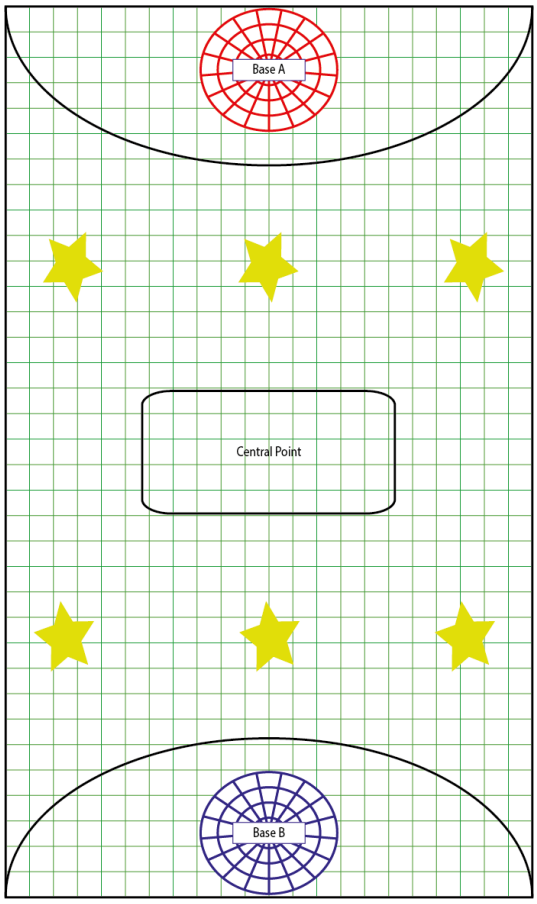
Big Little Wargame

Volledig adviesrapport



Team: Fatal Error C1003

Joost Wagensveld	1664713
Remco Nijkamp	1657833
Zehna van den Berg	1662506
Waila Woe	1615856

Docent: Gerald Ovink, Arno Kamphuis, Joost Schalken-Pinkster, Wouter van Ooijen

Versie: 1.0
Datum: maandag 9 november 2015

Samenvatting

In dit verslag is het project van het 2e jaar blok 1 van Technische Informatica vastgelegd. De opdracht van het project is een game, geprogrammeerd in C++ met gebruik van de SFML bibliotheek. De opdrachtgever is de Hogeschool Utrecht.
De opdracht is begonnen met het zelf bedenken van een computergame en het ontwerpen van een Game Design Document. Hierna is een klassendiagram gemaakt en daaruit is de basis van de gamecode ontstaan. Verder wordt kort toegelicht hoe er is gepland voor dit project.
Tijdens dit project zijn meerdere soorten software gebruikt. Deze hebben het ontwikkelproces vereenvoudigd.
Voor het ontwerpen van de game is gebruik gemaakt van het Game Design Document. Deze wordt kort toegelicht in dit verslag. Vervolgens is uit het Game Design Document een klassendiagram gemaakt. Ook deze wordt uitvoerig doorgenomen.
Uiteindelijk is met behulp van het klassendiagram de code van het spel ontstaan. In hoofdstuk drie worden de klassen toegelicht met hun methoden en attributen.

Inhoudsopgave

1 Inleiding

In dit verslag wordt verteld hoe de opdracht, het maken van een game in de programmeertaal C++, tot stand is gekomen en vooral ook hoe de game in elkaar zit.
Het spel is in feite een turn-based strategy game, maar wat het spel zo speciaal maakt is dat het meerdere mogelijkheden heeft tot het winnen van het spel.
Allereerst is er natuurlijk de mogelijkheid tot het vernietigen van de basis van je tegenstander. De tweede mogelijkheid is het voor langere tijd bezetten van een centraal punt, namelijk de Holy Grail. Hiermee doe je als speler per beurt schade aan de tegenstander wanneer een unit naast de Holy Grail staat. Zodra de tegenstander geen levens is meer over heeft, heb je uiteraard ook gewonnen.

De naam van het team komt van de vele foutmeldingen die tijdens het project langs zijn gekomen. De uitleg van Microsoft van deze foutmelding is:

> Fatal Error C1003 - error count exceeds number; stopping compilation
> Visual Studio 2015
> Errors in the program are too numerous to allow recovery. The compiler must terminate.(Microsoft, 2015)

Doel van dit verslag is als overdracht document te fungeren. Hierin kunnen gemaakte keuzes gevonden worden en zijn in het verleden gemaakte fouten opgenomen. Op deze manier worden deze hopelijk niet weer opnieuw gemaakt.

Er wordt teruggekeken op het Game Design Document wat aan het begin van het project is opgesteld en veranderingen die ten opzichte van dit document zijn doorgevoerd.
In het verslag wordt de globale structuur van het spel doorgenomen en op sommige zaken dieper ingegaan. De verschillende klassen die in het spel voorkomen worden behandeld aan de hand van het klassendiagram.

Tot slot zal worden teruggekeken op het project en zal advies worden gegeven over wat in het vervolg anders of beter zou kunnen en welke aanbevelingen er voor verdere vordering van het spel zijn.

2 Functioneel ontwerp

In dit hoofdstuk zal het Game Design Document kort worden behandeld. Hierin wordt vooral uitgelegd welke keuzes gemaakt zijn om het spel zo eerlijk mogelijk te laten verlopen en waarom er specifiek voor die keuzes zijn gekozen.

2.1 Samenvatting Game Design Document

In het Game Design Document worden de belangrijkste aspecten van het spel besproken. Er is gekozen voor een top-down view, turn-based game. Dit betekent dat om de beurt slechts één speler de beurt krijgt om zijn acties uit te voeren. Het spel begint in het menu waar men het spel kan starten of kan afsluiten. Tevens is het mogelijk vanuit dit menu het geluid aan dan wel uit te zetten. Het daadwerkelijke spel begint nadat de startknop van het menu is aangeklikt. Er verschijnt dan een speelveld waarop de actie zich afspeelt, en aan de zijkant een HUD met de nodige informatie voor de spelers. De basissen van de beide spelers zijn tegenover elkaar geplaatst aan de buitenzijde van het speelveld. In het midden van de het speelveld ligt het centrale punt, ook wel de Holy Grail genoemd, waarmee schade aan de tegenstander kan worden aangericht om het spel te kunnen winnen.

Aan het begin van het spel zullen beide spelers een startkapitaal krijgen. Elke speler krijgt ook een standaard set van units tot zijn beschikking. De speler kan van het geld dat hij verdient in het spel en van het startkapitaal verschillende units kopen. De speler kan zijn eigen units aansturen, zo kan de speler bijvoorbeeld een unit laten lopen, resources verkrijgen of een vijandige unit of gebouw aanvallen. Er zijn verschillende soorten units beschikbaar in het spel. Elk soort unit heeft zijn eigen unieke kracht die kan worden uitgevoerd. Om nieuwe units te kunnen maken is een gebouw vereist genaamd Factory. Geld kan verzameld worden wanneer een unit een resource point aanvalt. De resource points zullen op vaste plekken in het speelveld staan die beschikbaar zullen zijn voor beide spelers. Geld uit de resource points kunnen per unit per speelbeurt slechts één keer worden verkregen, anders zou de balans uit het spel raken.
De winconditie is bereikt wanneer de basis van een tegenstander is vernietigd of wanneer een speler zijn tegenstander genoeg schade heeft aangedaan door lang genoeg de Holy Grail te bezetten(Nijkamp, Wagensveld, van den Berg, & Woe, 2015).

2.2 Gemaakte keuzes

Veranderingen
Ten opzichte van het Game Design Document dat aan het begin van het project is opgesteld, zijn een aantal veranderingen doorgevoerd die ertoe hebben geleid dat de game op sommige vlakken iets afwijkt van het oorspronkelijke idee.

Zo was het eerst de bedoeling dat units zichzelf weer zouden kunnen helen als ze zich vlakbij de eigen headquarter bevinden. Er is uiteindelijk voor gekozen dit niet meer te doen om het spel wat sneller te laten verlopen. Bovendien zouden spelers dan een stuk defensiever gaan spelen en dat was juist niet de bedoeling, de actie en dynamiek waren juist iets wat ons een goed idee leek.

We hebben ook gekozen om een speciale unit in de zetten die alleen gebruikt kan worden om geld te verkrijgen uit een resource. Na wat overleg is ervoor gekozen toch alle units deze mogelijkheid te geven, voornamelijk omdat de spelwereld vrij klein is gebleven, waardoor dit vrij lastig zou worden.
Ook was het de bedoeling dat units meerdere aanvallen zouden hebben. Er is voor gekozen eerst te werken aan één aanval per unit. Tegen het einde van het project kwam de conclusie dat er eigenlijk niet veel tijd meer over was hier nog meer acties en aanvallen aan toe te voegen. Bovendien speelde het spel op dit moment al erg lekker en doordat er al verschillende soorten units aanwezig waren met een eigen walklimit en attackrange, zat er al genoeg diversiteit in het spel om leuk en uitdagend speelbaar te zijn.

Hieronder worden nog even een paar belangrijke objecten uit de game apart genomen en uitgelegd wat hun rol en mogelijkheid in het spel is.

Central point
De Central Point bevindt zich precies in het midden van het speelveld. Hierop komt de Holy Grail te staan. Er is voor gekozen de Holy Grail in het midden te plaatsen om het spelverloop zo eerlijk mogelijk te maken voor beide spelers. Het aantal stappen die gemaakt moeten worden met de unit naar de Holy Grail is dan namelijk gelijk voor beide spelers.

Resource point
In het gehele spel zijn zes resource points te vinden. Elk resource point heeft een opslag van 800 gold. Elke unit kan geld van de resource point verzamelen door hier een aanval op te doen terwijl hij zich direct naast de resource bevindt. Per keer dat de resource aangevallen wordt krijgt de speler 100 gold opgeteld bij zijn kapitaal. Daarna zal de beurt van de unit die aangevallen heeft voorbij zijn.
De resource point is leeg wanneer de aantal gold op 0 staat. Om het spel een beetje dynamischer te maken is ervoor gekozen om nog wel geld te laten verkrijgen uit de resource als deze leeg is, maar dit zal slechts 20 gold zijn, in plaats van de eerdere 100 gold. De speler moet dus 5 keer vaker een resource point aanvallen om hetzelfde geld te krijgen, waardoor het vaak interessanter zal zijn om de units te verplaatsen naar een andere resource.

Hoofdbasis
De hoofdbasis is een van de belangrijkste gebouwen in het spel. Wanneer de levenspunten van de basis van de huidige speler op 0 staat heeft de tegenspeler gewonnen. De levenspunten omlaag halen van de hoofdbasis van de tegenstander kan worden gedaan door deze met een unit aan te vallen. Beide spelers moeten een manier bedenken om hun eigen hoofdbasis te beschermen. Dit kan door bijvoorbeeld defensief te spelen of juist offensief en op die manier de eerste klappen uit te delen aan de tegenstander.

Units
Alle soorten units hebben dezelfde eigenschappen, namelijk lopen en aanvallen. Maar elk soort unit heeft een eigen walklimit, een eigen attackrange, eigen hitpoints en ook eigen attackpoints. Zo zal de ene unit meer levens hebben dan de andere, terwijl een andere unit juist weer sterker kan zijn, verder kan lopen of vanaf grote afstand kan aanvallen. Per beurt kan een unit slechts één aanval of resource attack doen.

Factory
Het spel wordt gestart met voor iedere speler een eigen Factory. De Factory wordt gebruikt om units aan te maken tegen aanbetaling. Elk soort unit heeft een ander prijskaartje. Hierdoor moet de speler goed nadenken of het wel verstandig is om geld uit te geven aan een unit. De speler kan er dus ook voor kiezen om geld te sparen om later een duurdere en sterkere unit te kopen.

3 Technisch Ontwerp

In dit hoofdstuk worden de technische aspecten behandeld. De architectuur van de game zal voornamelijk gaan over de Game klasse, aangezien dit de belangrijkste klasse is van onze game. Verder zal er ook ingegaan worden op het klassendiagram. Daar worden de belangrijkste zaken uitgelegd van elke klasse.
Voor de realisatie van dit project is gebruik gemaakt van de informatie die stond in het boek 'SFML Game Development'. Uit dit boek zijn enkele hoofdstukken gebruikt als basis voor de code. Er zijn aanpassingen gedaan op deze basiscode voor een goede implementatie binnen onze game(Haller & Hansson, 2013).

3.1 Architectuur

De architectuur van ons project gaat uit van de Game klasse als middelpunt. Deze roept de methodes aan van de objecten wanneer ze iets moeten doen. Waar meerdere objecten van één klasse zijn, worden ze in een std::vector ondergebracht met een std::unique_ptr, om geheugenlekken te voorkomen. Alle objecten in het spel erven van de algemene superklasse GameObject die de sprite en de texture van het object bevatten. De methodes hierin zorgen dat animaties verwerkt worden in een update methode en de sprite op het scherm worden getekend. De klassen die hier direct van erven zijn PlayerObject, Button, Terrain, Holy Grail en Resource. Button heeft attributen zodat gekeken kan worden of er op een button geklikt wordt en als dit het geval is, welke acties dan moeten worden uitgevoerd. Er zijn drie subklassen van button: MenuButton, PlayerButton en UnitButton, waarbij UnitButton voor elk type unit zijn eigen klasse heeft. PlayerObject is de superklasse van alle speler specifieke objecten. Deze behoren tot een speler en kunnen vernietigd worden. Hier gaat de splitsing richting Units en Buildings. Ieder heeft de uiteindelijke subklassen onder zich van de objecten die daadwerkelijk in de game gebruikt worden.
De Game besteedt de inputafhandeling uit aan de InputHandler en het inlezen van tekst bestanden aan ReadText.
De CommandQueue is bekend bij de Game. Er is binnen de code maximaal één CommandQueue object. Dit object wordt doorgegeven aan de desbetreffende methodes als hier informatie aan moet worden toegevoegd.
Tot slot worden de speler objecten apart binnen de game gealloceerd. Hiernaast zijn er ook voor de Music en Sound klasse ieder een object binnen de game.

3.2 Klassendiagram

Hieronder volgt per klasse of groepje van klassen een korte beschrijving.
Goed om nog even te benadrukken is dat de klasse Game het centrale punt van het spel is geworden. Hier ligt verreweg de meeste intelligentie en de klasse fungeert als een soort middelpunt tussen verschillende klassen. Achteraf was dit niet de beste en meest nette manier, maar in dit project werkt het naar behoren.

Game

De Game klasse bevat alle spelelementen die nodig zijn om het spel te spelen. Alle objecten waarvan vooraf geen vast aantal is vastgesteld worden ondergebracht in std::vector lijsten. Hiernaast bevat de klasse ook meerderen booleans die bijhouden in welke toestand de game zich bevindt. Aan de hand van deze attributen wordt in de code bepaald welke acties er op dat moment uitgevoerd moeten worden en op welke objecten dit invloed zal hebben. De grote hoeveelheid methoden die zich in de Game bevinden komen voort uit de nood om in verschillende containers van GameObjecten sequentieel data moet worden bewerkt of verkregen. Echter zullen er ook methodes zijn die maar in een enkele container of attribuut data hoeven te wijzigen. Deze zijn op het moment alsnog in de Game klasse gebleven wegens tijdgebrek om dit ergens anders onder te brengen. Bovendien zou dit de structuur van het programma zodanig beïnvloeden dat zodoende op andere plekken weer extra wijzigingen doorgevoerd zouden moeten worden.

GameObject

Dit is een vrij algemene superklasse binnen het project. Alle objecten die zich in het spel bevinden en getekend kunnen worden, dus bijvoorbeeld wel de units, maar niet de muziek, zijn GameObjecten.
Zoals te zien is hebben alle klassen die erven van deze superklasse een positie, zodat ze op een bepaalde positie op het scherm getekend kunnen worden. Tevens hebben ze een eigen sprite. Dit is nodig aangezien zonder een afbeelding niks te tekenen valt. Zoals te verwachten bevat deze GameObject klasse een draw methode om zijn Sprites op het scherm te laten tekenen.
De update methode is ervoor om geanimeerde plaatjes op het scherm te laten verschijnen.
Aan deze methode wordt daarom de tijd meegegeven zodat aan de hand van de tijd een berekening kan worden losgelaten op het plaatje om een mooie animatie mogelijk te maken.

PlayerObject
Deze klasse is een subklasse van het GameObject. Tevens is dit weer een superklasse voor alle klassen die objecten vertegenwoordigen die een speler in zijn bezit kan hebben. Aangezien de spelers een eigen kleur hebben waaraan ze kunnen worden herkend, hebben ook alle Player objecten een kleur. Het is de bedoeling dat de units en de buildings van een speler dood of kapot kunnen gaan. Om die reden is er een attribuut hitpoints aanwezig, die bij kan houden hoeveel levens zo'n Player object nog heeft. Uiteraard kan de waarde van dit attribuut worden opgevraagd met een public getter zodat de Game deze kan gebruiken.
Tot slot kunnen Player objecten schade oplopen, dit wordt dan van hun hitpoints afgetrokken.
Hiervoor is een speciale methode aangemaakt die deze afhandeling mogelijk maakt.

Unit tak:
Weer onder de klasse PlayerObject valt de Unit klasse met zijn subklassen Bomber, Recruit, Scout en Soldier. Deze klasse bevat alle informatie die voor de units nodig zijn, zoals de naam, attackpoints, walklimit, kleur enzovoort. De walklimit bevat het maximaal aantal hokjes dat een unit zich per beurt mag verplaatsen. Onder de attackrange valt het aantal hokjes dat een aanval van een unit bereiken kan. Tevens bevinden zich twee bijzondere attributen in deze klasse die met de huidige turn te maken hebben.
turnWalkLimit houdt per beurt bij hoeveel hokjes van zijn oorspronkelijke walklimit hij nog over heeft om te lopen. turnAttackRange houdt per beurt bij of de attackrange nog groter is dan 0.
Er zijn twee speciale methoden te zien in het klassendiagram die toonaangevend zijn voor de units, namelijk walk en action. Walk maakt het mogelijk een unit te verplaatsen naar een ander hokje op het veld. Action handelt eventuele aanvallen op vijandige units of buildings uit, of verzamelt geld uit een resource.

Building tak
Een andere subklasse van de PlayerObject klasse is de Building klasse, die zelf ook weer als superklasse fungeert voor Headquarter en Factory. Hier geldt eigenlijk een beetje hetzelfde verhaal als voor de units, alleen kunnen buildings zelf natuurlijk niet lopen of aanvallen. De buildings hebben wel een eigen action, zo is de action bij een Factory dat bij selecteren hiervan een klein soort menuutje wordt geopend rechts onderin het scherm, waarin de speler kan kiezen een unit te kopen en deze dus te fabriceren. Bij een headquarter wordt in de action bekeken of deze Headquarter nog hitpoints over heeft. Is dit niet het geval, dan heeft de tegenstander gewonnen en zal de action ervoor zorgen dat de Game weet dat hij een ander scherm moet tonen.

ResourcePoint
De ResourcePoint klasse maakt het mogelijk om resources aan te maken voor in het spel. Een resource is neutraal, dit houdt in dat hij niet in het bezit valt van een speler. Vandaar dat hij niet opgenomen kon worden onder de tak van de PlayerObject, omdat hij dan automatisch bezit zou zijn van een speler.
De resource erft uiteraard wel van de GameObject klasse, omdat het objecten vertegenwoordigt die op het scherm getekend kunnen worden. Als attribuut bevat hij resourceMoney. Hiervan kunnen units geld ophalen waarna de waarde daalt. Hiervoor zijn dan ook methoden nodig om deze interactie mogelijk te maken. GetResourceMoney geeft terug hoeveel geld de resource nog bevat. Dit wordt gebruikt om in de HUD te tonen hoeveel geld de resource nog bevat. De methode getMoney wordt gebruikt om daadwerkelijk geld te verkrijgen uit de resource door een unit, de inhoud van de resource daalt hierna dus ook.

HolyGrail
Ook de Holy Grail kan niet onder worden gebracht bij de Player objecten, omdat het een neutraal punt voorstelt. Als speciale methode heeft deze methode Reckoning. Deze methode wordt na afloop van een speelbeurt aangeroepen. Dit zorgt ervoor dat alle units dit rond de Holy Grail staan schade aanrichten aan de andere speler.

Terrain
Deze klasse speelt een belangrijke rol in het spel. Elk object van de klasse stelt namelijk een hokje voor van het speelveld. In de Game wordt het hele speelveld gemaakt door middel van een vector vol met objecten van deze Terrain klasse. De klasse bevat een attribuut free, dit is een boolean die aangeeft of een Terrain object vrij dan wel bezet is. Dit is heel belangrijk voor het spel. In de Game wordt namelijk gekeken aan de hand van dit gegeven of een unit zich bijvoorbeeld mag verplaatsen naar het geselecteerde hokje. Er zijn dan ook methoden die deze interactie mogelijk maken, namelijk een setter en een getter voor dit free attribuut. Verder zijn er nog belangrijke methoden die de kleur van de vakjes aan kunnen passen en op kunnen vragen. Ook deze informatie wordt in de Game gebruikt om bijvoorbeeld de walklimit van de units aan de speler te kunnen tonen op het veld. Door de vakjes anders te laten kleuren ziet de speler op welke

vakjes hij kan klikken voor een verplaatsing of een aanval. Deze markering wordt niet alleen gebruikt om het de speler makkelijker te maken, maar ook om te kunnen bepalen of een unit zich überhaupt wel mag verplaatsen. Er wordt dus een kleurcheck gebruikt.

ReadText

Bij het opstarten van het spel worden alle units, buildings, resources en de Holy Grail op hun beginpositie geplaatst. Dit gebeurt aan de hand van het inlezen van een tekst bestand deze gegevens bevat. De klasse ReadText is hiervoor verantwoordelijk. Vanuit de Game wordt een aanroep gedaan van de Read methode in ReadText. Door op deze manier te werk te gaan is de code een stuk overzichtelijker geworden. Eerst gebeurde het opbouwen van het speelveld namelijk in de code zelf. Ook kan nu makkelijk een andere startindeling worden meegegeven door een ander tekst bestandje in te lezen.

Button tak

In het klassendiagram is nog een grote tak te vinden die valt onder de GameObject klasse, namelijk de tak van de Button met zijn subklassen. Een object van de Button klasse heeft als kenmerk dat hij aangeklikt kan worden. Vandaar de algemene methode getClicked, die kijkt of een button al dan niet is aangeklikt.
Onder de Button vallen weer 3 subklassen, de UnitButton, de MenuButton en de PlayerButton.
Elk zorgen deze voor hun eigen afhandeling als er een muisklik optreedt.
Zo is er een subklasse EndTurnButton die onder PlayerButton valt en ervoor zorgt dat de beurt van de huidige speler wordt beëindigd en de beurt van de andere speler ingaat.
De UnitButtons hebben als extra nog een attribuut Cost die aangeeft hoeveel het kost om de unit die bij deze button hoort te kopen. Dit kan uiteraard ook worden opgevraagd met een getter.

Player

De klasse Player bevat de informatie die nodig is om de twee spelers aan te kunnen maken. Elke speler heeft namelijk zijn eigen kleur. Hiermee zijn de spelers dan ook te onderscheiden. Bovendien heeft elke speler zijn eigen hoeveelheid punten en geld. Deze informatie is terug te vinden in de attributen van de klasse en er zijn ook methoden aanwezig in de klasse waarmee de punten en het geld kunnen worden opgevraagd en aangepast. Verder houdt een object van de Player klasse bij of hij aan de beurt is en hoeveel units nog kunnen aanvallen of lopen deze beurt. De informatie over hoeveel units nog een actie kunnen uitvoeren wordt opgevraagd door de Game zodat deze ervoor kan zorgen dat dit zichtbaar wordt in de HUD.

InputHandler

Deze klasse zorgt voor de afhandeling van de input van de spelers. De methode processInput is vanuit de Game aan te roepen en zorgt ervoor dat alle input events worden afgevangen. Vervolgens zijn er de private methoden handleKeypress en handleMouseClick aanwezig die op hun beurt weer kijken welke actie ze moeten uitvoeren bij het indrukken van een bepaalde toets of bij een muisklik. Een Command wordt dan gezet en die wordt in de Game verder in afhandeling genomen.

ResourceHolder

Dit is een template klasse waarin meerdere soorten resources opgeslagen kunnen worden die in SFML gebruikt worden. Hij heeft twee loadmethodes, beide met een string als parameter voor de file-locatie waarvan geladen moet worden en een identificatie id (textureID) die aan elkaar gekoppeld worden in een std::map. De tweede heeft nog een extra parameter voor bepaalde resources zoals sf::Music. De resources kunnen vervolgens met een getter opgehaald worden met het textureID als parameter.

Music

In deze klasse staat alles dat nodig is voor het afspelen van muziek in het spel. Niet alleen wordt hier bijgehouden welke muziekfiles in de game aanwezig zijn. Er staan natuurlijk ook methoden in die zorgen dat de muziek kan worden afgespeeld, gepauzeerd en ook het volume kan worden aangepast. Er zijn ook attributen volume en paused aanwezig, deze houden het volume van de muziek bij en of het nummer al dan niet gepauzeerd is.

Sound

Deze klasse lijkt best wel op de Music klasse, maar er zijn toch een paar grote verschillen. Zo kan een sound enkel worden afgespeeld via de play methode en niet worden gepauzeerd. Ook wordt een sound pas ingeladen in het geheugen op het moment dat hij nodig is. Zodra de sounds niet meer direct nodig is wordt de methode removeStoppedSound gebruikt om de sound weer uit het ingeladen geheugen vrij te geven.

CommandQueue

De CommandQueue bevat een FIFO queue met Commands. De Commands worden hierin gepusht als er een actie in de game moet gebeuren. In de game.cpp worden vervolgens alle Commands uit de lijst met een pop gelezen en verwijderd uit de queue. Door middel van een switch statement wordt elke soort Command afgevangen en de bijbehorende actie uitgevoerd. Indien het vereist is, wordt ook de positie van de muis op het moment van invoeren van het Commando aan de desbetreffende methodes doorgegeven.

4 Realisatie

Samenwerken in een project met andere leden is niet altijd makkelijk. Om dit in soepele banen te laten lopen kan men van verschillende hulpmiddelen gebruik maken. Een goede planning is noodzaak bij een project, zo weet iedereen waar hij aan toe is. Ook zijn er programma's beschikbaar voor het uitwisselen van werk. Door zulke programma's te gebruiken, kan ieder project-lid een goed overzicht houden van wat er al is gebeurd. En wat er dus nog moet gebeuren.

4.1 Software

In het komende stuk wordt een korte uitleg gegeven van de software die tijdens het project is gebruikt.

Visual Studio 2015

Visual Studio is gebruikt voor het schrijven van de code van de game. Visual Studio is een programmeerontwikkelomgeving van Microsoft. Het biedt een complete set ontwikkelingstools voor computerprogramma's in diverse programmeertalen. De taal die bij de realisatie van Big Little Wargame is gebruikt is C++. In combinatie met de bibliotheek van SFML, hierover verderop meer.

SFML

Simple and Fast Multimedia Library (SFML) is een cross-platform software ontwikkel bibliotheek, ontworpen als een eenvoudig interface voor verschillende multimedia componenten in computers(Gomila, 2015). Binnen dit project is hier dan ook uitvoerig gebruik van gemaakt

Software Ideas Modeler

Het klassen diagram is tot stand gekomen met behulp van Software Ideas Modeler. In dit programma kan men verschillende diagrammen maken. Verder is het ondervonden als een eenvoudig te gebruiken programma, wat redelijk voor zichzelf spreekt.

Git

Om de code tussen de projectleden uit te wisselen is gebruik gemaakt van GitHub met als software Git Shell(Chacon, 2009). Met deze software kan elk project lid zijn code uploaden naar een online Git repository, waar de rest van de groep deze weer vanaf kan halen. Ook kan de software van Git gebruikt worden om de codes van de verschillende teamleden samen te voegen. Zo wordt het tegelijkertijd werken op verschillende computers een stuk eenvoudiger. Ook zijn de verschillende documenten die voor dit project nodig zijn in de Git repository opgeslagen. Zo kan iedereen erbij en zijn ze uiteindelijk ook terug te vinden voor de klant. Hiervoor is een bepaalde structuur gebruikt die tijdens de hoorcolleges is uitgelegd(Kamphuis, 2015).

Microsoft Office

Voor dit project zijn meerdere programma's gebruikt van het Office pakket. Microsoft Word voor de verslaglegging, PowerPoint voor de presentaties en Outlook voor het onderlinge emailcontact.

Google Drive

Om de documenten van dit project voor ieder project lid beschikbaar te maken, is naast GitHub gebruik gemaakt van de online dienst Google Drive(Google, 2015). Hier kan elk project-lid zijn documenten in kwijt. Hier staan alle documenten die betrekking hebben tot dit project. Documenten die ook voor de klant beschikbaar moeten zijn staan zoals eerder vernoemd ook in de Git repository.

Doxygen

De documentatie van de code is tot stand gekomen door middel van het programma Doxygen. Dit programma kan na een aantal aanpassingen in de code, automatisch een verslag maken van de code(van Heesch, 2008).

4.2 Planning

Voor dit project is gewerkt volgens de agile methode. Om meer specifiek te zijn hebben we gewerkt met Scrum om het project te realiseren.

Scrum

Scrum is een agile methode dat gebruikt wordt bij het opleveren van producten in teamverband. Er worden tijdens de ontwikkelperiodes meerdere sprints geformuleerd van een van tevoren bepaalde lengte. Aan het eind van elke sprint moet een werkende versie worden opgeleverd waar de klant duidelijk verbetering kan zien ten opzichte van het product sinds de vorige sprint. De teamleden overleggen voor de sprint begint over wat er deze sprint moet en kan worden bereikt. Ook wordt er overlegd hoeveel tijd hier voor moet worden uit getrokken. Als de sprint qua tijdverstrek is afgelopen, wordt er gekeken welke doelen er bereikt zijn. Hier wordt op gereflecteerd en indien nodig voor nieuwe sprints de werkdruk aangepast.
Een voordeel van scrum als ontwikkelmethode is dat er veel overleg wordt gepleegd binnen het team. Hiernaast is er ook constant feedback van de klant of deze tevreden is met de ingebrachte functionaliteit van de sprint of anders toch een andere richting met het product op wil. Het definitieve product zal niet altijd zijn zoals aan het begin van het project is bedacht. Dit heeft als reden dat er tijdens het project veranderingen doorgevoerd kunnen worden. Ook kunnen er in de sprints nieuwe doelen ontstaan.
In de sprintverslagen, die als bijlage zijn toegevoegd, is goed de ontwikkeling van de game door het project heen te volgen.

5 Foutverwerkingen

Tijdens het project zijn we een aantal problemen tegengekomen. Deze problemen worden in dit hoofdstuk beschreven. Er zal worden behandeld hoe de problemen zijn ontstaan en op welke manieren ze zijn opgelost.
Halverwege het project heeft zich een probleem in compatibiliteit geopenbaard tussen de verschillende versies van Visual Studio. Het ging hier om de versies Visual Studio 2013 Update 5 en Visual Studio 2015. Hierdoor ging het werken niet goed. Later bleek ook dat er tussen de projectleden onderling verschillende versies van de SFML bibliotheek in gebruik waren.
In het vervolg is het beter om strakkere afspraken te maken over het gebruik van de bepaalde softwareversies. En ook dient bij een volgend project de bibliotheken die in gebruik zijn tijdens het project, in de Git repository opgeslagen te worden, op die manier gebruikt iedereen hetzelfde, waardoor minder conflicten op zullen treden. Hieronder worden nog een aantal errors in behandeling genomen:

Microsoft Visual Studio 2013-2015 - 0xc000007b error

Het probleem ontstond wanneer men de software Microsoft Visual Studio 2013 ging upgraden naar Microsoft Visual Studio 2015. Na het upgraden werden de volgende foutmelding weergegeven. De foutmeldingen geven aan dat de applicatie niet samen werkt met de Windows besturingssysteem die gebruikt wordt.

- Error
  - 0xc000007b error
- Missende dll bestanden:
  - msvcr100.dll
  - msvcr100d.dll
  - msvcp100.dll
  - msvcp100d.dll
  - msvcr100_clr0400.dll

Het probleem kon worden verholpen door verschillende pakketten te installeren van Microsoft. De volgende pakketten moesten up-to-date zijn om de error 0xc000007b op te lossen:

- Visual Studio 2008 (VC++ 9.0)
- Visual Studio 2011 (VC++ 10.0) SP1
- Visual Studio 2012 (VC++ 11.0)
- Visual Studio 2013 (VC++ 12.0)

Microsoft Visual Studio 2015 – Visual Studio tools 2015

Een ander probleem van Visual Studio was dat bij het runnen van een project een melding werd weergegeven dat verschillende pakketten geïnstalleerd moest worden. De pakketten zouden automatisch geïnstalleerd kunnen worden maar, door een onbekende reden was dat niet mogelijk.
De volgende pakketten waren handmatig geïnstalleerd:

- Microsoft Build Tools 2015
- Visual C++ Redistributable for Visual Studio 2015
- Visual Studio 2015 SDK
- Visual Studio Tools

Het handmatig installeren van de ontbrekende pakketten hebben niet geholpen. Doordat het probleem niet kon worden opgelost is er gekozen om de hele software te her installeren.

SFML – missing .dll

De problemen waar wij tegen aan liepen waren projecten in Visual Studio's die niet uitgevoerd konden worden. Er werden meldingen weergegeven van het ontbreken van bepaalde dll bestanden. De oorzaak hiervan is het werken met verschillende versies van SFML. SFML update regelmatig zijn software waardoor de oudere versie niet goed samenwerkt met de nieuwere versie van SFML.

- openAL32.dll

Oplossen van dit probleem is voor alle computers het meest recente versie te gaan gebruiken van SFML. SFML is te downloaden op hun eigen website.

Visual Studio 2015 – unable to start program

Dit melding komt voor wanneer de debug sessie niet goed is afgesloten. De program database file zit hierdoor in een lock.

- "LINK : fatal error LNK1201: error writing to program database 'C:\Users\waila\Documents\GitHub\Themaopdracht-5\Debug\gameproject.pdb'; check for insufficient disk space, invalid path, or insufficient privilege."

Dit probleem kan worden opgelost door het opnieuw uitvoeren van de Visual Studio project waarin er gewerkt wordt.

6 Conclusies en aanbevelingen

In het volgende hoofdstuk worden een paar methoden van het programma toegelicht. Bepaalde fouten die zijn gemaakt zullen naar voren komen.

6.1 Functionele conclusies en aanbevelingen

Prijs unit

De prijs van de unit Scout is niet realistisch. De speler kan hierdoor vaker deze unit kopen. Deze unit kan ook veel lopen waardoor je snel bij een resource kon komen en snel geld mee verdiend.
Door de kosten van de unit scout aan te passen(duurder) zal de gameplay wat stabieler en ook uitdagender worden voor beide spelers.

Menu

Aanpassingen die gedaan kunnen worden in Menu

- How-to play knop
- feedback message in de sound menu
- instellen van volume(omhoog/omlaag)
    - achtergrond muziek
    - sound effect

De menu bevat in principe de belangrijkste opties, maar wat er bijvoorbeeld nog bij kan komen is een knopje "how-to-play". Voor nieuwe spelers die het spel niet kennen is het wel handig te weten hoe het spel precies werkt, welke knoppen wat doen en wat precies van je wordt verwacht als speler als je het spel wilt winnen.

Momenteel kan bij de sound menu alleen gekozen worden om geluid dempen. Wanneer op de sound button geklikt wordt zal het geluid gedempt worden. Er wordt echter geen feedback teruggeven. Hierdoor is het vaak onduidelijk om te zien wat de huidige toestand is. Het toevoegen van de huidige toestand als tekst die rechts boven het scherm geprint wordt, zou de onduidelijkheid kunnen verhelpen.

De geluidsvolume kan momenteel alleen in de code worden aangepast. Tijdens het spelen van het spel zijn we erachter gekomen dat het volume te laag was waardoor je het geluid niet te horen was. De optie toevoegen zodat je in menu ook de volume kan aanpassen zou dit probleem op kunnen lossen.

Aanvalsrange

Op het moment moet de aanvalsrange gelijk of groter zijn dan de walklimit van de units. Als dit niet het geval is kan er, als een unit nog kan lopen, binnen het loopgebied aangevallen worden ondanks dat de attackrange zelf dit niet zou halen. Dus als een unit een walklimit van drie heeft zou deze, om ervoor te zorgen er geen aanvallen worden uitgevoerd die niet zouden mogen plaatsvinden, altijd een attackrange van minimaal drie hebben.

Loopberekening

Momenteel wordt er alleen gekeken of er op een vakje zelf niet gelopen mag worden en zal vervolgens vanuit deze positie wel verder kijken of erachter gelopen kan worden. Dit zorgt ervoor dat er geen directe blokkade gemaakt kan worden als de vakjes achter units of gebouwen wel leeg zijn. Ter verbetering van de gameplay zou aangepast kunnen worden dat alleen bepaalde units over objecten heen kunnen en andere units niet. Deze units zullen gewoon om het obstakel heen moeten lopen. Deze verandering zal ook doorgevoerd moeten worden naar het berekenen van hoeveel stappen de unit heeft gezet aangezien dit momenteel, net als de markering van het loopgebied, hemelsbreed gebeurt.

Sound

Het toevoegen van sound zal ook een grote toevoeging zijn aan de spelbeleving. De klasse voor het toevoegen van soundeffecten is wel aanwezig alleen de implementatie van het geluid is nog niet doorgevoerd. Daarom is het sterk aan te raden goede geluidseffecten te zoeken die bij de acties in het spel passen, o.a. schieten, lopen en explosies.

6.2 Technische conclusies en aanbevelingen

Bestand grootte

Van onze bestanden is uiteindelijk de Game.cpp het grootst geworden in omvang en verantwoordelijkheden. Er is tijdens het begin van het project te weinig aandacht aan besteed waar de grenzen hiervan lagen wat tot gevolg had dat nieuwe features standaard in de game terecht kwamen. Uiteindelijk vereiste dit, door de vele informatie die de game bevat, dat nieuwe methodes ook hierin kwamen, indien geen omslachtige doorgeefmethodes gebruikt zouden willen worden.
Onze aanbeveling op dit gebied is dan ook om van tevoren specifieker de verantwoordelijkheden van alle klassen vast te stellen en indien de noodzaak verschijnt voor extra functionaliteit, eerst in het klassendiagram wordt nagekeken of deze niet anders in een eventuele nieuwe klasse kan worden ondergebracht.

Game states

Voor verdere ontwikkeling is onze mening dat het gebruik van verschillende game states het verloop van de code overzichtelijker zal maken. De toestand van de game wordt momenteel bijgehouden door booleans die op meerdere plekken in de code gecheckt worden of een bepaalde toestand actief is de acties specifiek daarvoor uitvoert. Het gebruik van gamestates zou het checken op toestanden tot maar één plek beperken en daardoor ook bij toevoeging van nieuwe toestanden alleen daar te hoeven wijzigen.

Microsoft Visio Studio's – versies

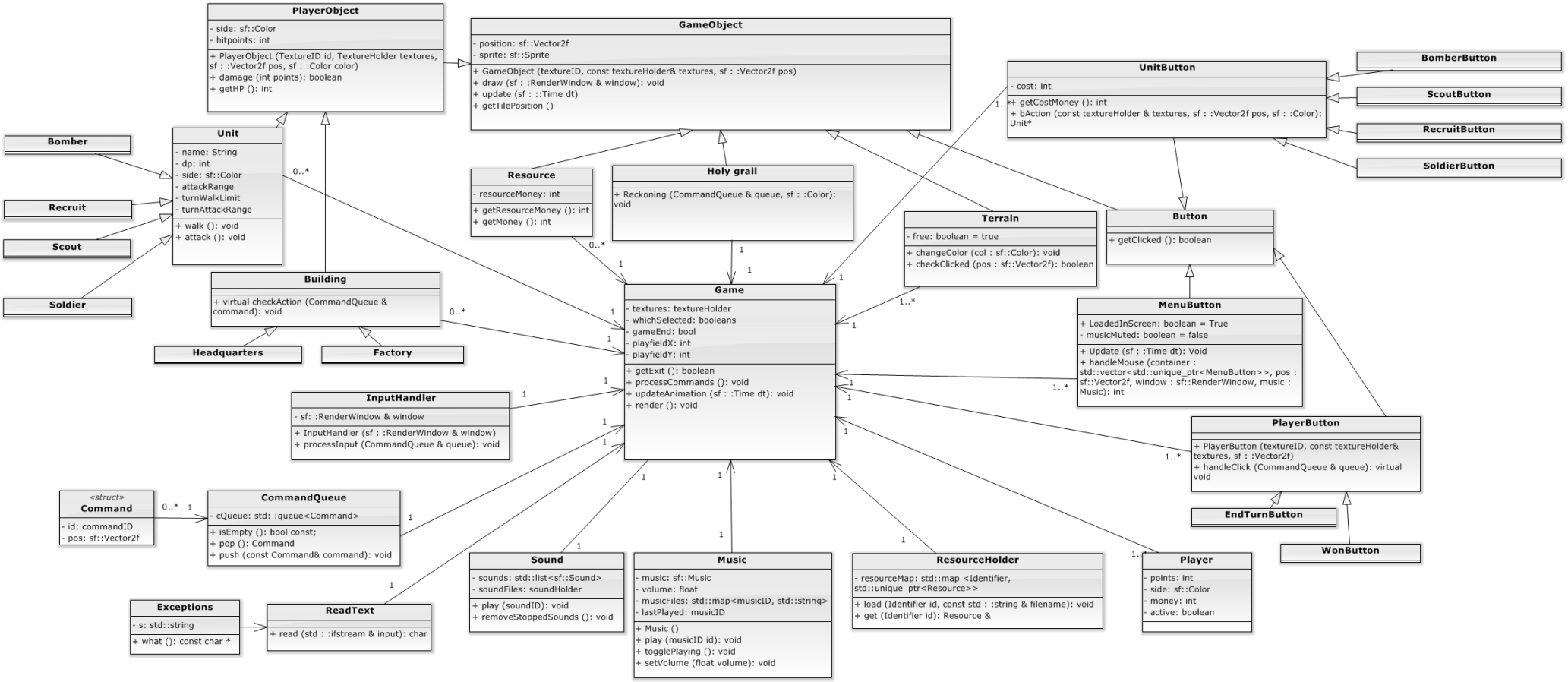
Het gebruiken van verschillende versies van Microsoft Visio Studio kan mogelijk problemen veroorzaken. Projecten die gemaakt zijn in Visual Studio 2015 werken niet goed of zelfs helemaal niet wanneer deze in Visual Studio 2013 worden geopend. Bovendien zijn er ook meerdere versies van de SFML library in omloop en verschil hierin kan ook voor aardige problemen zorgen. Zo is tijdens dit project onnodig veel tijd uitgegaan naar het werkend krijgen van de game op alle systemen van de project leden. Door een volgende keer strakkere grenzen te trekken en betere afspraken te maken, kunnen deze problemen worden voorkomen.

7 Evaluatie

Tijdens dit project is onder de leden veel geleerd op verschillende vlakken. Niet alleen rond C++ en het maken van een game, maar ook zeker rond agile ontwikkelmethoden en het nauw samenwerken met elkaar. Er zijn ten opzichte van het eerste idee een aantal veranderingen doorgevoerd in het eindproduct. Juist door het gebruik van de agile ontwikkel methode was veel meer feedback in kleine stappen mogelijk en kon snel gezien worden of bepaalde zaken wel of niet haalbaar dan wel handig zouden zijn. Een grote les die is geleerd, is om wel vanaf het begin zoveel mogelijk code in verschillende klassen onder te brengen. Hier liepen we in de laatste paar dagen nog tegenaan. Al met al was het een mooi project en er wordt dan ook met tevredenheid op teruggekeken.

Appendices

A.1 Klassendiagram



A.2 Bronvermelding

Chacon, S. (2009). *Pro git*: Apress.

Gomila, L. (2015). SFML (Simple and Fast Multimedia Library). from http://www.sfml-dev.org

Google. (2015). Google Drive. from https://drive.google.com/

Haller, J., & Hansson, H. V. (2013). *SFML Game Development*: Packt Publishing Ltd.

Kamphuis, A. (Producer). (2015). GIT en Dependencies. [Powerpoint] Retrieved from https://cursussen.sharepoint.hu.nl/fnt/46/TCTI-V2THO5-12/Studiemateriaal/GIT%20en%20Dependencies.pdf

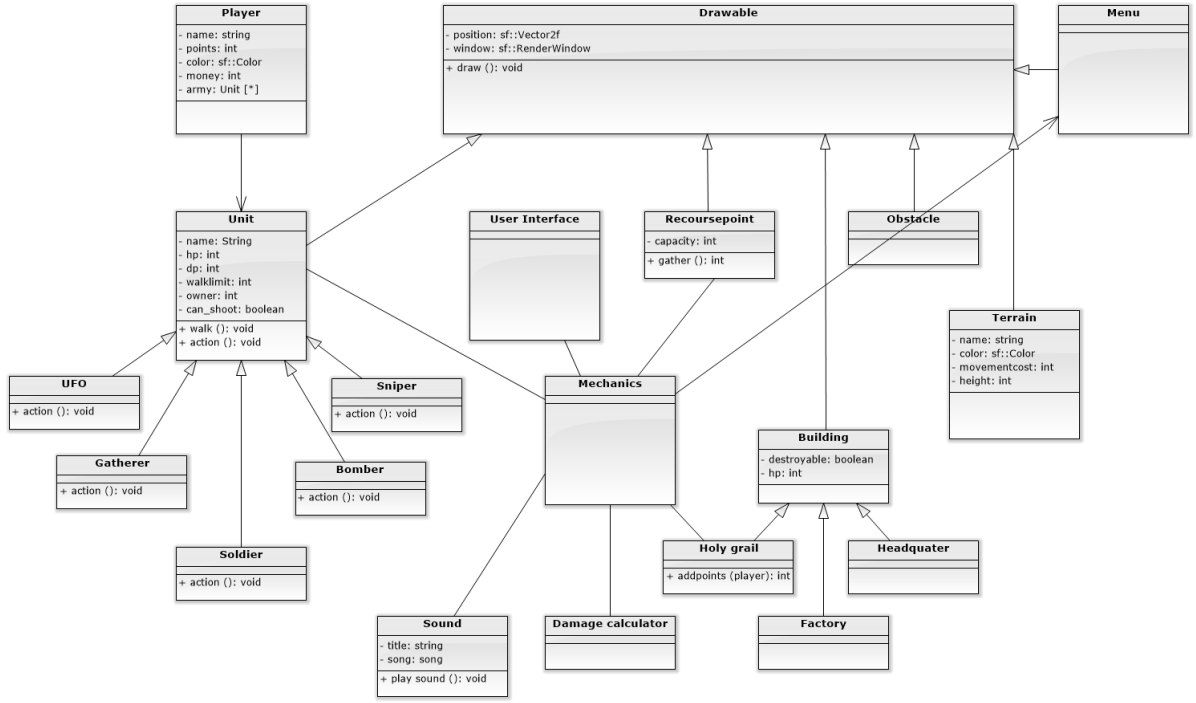
Microsoft. (2015). from https://msdn.microsoft.com/en-us/library/yeky0a1w.aspx

Nijkamp, R., Wagensveld, J. P., van den Berg, Z., & Woe, W. (2015). Big Little Wargame - het game design document. 15.

van Heesch, D. (2008). Doxygen: Source code documentation generator tool. *URL: http://www. doxygen. org*.

A.3 Technisch ontwerp

Klassendiagram



Objecten:
De klasse Mechanics vormt als het ware het hart van het spel. Hier ligt een groot deel van verantwoordelijkheden die bij het spel komen kijken.

Er zijn een aantal klassen die objecten in het spel voorstellen. De categorieën hiervan zijn:

- unit
- building
- resource point
- obstacle
- terrain

Deze klassen zijn allemaal subklasses van een drawable superklasse. Ook kunnen deze objecten zelf weer superklasse zijn, zo heeft een unit onder zich nog de verschillende soorten units (ufo, soldier, gatherer, sniper, bomber).

Omdat het spel met 2 spelers gespeeld wordt, is ervoor gekozen een Player klasse te maken die zorgt dat er meerdere Player objecten aangemaakt kunnen worden die de informatie van iedere speler onthoudt.

Erg belangrijk verder is nog de user interface. Deze zorgt voor de afhandeling van de interactie van de speler.

Fileformaten

- Menu file (bevat menu)
- Sound file (bevat alle geluiden van het spel)
- HUD file (eventueel onder menu / player)
- Player file (bevat player info)
- Terrain file (bevat speelveld en de verschillende tiles)
- Units file (bevat alle units)
- Buildings file (bevat alle buildings)

- Mechanics(controller) file
- Exceptions file (bevalt alle exceptions)

Libraries

- SFML/Graphics
- fstream
- iostream

A.4 Scrumverslagen

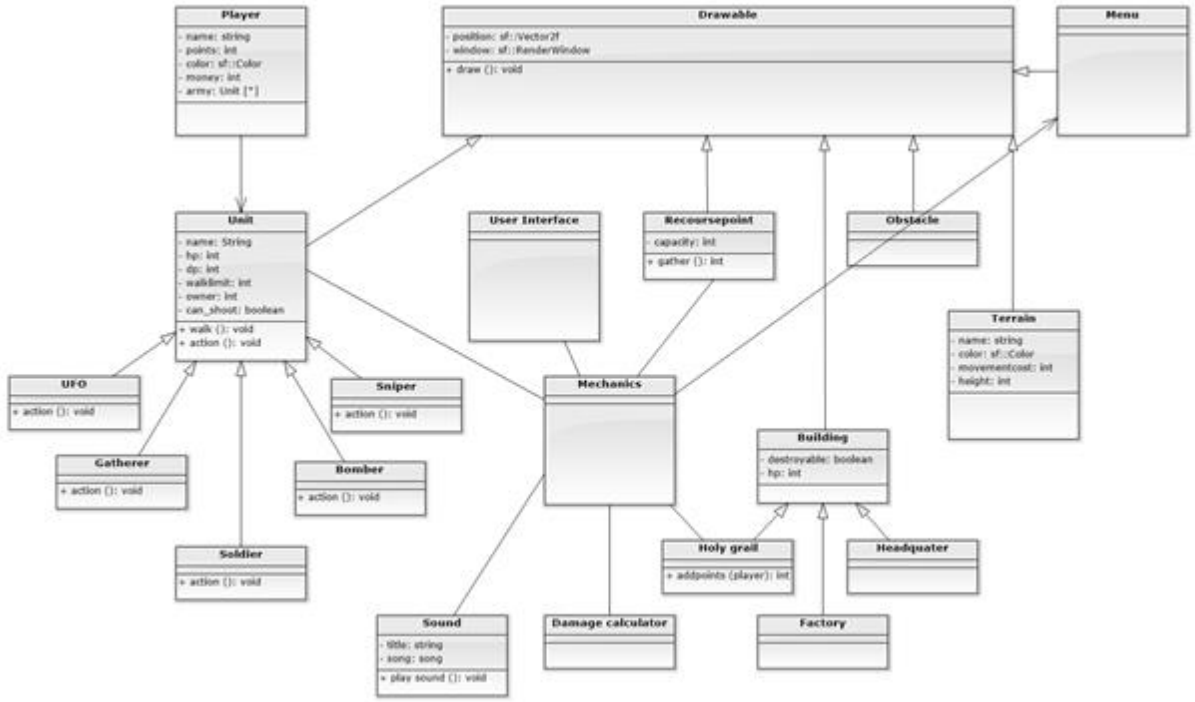
Verslag eerst sprint

Formulering eerste sprint:

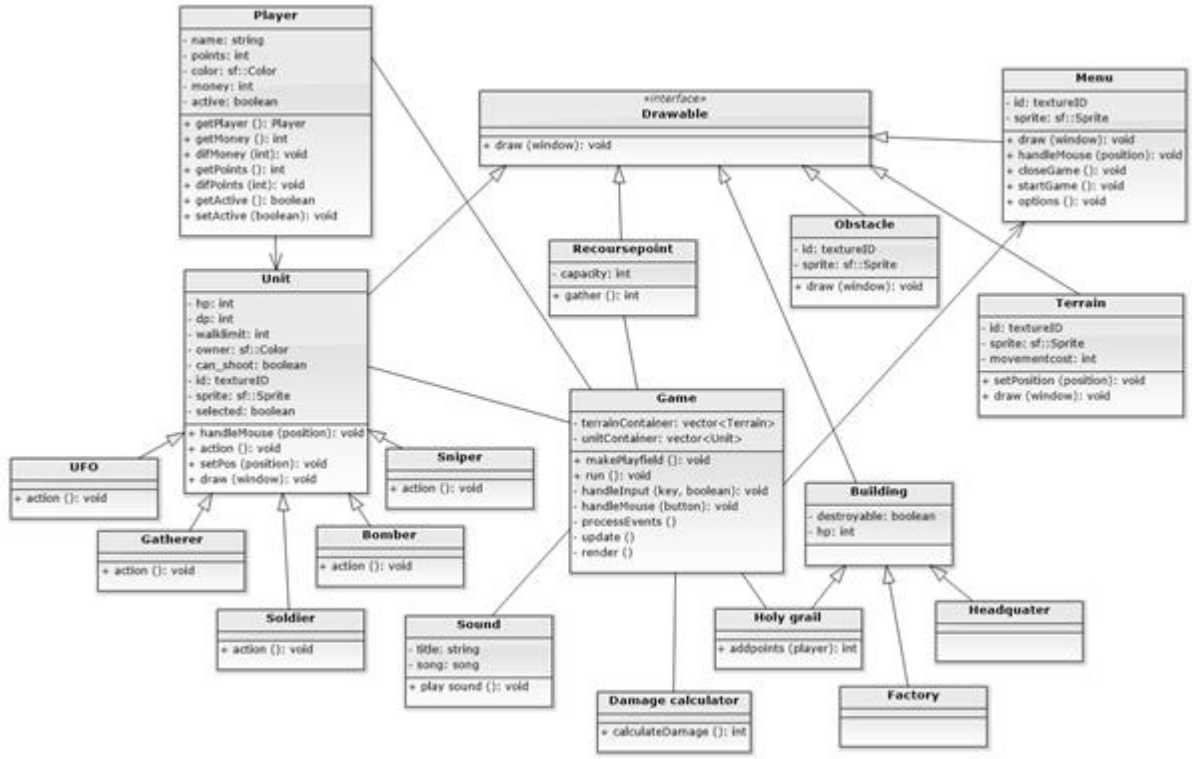
- klassendiagram vertalen naar echte klassen
- speelveld
- vakjes
- units op speelveld
- turn order geïmplementeerd
- verplaatsen van units
- (de verschillende units implementeren)

Kijkend naar de van tevoren opgestelde formulering van de eerste sprint kan worden gesteld dat aardig aan de sprint is voldaan.
Hieronder wordt per punt een korte omschrijving gegeven van wat in de afgelopen sprint is gedaan:

Klassendiagram vertalen naar echte klassen:
Voor de globale opzet van het spel is gebruik gemaakt van het klassendiagram dat eerder is opgesteld. Er bleek al snel het een en ander te missen in de eerste versie van het klassendiagram. Ook bleken bepaalde dingen niet helemaal handig te zijn ingedeeld. Daarom is eerst gewerkt aan een verbeterde versie van het klassendiagram. Hieronder dan ook beide diagrammen.



oude klassendiagram



nieuwe klassendiagram

Zoals te zien is de game klasse (voorheen mechanics) nog altijd de klasse met als het ware de meeste verantwoordelijkheden(het kloppend hart van het spel). Het is een thread die continu zorgt dat de input van de speler wordt afgevangen en alles netjes tekent op de plekken waar het moet.

Aan de hand van dit klassendiagram is de globale opzet voor het spel gemaakt. Er missen natuurlijk nog wel klassen in het spel, aangezien dit slechts een eerste sprint betreft. Zo missen we de sound, building (en dus ook geen subklassen hiervan), damage calculator, obstacle en resource point klassen. Ook is nog slechts een enkele unit klasse gemaakt en niet al meteen met bijbehorende subklassen. De subklassen zullen bij de volgende sprint wel worden meegenomen.

Speelveld:
Het speelveld is een veld dat bestaat uit aan aantal vakjes waarover de units zich kunnen verplaatsen. Deze kunnen zich dus niet vrij verplaatsen. Voor het maken van het speelveld is gekozen om met twee containers te werken: een container met posities en een container met terrain objecten. De eerste container kan voor meerdere doeleinden worden ingezet. Zo kunnen de units hier ook gebruik van maken. Deze container is een array met daarin arrays van Vector2f posities. Zo kan gemakkelijk het speelveld worden aangepast in grootte en heeft elk vakje als het ware zijn eigen positie. De container van de terrain objecten verbindt zijn objecten dus eigenlijk met deze posities, Op deze manier is het dus makkelijker om een speelveld te kunnen maken met verschillende landschappen. Het idee verdient misschien niet de schoonheidsprijs, maar na testen (ook in combinatie met units) blijkt het wel gewoon te werken zoals gehoopt was. Dus kan wel worden gesteld dat dit punt van de sprint gehaald is.

Turn order geïmplementeerd:
Voor dit punt was het nodig om de Player klasse werkend te krijgen. De sleutel om de turn order te kunnen verwerken is de Player zelf bij te laten houden of hij actief is door middel van een boolean. Vervolgens worden de beurten vanuit de Game klasse geregeld. Voor het verplaatsen van een unit moet dan ook eerst worden gekeken welke speler aan de beurt is, zodat niet zomaar een unit van het andere team kan worden verplaatst.

Player heeft een eigen kleur en dit is tevens zijn identiteit. De spelers zijn of blauw of rood. De units weten tot welke Player ze behoren en voor het tekenen van de units wordt dan ook een filter gebruikt die ze rood dan wel blauw afbeeldt op het scherm. Zo is makkelijk te zien bij welke speler de units horen.
Al met al kan worden gesteld dat aan dit punt is voldaan.

Verplaatsen van units:
Voor het verplaatsen van een unit is gekozen voor de volgende mogelijkheid: eerst moet op de unit worden geklikt met de linkermuisknop. Vervolgens is de unit geselecteerd (dit is een toestand die de unit zelf bij zal houden) en verandert hierbij ook van kleur om de speler ook te laten weten dat hij is geselecteerd. Nu kan de speler op een omringend veld klikken om de unit (één hokje) te verplaatsen. Hierna zal de unit weer gedeselecteerd worden. Als de speler op een vakje klikt dat buiten het loopbereik van de unit ligt zal de unit zich niet verplaatsen en dan ook zichzelf gaan deselecteren.
Om dit te realiseren wordt binnen de Game klasse bij het updaten gecheckt of er een muis event is opgetreden. Als dit zo is zal de unitcontainer worden afgelopen en per unit worden gekeken of deze geselecteerd moet worden. De afhandeling van het lopen wordt verder binnen de unit afgehandeld.
Na testen bleek dit prima te werken dus ook voor dit punt kan gesteld worden dat het is behaald.

(De verschillende units implementeren):
Dit laatste punt staat niet voor niks tussen haken. Al voor de sprint begon was niet zeker of dit wel of niet in de sprint moest worden opgenomen. Uiteindelijk is alleen een algemene Unit klasse gemaakt tijdens de eerste sprint en is dus niet voldaan aan dit punt. Gelukkig was dit niet noodzakelijk dus er is geen man overboord. Voortaan moet gewoon beter gekeken worden naar wat wel of niet in de sprint opgenomen moet worden. Eventuele zaken kunnen beter worden weggelaten en meegenomen naar een latere sprint.

Conclusie:
Aan alle punten die waren opgesteld voor deze sprint is voldaan dus ook de sprint is gehaald. Hieruit kan dan ook worden opgemaakt dat een aardige inschatting is gemaakt van de dingen die binnen de afgesproken tijd behaald zouden kunnen worden. Zeker omdat er verder geen tijd over was om vast verder te werken aan nieuwe zaken. Een leerpunt voor de volgende sprint is het strikter formuleren van de sprint. Bijzaken moeten hier niet in worden opgenomen.

Verslag tweede sprint

Formulering tweede sprint:

- walk limit instellen, met preview (oplichting vakjes)
- andere units maken (+ alle attributen)
- HUD
- Menu
- Sound

Terugkijkend op de tweede sprint zijn we aardig tevreden. Bijna alles is afgerond, ondanks de tegenslagen die we hebben gehad. Joost bleek namelijk problemen te hebben met zijn Visual Studio. Aan het oplossen hiervan is hij veel tijd verloren (en het probleem is nog niet helemaal verholpen).

Hieronder wordt per punt even een korte omschrijving gegeven van wat er de afgelopen sprint is gedaan:

Walk limit instellen, met preview (oplichting vakjes):
Het eerste punt is meteen een lastig punt. Hiervoor even een stukje achtergrond informatie. Voor het spel is besloten een game klasse te maken die als het ware het kloppend hart van het spel is. Buiten dat vanuit de main de run methode van de game wordt aangeroepen, wat eigenlijk zorgt voor het runnen van het hele spel (updaten, renderen), is de game klasse hier ook een centrale klasse die zelf alle andere klassen kent. Om bijvoorbeeld het speelveld te maken heeft de game klasse een container met daarin terrain objecten. Evenzo zijn er twee containers met unit objecten (voor elk team 1). We hebben ervoor gekozen het bepalen of de unit mag lopen en het markeren van deze vakjes te scheiden. De unit klasse is verantwoordelijk voor het bepalen of hij mag lopen, terwijl de game aan de terrain tiles doorgeeft welke vakjes op moeten lichten. Waarschijnlijk is dit niet de beste en meest efficiënte oplossing, maar we beschikken op het moment nog niet over voldoende kennis om dit anders aan te pakken en voor nu werkt het prima. Mochten we later met een betere oplossing komen dan kan dit alsnog worden aangepast.

Andere units maken (+ alle attributen):
Dit was dan weer een wat sneller te halen punt. Van de unit klasse is een superklasse gemaakt en er zijn subklassen toegevoegd om hiervan te erven. Deze subklassen zijn de verschillende soorten units, namelijk de soldier, bomber, sniper, UFO, gatherer. Deze klassen hebben nu ook allemaal hun eigen stats gekregen, zoals de walklimit en het healthpoint. Een methode die deze subklassen overriden van hun superklasse is de action methode, deze is namelijk per unit verschillend en zat tijdens de volgende sprint onder handen worden genomen voor de aanvallen van de units.

HUD:
Voor dit punt was het nodig ook met fonts te gaan werken. Dit was even werk om uit te zoeken hoe dit moest, maar het is uiteindelijk gelukt. Op dit moment wordt een tekst rechts boven in de hoek van het spel afgebeeld. Dit moet straks de algemene informatie van de actuele speler voorstellen waarin hij zijn aantal units kan zien, het aantal punten en zijn geld. Verder is het bedoeling straks bij het aanklikken van een unit de informatie van die geselecteerde unit af te beelden.

Menu:
Ook is gewerkt aan een menu. Als het spel opstart moet hij eerst in een begin menu komen, waarna keuze gemaakt kan worden om het echte spel te starten, naar het optie menu te gaan, of het spel weer af te sluiten. Op dit moment is het optie menu nog niet gemaakt, maar wel het begin menu. Ook is er een pauze menu gemaakt. Zodra de window waarin het spel zich afspeelt wordt verlaten, zal het spel zichzelf automatisch pauzeren en in het pauze menu terechtkomen.

Sound:
In het begin van het project is een resourceholder gemaakt om alle resources in gestructureerd op te kunnen vragen. Omdat dit al gemaakt is in de vorige sprint voor de Sprites was het vrij eenvoudig om hieraan ook sounds toe te voegen. Voor het afspelen van de sounds worden de sounds opgeslagen in een lijst. De methode "remove stopped sounds" kijkt of een geluid nog bezig is met spelen. Als dit niet het geval

is wordt deze verwijderd uit de lijst. Het is zowel mogelijk om korte sounds af te spelen, zoals shot sounds, als ook hele liedjes voor in de background.

Conclusie:
We kunnen aardig positief terugkijken op deze sprint. Ondanks veel tijdverlies door technische problemen en natuurlijk het uitzoeken van hoe dingen het beste aangepakt konden worden op codeergebied is toch naar behoren aan de punten voldaan. Een les die deze week naar voren is gekomen is toch wel de communicatie tussen de teamleden onderling. Bij problemen duurde het vaak erg lang voor er ook echt hulp kwam van de teamleden of voordat de andere teamleden snapten wat het probleem was. Voor de volgende sprint gaan we dan ook strakker en nauwer contact houden.

Verslag derde sprint

Formulering derde sprint:

- Units aanvallen
- Animatie van Sprites
- Sprites
- Buildings maken

Tijdens de derde sprint heeft het spel al wat meer vorm gekregen en werd het langzaamaan een speelbare game. Natuurlijk nog niet echt speelbaar, maar inmiddels kunnen de units elkaar aanvallen (zei het nog niet optimaal) en dat zorgt dus al wel voor echte interactie tussen de spelers.

Deze sprint hebben we speciaal iets langer laten duren dan anders, aangezien er tussendoor ook tentamens op de planning stonden. Tijdens de toets week is het project dan ook in overleg op een laag pitje komen te staan.

Helemaal aan het einde van deze sprint is eindelijk het probleem van Joost verholpen. We hebben met z'n allen flink wat achterstand opgelopen door technische problemen dus het is fijn dat het nu eindelijk is opgelost.

Hieronder wordt per punt even een korte omschrijving gegeven van wat er de afgelopen sprint is gedaan:

Units aanvallen:
Het eerste punt dat op de agenda stond was het grootste punt. Hier is de meeste tijd in gaan zitten en is nog niet optimaal. Het optimaliseren zullen we echter meenemen naar de volgende sprint. De units konden al lopen en kunnen nu ook aanvallen met een eigen attack range. Dit houdt in dat sommige units alleen van dichtbij aan kunnen vallen, terwijl anderen dit juist al op een afstandje kunnen doen. Een belangrijk punt dat sowieso in een volgende sprint nog moet worden meegenomen is het feit dat een unit slechts één keer per beurt aan kan vallen. Nu kan je onbeperkt aan blijven vallen tijdens je speelbeurt en dat is natuurlijk niet de bedoeling. Ondanks deze optimalisatie puntjes is dit punt ruim voldoende gehaald en daar zijn we dan ook positief over.

Animatie van Sprites:
Voor dit punt was het bedoeling dat de units een animatie zouden krijgen in plaats van enkel een statisch plaatje. Dit hebben we gerealiseerd door met spritesheets te werken. In de code van de units kan worden aangegeven uit hoeveel "losse" plaatjes het totaalplaatje bestaat en dan wordt met een berekening de animatie gemaakt. Zo ziet meteen het spel er minder statisch uit.

Sprites:
Het verschil van dit punt met het vorige punt is dat hier plaatjes zijn gezocht die we ook echt in ons spel willen gebruiken en niet speciaal animaties. In de vorige sprints werkten we namelijk alleen nog maar met gekleurde vierkantjes, puur omdat we alleen hoefden te zien of de code zou werken. Dat het er ook een beetje als spel uit moet zien was een zorg voor later. Nog niet alle plaatjes zijn inmiddels aangepast, maar het ziet er al wel een stuk beter uit dan enkel de gekleurde vierkantjes die we eerst in het spel hadden zitten.

Buildings maken:
Verder is nog een begin gemaakt met de buildings die in het spel moeten komen. Inmiddels hebben we een factory die op het veld kan worden geplaatst. Er is een algemene superklasse gemaakt waarvan de verschillende buildings straks kunnen erven.
De buildings hebben zelf een health, wat betekent dat ze straks ook vernietigd kunnen worden. Zo is het de bedoeling dat straks ook een headquarter wordt gemaakt. Als deze wordt vernietigd door de vijand dan heb je verloren.

Conclusie:
Dit was een vrij heftige sprint, omdat we nog met veel problemen zaten en natuurlijk met tentamens tussendoor. Ondanks dat zijn alle punten van de sprint op tijd afgerond en daar zijn we dan ook positief

over. Ook met Git zijn er vaker problemen geweest, bijvoorbeeld met het mergen van verschillende branches. Om dit de volgende keer te voorkomen willen we vaker een push richting de Git gooien zodat per stap niet een te grote verandering is opgetreden en het samenvoegen van branches en dergelijke makkelijker zullen verlopen.

Verslag vierde sprint

Formulering vierde sprint:

- HUD van unit (health / damage)
- Buildings kunnen units spawnen
- Feedback van Wouter verwerken (verkorten code / leesbaarder / duidelijker)
- Input afhandeling loskoppelen van Game logic

Tijdens de vierde sprint is voornamelijk aandacht uitgegaan naar het overzichtelijker en leesbaarder maken van de code. Tevens is de HUD onder handen genomen zodat ook de health en damage van de units zichtbaar zijn. Ook is verder gewerkt aan de buildings, deze kunnen inmiddels units laten spawnen voor de actieve speler.

Nu het probleem van Joost is verholpen kon ook hij meer bij gaan dragen aan de code, hij heeft dan ook meteen de HUD voor zijn rekening genomen.

Hieronder wordt per punt even een korte omschrijving gegeven van wat er de afgelopen sprint is gedaan:

HUD van unit (health / damage):
Het eerste punt dat we onder handen hebben genomen is meer informatie in het HUD, namelijk belangrijke informatie over de units. Het is nu mogelijk om een unit te selecteren, waarna zichtbaar wordt om wat voor soort unit het gaat, wat zijn levens zijn en hoeveel schade hij kan aanrichten bij een aanval. Dit maakt het spel een stuk overzichtelijker en speelbaarder.

Units aanvallen:
Het eerste punt dat op de agenda stond was het grootste punt. Hier is de meeste tijd in gaan zitten en is nog niet optimaal. Het optimaliseren zullen we echter meenemen naar de volgende sprint. De units konden al lopen en kunnen nu ook aanvallen met een eigen attackrange. Dit houdt in dat sommige units alleen van dichtbij aan kunnen vallen, terwijl anderen dit juist al op een afstandje kunnen doen. Een belangrijk punt dat sowieso in een volgende sprint nog moet worden meegenomen is het feit dat een unit slechts één keer per beurt aan kan vallen. Nu kan je onbeperkt aan blijven vallen tijdens je speelbeurt en dat is natuurlijk niet de bedoeling. Ondanks deze optimalisatie puntjes is dit punt ruim voldoende gehaald en daar zijn we dan ook positief over.

Buildings kunnen units spawnen:
Dit punt van een belangrijke stap voor het spel. De spelers kunnen nu units laten spawnen via een factory, waardoor ze dus nieuwe troepen kunnen kopen met het geld dat ze hebben verdiend door resources “leeg” te halen. Als een speler aan de beurt is en hij klikt op zijn factory, dan opent aan de zijkant van het scherm een soort menuutje met een aantal knoppen van de verschillende units die verkrijgbaar zijn. Elke unit kost een bepaalde hoeveelheid geld om te verkrijgen. Als de speler genoeg geld heeft kan hij door op de knop te drukken een unit laten spawnen voor zijn team. Hierdoor heeft het spel ook meteen een extra doel, namelijk het “leeghalen” van resources om genoeg geld te verzamelen zodat een sterk leger gevormd kan worden om te vijand te kunnen verslaan. Het is gelukt om dit in het spel te verwerken. Een puntje waar we achteraf nog aan dachten is het feit dat nu in het menu niet zichtbaar is hoe duur de units zijn. Dit weten wij wel, maar de spelers straks natuurlijk niet dus dit moeten we in de volgende sprint nog even aan gaan passen.

Feedback van Wouter verwerken (verkorten code / leesbaarder / duidelijker):
We hebben ons best gedaan naar aanleiding van Wouter van Ooijen om de code wat leesbaarder te maken en tevens op plekken waar grote lappen code voorkwamen kijken of de code hier niet veel overlap bevatte en we dit dus korter zouden kunnen noteren. Natuurlijk is het erg persoons afhankelijk of de boel er ook echt duidelijker van is geworden, maar we vinden in elk geval dat het al een aardig stuk scheelt. Zo is bijvoorbeeld de check of een unit zich mag verplaatsen onder handen genomen, deze is niet alleen duidelijker geformuleerd maar is ook bijna de 2/3^{e} aan omvang afgenomen.

Input afhandeling loskoppelen van Game logic:
Ook in het kader van het vorige punt hebben we de input afhandeling losgehaald van de Game logic en dit in een aparte klasse gezet genaamd InputHandler. Dit maakt de boel niet alleen leesbaarder zoals in het

vorige punt vooral voorop stond, maar is ook een betere manier van coderen. Het is netter om de afhandeling en de verwerking te scheiden van elkaar. Dit hebben we dan ook gedaan.

Conclusie:
Deze sprint was een sprint waar eindelijk weer een beetje schot in de zaak begon te komen. We liepen zeker tegen moeilijkheden aan, maar dan vooral op het gebied van bugs die ontstonden in het spel zelf, en zodra deze dan werden opgelost ontstonden er meestal weer nieuwe bugs. Maar gelukkig zaten de moeilijkheden dit keer niet meer in de vele errors van Visual Studio zelf en van moeilijkheden met Git. We beginnen nu aardig op gang te komen en hebben er vertrouwen in de volgende sprint op een hoger tempo te kunnen werken.

Verslag vijfde sprint

Formulering vijfde sprint:

- Headquarter toegevoegd + winconditie
- Holy grail + winconditie
- Level geladen uit txt bestand
- 2 nieuwe Units
- Textures mooier maken en bugfix

Tijdens de vijfde sprint hebben we aandacht besteed aan de wincondities binnen het spel. Zo kan eindelijk het spel "echt" worden gespeeld en kan dus ook worden gewonnen of verloren. Verder is ook het spel uitdagender en leuker gemaakt door 2 nieuwe units toe te voegen aan de game.

Echter is de meeste tijd wel gaan zitten in de vele kleine bugs die in het spel zaten. Soms waren ze wat makkelijker op te lossen en dan waren de fouten meestal ook groter en sneller ontdekt. Andere bugs kwamen pas aan het licht toen we speelsessies hebben gehouden. Bijvoorbeeld als je als speler, terwijl je een van je eigen units had geselecteerd, probeerde de beurt af te geven aan de andere speler, ging het spel vaak onvoorspelbaar gedrag vertonen.

Hieronder wordt per punt even een korte omschrijving gegeven van wat er de afgelopen sprint is gedaan:

Holy grail + winconditie:
Het eerste punt was meteen een belangrijke stap voor het spel. Door het toevoegen van de Holy Grail werd namelijk ook één van de belangrijkste mogelijkheid om te winnen ingevoerd, namelijk door dit centrale punt te bezetten en daarmee schade aan de tegenstander aan te richten. Zodra deze geen levens meer heeft heb je als speler gewonnen. Na het maken van de Holy Grail bleek het spel ook eindelijk leuk te worden om te spelen. Je moet namelijk zorgen dat je de vijand weg houdt bij dit middelpunt, zelf dit punt onder controle zien te krijgen en tegelijkertijd ook je eigen basis verdedigen. Het is aardig gelukt dit in het spel te verwerken.

Headquarter toegevoegd + winconditie:
Net als het eerste punt was het toevoegen van headquarters, de belangrijkste buildings die de spelers in hun bezit hebben, een toevoeging aan de spelervaring. Dit komt doordat ook hier een winconditie aan gebonden is, namelijk zodra je headquarter is gesloopt heb je verloren.

Level geladen uit txt bestand:
Tot voor deze sprint moesten de units, buildings en resources allemaal handmatig worden toegevoegd in het spel. Dit was natuurlijk niet praktisch dus hebben we ervoor gekozen aan het begin van het spel een txt bestand in te laten lezen die omschrijft waar al deze objecten geplaatst moeten worden. Dit is niet alleen overzichtelijk geworden, maar is voor de toekomst ook makkelijk aanpasbaar. We liepen wel even tegen een flinke muur op, want het bleek dat ons veld dat toen nog 10 bij 8 hokjes breed was niet zomaar even aangepast kon worden naar een groter veld. Dit bracht veel problemen met zich mee. Maar na dit verholpen te hebben is het vrij eenvoudig om zowel de grootte van het speelveld, als een level zelf aan te passen. Een goede toevoeging aan ons spel dus.

2 nieuwe Units:
Tijdens deze sprint hebben we naast de 2 units die we al hadden nog 2 units toegevoegd om het spel wat meer uitdaging en diepgang te geven. In principe hadden we het meeste hiervoor al gereed, dus het bleek niet een hele lastige klus te zijn. Dit is dan ook zeker gehaald.

Textures mooier maken en bugfix:
Voor dit laatste punt stonden de textures en de bugfixes op de planning. De textures hadden natuurlijk de nodige tijd en aandacht nodig, maar dit was verder niet moeilijk. De bugfixes daarentegen zijn en ander verhaal. Hier is dan ook de meeste tijd in gaan zitten tijdens deze sprint. Vaak waren ook net kleine foutjes opgelost, en dan doken er alweer nieuwe op. Natuurlijk zullen er nog wel wat dingen fout gaan, maar over het algemeen presteert het spel aardig naar behoren dus hier zijn we dan ook erg tevreden over.

Conclusie:
Na afloop van deze sprint kunnen we wel stellen dat zo ongeveer alle elementen in het spel zitten die erin zouden moeten zitten. Natuurlijk hebben we hier en daar wat ideeën aan moeten passen, omdat het of te moeilijk was, of omdat het qua tijd niet haalbaar bleek te zijn. Maar ondanks dat is er toch een echt speelbare game opgeleverd dit keer en daar zijn we positief over. Vanaf nu kan de finetuning van start gaan en gaan we ons meer dan nu richten op het technisch verslag.