert (dit mag een tekstbestand in je code zijn). Houd bij in welke klasse je welke methode hebt gewijzigd, wat de wijziging was, en waarom de wijziging nodig was.
- Het wijzigen van bestaande functionaliteiten in de engine is *niet* toegestaan.
- Het maken van een nieuwe implementatie van bestaande functionaliteit is *niet* toegestaan.
- Functionaliteiten toevoegen is toegestaan, mits het niet kan via je eigen code (zie tekst boven deze opsomming).
- Het verbeteren van bugs in de engine is toegestaan. Geef deze bugs (en je oplossing) zo snel mogelijk door aan je docent.

Minimumeisen

- Het FO is verzorgd (zie ook de ICA-controlekaart) en bevat voldoende informatie om het TO en de realisatie op te baseren.
- Het TO is verzorgd (zie ook de ICA-controlekaart), bevat voldoende informatie om de realisatie op te baseren en is in lijn met het FO.
- Alle zelfgebouwde publieke methoden en zelf toegevoegde publieke attributen zijn voorzien van documentatie (bij voorkeur met Javadoc, zie bijv. https://nl.wikipedia.org/wiki/Javadoc)
- Er zijn minimaal acht eigen klassen gerealiseerd die voldoende verschillend zijn en die op z'n minst enige functionaliteit bevatten (klassen met vrijwel identiek gedrag dat ook met andere attribuutwaarden te realiseren zou zijn, tellen niet mee, en ook interfaces of volledig abstracte klassen tellen niet mee).
- Er wordt minimaal één van de interfaces uit de engine succesvol toegepast.
- Er wordt overerving toegepast binnen de zelfgebouwde klasses (dus niet alleen overerven van objecten uit de engine).
- Er wordt minimaal eenmaal polymorfie toegepast binnen zelfgebouwde klassen (dus: er is de mogelijkheid van polymorfie ingebouwd, en deze wordt daadwerkelijk gebruikt).
- Klassen en methoden hebben duidelijke verantwoordelijkheden:
    - Naam komt overeen met de taak
    - Namen van attributen dekken de lading
    - Zo weinig mogelijk dubbele code
    - Geen static variabelen, tenzij daar een goede reden voor is

- Alle studenten uit het groepje begrijpen alle code en kunnen deze tijdens het assesment toelichten

Nog wat schrijftips voor het FO

Hieronder nog wat leestips bij het schrijven van je FO. Goed schrijven is herschrijven. Het is prima om eerst eens wat te brainstormen en dingen op te schrijven, maar dus als je minstens verplichte items tijdens de analyse hebt ingevuld, is het tijd je tekst wat te gaan herschrijven, zodat deze ook voor buitenstaanders leesbaar is. En deze een goede hoofd- en substructuur te geven.

Conform de ICA controlekaart moet je ook dit document doelgroepgericht schrijven. Omdat een ander team op basis van je FO tot dezelfde game zou moeten komen is je doelgroep niet per se de docenten, maar je mede student, en dan niet per se. Je document moet ook enigszins zelfbeschrijvend zijn, dus begin met de context van OOPD, en de opdrachtbeschrijving en daarbijvereiste zaken als minimum aantal klassen en (eigen) polymorfie.

Hoofdstructuur

Zorg dat je (FO) document leesbaar is en geef deze structuur. Net als elk goed verhaal kun je je document opdelen in drie stukken:

- introductie (context, leeswijzer e.d.)
- kerntekst (spelbeschrijving)
- conclusie/slot (niet gerealiseerde zaken, mogelijke verbeterpunten, leermomenten)

NB: Gebruik hierbij 'introductie', 'kern' en 'slot' niet als hoofdstuknamen in je FO, maar bepaal zelf goede hoofdstukken. De introductie is 'H1. Inleiding' maar de rest is afhankelijk van je game. De spelbeschrijving is een aantal hoofdstukken, deze bevatten de vereiste punten als besturing, schermschetsen, gamewereld, maar gebruik dit dus ook niet als hoofdstuknamen, maar kies liever hoofdstuk namen op basis van je game. Je structuur beschrijf je al in een stukje leeswijzer in je inleiding. Dit is een tekstuele variant van je inhoudsopgave, zodat een lezer eventueel snel naar een voor hem interessant deel kan doorbladeren. Je leeswijzer is echter geen bullet lijst, want dan voegt het niks toe t.o.v. je inhoudsopgave. Extra in een leeswijzer is bijvoorbeeld dat je verbanden tussen hoofdstukken kunt aangeven.

Geef ook substructuur en gebruik verwijzingen

Zorg naast algemene hoofdstukstructuur ook voor wat structuur binnen hoofdstukken via subkopjes/subsecties (niet meer dan 3 niveaus diep), en herhaal jezelf of anderen niet, maar gebruik interne en externe verwijzingen (=externe referenties/bronnen; volgens APA, neem bijvoorbeeld in ieder geval een referentie naar de gebruikte OOPG GameEngine (github) en naar je eigen TO).

Schrijf niet één lap woorden en hou het ook niet bij enkel todo-/bullet-lijstjes, maar wissel tekst, lijstjes en afbeeldingen af.

Over afbeeldingen: neem schermschetsen NIET pas aan het eind van je FO op. Een plaatje zegt meer dan 1000 woorden. Dus plaats deze meer aan het begin als je je spel introduceert. Geef met tekst wel een bijbehorende toelichting, waarin je de schermonderdelen ook uitlegt en namen geeft, die je in TO en Code weer terug laat komen. Kleine variaties in het scherm kun je tekstueel toelichten, maar als je aparte menu's hebt of bijvoorbeeld een level einde die anders is heb je aan één schermschets niet genoeg