**0. Premise**  
  
Zoals elke 3de-jaars student heb ik de opdracht gekregen om een Bachelor-proef te maken die een nauw verband heeft met de richting Multi Media Productie, afstudeerrichting New Media Development.

Ik heb hierbij gekozen voor de productie van een educatieve videogame.

Deze game is vooral bestemd voor nieuwe GDM-studenten maar heeft nog een tweede doelpubliek, namelijk ook de huidige GDM-studenten.

Het doel van het spel is tweeledig:

* De speler maakt kennis met de huidige campus Artevelde Mariakerke in al zijn locaties en functies, gespecifieerd op het B-blok omdat de eerstejaarsstudenten vooral daar les hebben.
* Tegelijkertijd krijgt de speler ook info over de docenten en hun vakken en leert ze op die manier kennen.

Ik zou kiezen voor het genre van de horror-game met stealth interacties.

Je bent eerstejaarsstudent en je hebt een doel van in het begin.

Je wil er namelijk graag bij horen en je neemt deel aan de doop.

Om te mogen deelnemen echter moet je bewijzen dat je het waard bent, en moet je als het ware toegangstickets verzamelen. Die zijn te vinden op de campus in Mariakerke. Studenten MMP gaan bij voorbeeld voor fototoestellen, studenten CMO voor potloden, studenten NMD voor chips en studenten GMB voor rekenmachines.

Er is natuurlijk ook een verboden aspect aan de hele zaak – anders zou het geen doop zijn - , je doet dit in het holst van de nacht en je mag daar eigenlijk helemaal niet zijn.

Je moet je dus continu verstoppen voor “boze” docenten die op de campus patrouilleren en die je eruit kunnen gooien. Tenzij je correct antwoordt op hun vraagjes in verband met de cursus die ze doceren. Op die manier kan je als speler kennis maken met richtingen en docenten.

Tijdens het spel wordt je tijd bijgehouden en de tijd wordt op een scorebord gepubliceerd. Zo kunnen spelers met elkaar in competitie gaan en in een ranking terecht komen.

Deze bachelor-proef zal vooral een combinatie zijn van 3D modelleren, animaties en het programmeren van game-logica.

**1. Technisch onderzoek.**  
  
Deze bachelor proef zal vooral een technisch onderzoek zijn naar hoe ik mijn game gemaakt heb en welke beslissingen in genomen heb tijdens de productie.  
  
Tijdens de productie en planning van de verschillende features of specificaties in game development botste ik op verschillende problemen die ik zou kunnen tegenkomen. Deze problemen gaan van het laden van modellen tot een patrouille te programmeren die de antagonist zal afleggen.  
  
Ik zal dus beginnen met een chronologische lijst van alle technische onderzoeken en beslissingen die ik genomen heb.

**1.1 Game Engine.**

Zowat de belangrijkste beslissing voor dit project is de game engine. De engine is zowat het hart van je game of simulatie, ze bevatten een grote hoeveelheid aan functionaliteiten die het voor de ontwikkelaar makkelijker maakt om een game te produceren. Het klaarzetten van scenes om je game op te splitsen in verschillende hoofdstukken, het automatisch configureren van een 2D of 3D ruimte en het renderen hiervan, het importeren van een Camera, fysica en materiele simulaties, kortom: een engine is zeer belangrijk.

Er zijn hiervoor enkele mogelijkheden die ik kan gebruiken. Of ik gebruik een commerciële engine. Of ik maak zelf een game zonder engine.  
  
Het 2de voorstel ga ik zelf niet eens proberen. Anders zou ik hier binnen enkele jaren nog steeds zitten. Vandaar dat ik gekozen heb voor het DRY principe en ik een commerciële game engine zal gebruiken.

**1.1.1 Unity**

Voor commerciële engines zijn er enkele opties, de meest bekende is wellicht de Unity Engine. Unity is een all-purpose engine die voor je voor verscheidene projecten kan gebruiken, de meest gebruikte use-case is natuurlijk wel game development, maar Unity kan de volgende functies goed uitvoeren:

* Film, Animatie en Cinematics
* Architecture, Engineering en Construction
* Automotive, Transportation en Manufactoring

Unity kan ook de geschreven code en games compileren naar een grote hoeveelheid beschikbare platformen, dit zorgt ervoor dat je de code slechts eenmaal moet schrijven en deze overal kan gebruiken. Enkele belangrijke beschikbare platformen in de scope van mijn bachelorproef zijn:

* iOS
* Android
* Windows
* WebGL
* Switch
* Linux
* PS4
* XboxOne

Unity maakt gebruik van C#, C# is een object georienteerde programmeertaal die als scripttaal werkt met Unity. De functies die je in de code aanroept zal dan ook machine level code aanspreken zodat de simulatie of game performant blijft.

**1.1.2 Unreal Engine**

De Unreal Engine is de game engine die gebruikt werd om Fortnite te produceren en is bezit van Epic Games. Andere bekende titels zijn o.a. Minecraft Dungeons, Life is Strange, We Happy Few en Ark: Survival Evolved.

UE is dan ook de koploper van 3D games in de laatste jaren vanwege hun zeer performante compilers en builds.

Net zoals Unity heeft Unreal verschillende use-cases, deze zijn o.a.

* Games
* Architecture
* Automotive & Transportation
* Broadcast & Live-events
* Film & Televisie
* Training en simulatie

En met de opkomende hype voor de hyper-performante UEv5 wordt de engine enkel populairder.

Unreal Engine heeft tenslotte 2 mogelijkheden om games te produceren, dit zijnde met het gebruik van C++ of Blueprints.

C++ is een low level programmeertaal met enorm veel controle over de engine en hoe performant de game kan zijn, het enigste nadeel is de hoge leercurve en de complexiteit van het schrijven van C++ code.

Blueprints is dan weer een oplossing voor dit probleem. Blueprints is een visuele script taal waarbij je nodes kan verbinden om zo computer behaviour te simuleren en controleren.

**1.1.3 Godot Engine**

Godot is een nieuwkomer bij de gratis te gebruiken game engines, de engine bestaat officieel sinds 2014 en is compleet gratis, dat betekend dat je i.v.m. Unity en Unreal niets betaald.

De godot engine laat de gebruiker ook toe om in 4 verschillende opties te scripten, het gebruik van GDScript (een python versie van Godot), Nativescript, C# en VisuelScript.

Natuurlijk is het belangrijk om rekening te houden met wat de Engine kan voor de prijs die je betaald, en aangezien deze gratis is zitten er heel wat features niet in.

**1.1.4 Besluit**

Achter alle mogelijkheden op een rij te zetten heb ik uiteindelijk gekozen voor Unity te gebruiken, dit omdat Unity veel mogelijkheden heeft en gratis is binnen bepaalde limitaties.

Unity gebruikt bovendien ook C# als scripttaal, een taal die ik min of meer beheers en ten slotte heeft Unity een grote community waar je makkelijk vragen kan stellen.

**1.2 3D modellen**

Een 3D game heeft natuurlijk ook 3d assets nodig, deze assets bestaan uit polygonen en hoe meer polygonen de Engine moet renderen hoe zwaarder de game.  
  
Voor het maken van deze modellen bestaan er verschillende programma’s. De meest bekende hiervan zijn ZBrush, Maya en Blender.

Aangezien ZBrush en Maya beiden betaalde software zijn heb ik gekozen voor Blender te gebruiken als mijn 3D software.

**1.2.1 Blender**

Blender is een gratis te gebruiken programma dat door amateurs vaak gebruikt wordt voor game-objecten en animaties te maken. Blender heeft hiervoor dus ook verschillende functies die het makkelijker maken om 3d-modellen te maken en aan te passen.

**2 Game Productie**

Wanneer je een game maakt of software schrijft blijft het proces hetzelfde. Je begint allereest met het onderzoeken van de bestaande bibliotheken en frameworks die je kan gebruiken, vervolgens zal je dan overgaan tot het onderzoek binnen een framework zelf.  
  
Ik heb het daarstraks gehad over het onderzoek naar de te gebruiken frameworks, deze heb ik dan eens op een rij gezet en de meest toegankelijke uitgekozen voor mijn project, dit zijnde de Unity Engine voor de game en Blender voor de assets.

Net zoals mijn onderzoek naar het ideale framework zal ik mijn onderzoek en werkwijze opsplitsen in het Unity en Blender gedeelte.

**2.1 Inspiraties**

**2.1.1 Slenderman**

De game die ik zal maken neemt inspiratie van de Slenderman horror game. In de slenderman games is het de bedoeling dat je 8 pagina’s verzameld om te winnen, maar de “Slenderman” achtervolgd je en als je hem te veel of te lang aankijkt verlies je het spel.  
  
Het concept is simpel maar krachtig, de spelers voelen een constante druk van een mythisch wezen die voor hem een prooi is, hierdoor zal de speler zelf voorzichtig te werk moeten gaan en ook strategisch nadenken.

Bij slenderman is het wel belangrijk dat het een soort van geest is die constant verdwijnt en verschijnt, de speler heeft dus geen controle over de acties van de AI.

**2.1.2 Alien Isolation**

Alien Isolation is een first person horror game waarbij de speler het ruimteship van Svestapol binnendringt en merkt dat het iedereen voor zichzelf is. De game onthuld stap voor stap dat er een wezen jacht maakt op de mensen op het ruimteship.

Deze antagonist werkt anders dan Slenderman, deze alien zit constant achter je aan en zet heel veel druk op de speler. Elk klein geluidje leidt tot een game over scherm, de speler kan niet zomaar weglopen en moet rekening houden met waar de alien zit op elk moment.

De A.I. van Alien Isolation is bijzonder complex en bevat een behaviour tree van meer dan 100 verschillende branches die zich dan telkens opnieuw vertakken.

**2.2 Vereisten van de game**

De game die ik wil maken zal een combinatie zijn van Slenderman en Alien Isolation, natuurlijk op een kleinere schaal aangezien de AI van Alien Isolation enorm complex is.

De student of speler zal de opdracht krijgen om voor zijn /haar studentendoop 8 voorwerpen te verzamelen in het schoolgebouw, daarbij zal hij rekening moeten houden dat er docenten, of beter gezegd: de geest van de docenten de gangen zal verkennen en zal zoeken achter hun volgend slachtoffer.  
  
Ik wil dus diezelfde sfeer creëren als in Alien Isolation maar da op een kleinere schaal. De speler zal voorzichtig moeten zijn, je kan ontsnappen, maar de kans is enorm klein.

Tenslotte is het belangrijk om rekening te houden met de verschillende functionaliteiten die in de game verwerkt moeten worden.

De speler:

* Een brochure bij zich hebben met de voorwerpen op die hij moet vinden.
* Voorwerpen kunnen opnemen.
* Kunnen wegrennen van de geesten.
* Zijn/haar zaklamp kunnen aan/uitschakelen.

De geesten

* Een vastgelegde weg om de school te patrouilleren.
* De speler kunnen detecteren vanop een afstand.
* Licht van de zaklamp kunnen detecteren.
* De speler achtervolgen en indien hij de speler kwijt is verder patrouilleren.

De school

* Voorwerpen in de school plaatsen en wanneer je ze verzameld van de lijst schrappen.
* Deuren kunnen open en dichtdoen.

**2.3 Game assets.**

Game development is een van de zwaarste en moeilijkste jobs die je kan doen, ieder deel van een game heeft enorm veel impact op de speler. Kloppen de geluidseffecten? Hebben de modellen dezelfde stijl? Past de muziek bij de game etc…

Ieder detail is enorm belangrijk, en daarom is het ook belangrijk om hiermee rekening te houden.

**2.3.1 Modellen**

De game modellen zijn gemaakt met Blender en hebben zo min mogelijk polygonen. Ik heb zelf gekozen voor een simpel design aangezien ik geen 3d artiest ben. Een simpel design is dan ook makkelijker om op alles toe te passen zodat alle modellen passen binnen de sfeer die mijn game zal geven.

Helaas zijn er enkele zaken waarbij je rekening moet houden met het maken van deze modellen, en deze kunnen een gigantische impact hebben op de speelervaring.

Wanneer je kijkt naar bepaalde games zal je ongetwijfeld wel eens een object gehad hebben dat slechts van 1 kant zichtbaar was, dit fenomeen noemt met normals. Normals bepalen de kijkrichting ten opzichte van de camera, het is dan belangrijk dat de normals altijd naar buiten gericht staan zodat de camera deze kan zien. In de Blender viewport kan je normals van beide kanten zien, maar wanneer je deze dan importeert in Unity kan je slechts 1 kant zien, en dan komt dus door deze normals.

[INSERT PICTURE HERE]

Hetzelfde kan gezegd worden van textures, bepaalde shaders in Blender worden niet ondersteund door de Unity Engine, glas is hier een goed voorbeeld voor, je moet dus een shader gebruiken in Blender die ook door de UnityEngine ondersteund word. Anders krijg je de in fameuze paarse texture te zien die staan voor “Missing texture”.

**2.3.2 Muziek en geluidseffecten.**

Ik kan niet genoeg hameren op het belang van muziek en geluidseffecten, voor iedereen die ooit eens een stomme film gezien heeft is dit zeer duidelijk. Wanneer Charlie Chaplin een hamer laat vallen is het gewoon logisch dat er een geluid met zijn, anders komt het niet natuurlijk over en breekt het de immersie in de game.

Muziek volgt dezelfde redenering, wanneer je achtervolgt wordt door een gevaarlijke alien wil je geen Mega Mindy horen, dat breekt de sfeer en houdt de speler direct tegen van zelfs verder te spelen.

**2.4 Functionele systemen van de Game.**

**2.4.1 Mesh handling**

Wanneer je een videogame speelt kom je ongetwijfeld wel objecten tegen in die virtuele wereld, in 3D terminologie noemen we deze meshes. Een mesh is iets wat door de cpu eerst gedefinieerd wordt met behulp van een draw call en vervolgens doorgestuurd wordt naar de gpu voor te renderen. Het is dus zeer belangrijk dat we hier voorzichtig mee te werk gaan zodat we niet te veel resources gaan renderen.

In Unity is één draw call gelijk aan mesh zonder texture, wanneer je de textures wilt mee renderen zal je telkens 1 call per texture moeten maken, dit klinkt onschuldig maar het is heel makkelijk om 100.000 draw calls naar de gpu te sturen zonder dat je het beseft, een bekende oorzaak hiervan is het niet combineren van je meshes nadat je ze in een 3d programma gemaakt hebt.

Stel nu dat we een kerk zouden modeleren in 3D, dit doen we meestal in verschillende stuken zodat het later aanpasbaar is. Als onze kerk 300 onderliggende stukken bevat met elk 2 textures dan zal Unity ~600 individuele draw calls moeten sturen naar de GPU voor de kerk alleen. Als onze CPU dat niet aan kan ontstaat hetgeen wat men een bottleneck noemt. De GPU kan enkel maar renderen wat de CPU doorstuurt, en als de CPU veel werk heeft zullen we dus heel weinig FPS krijgen.

Voor dit probleem zijn er een paar oplossingen:

We kunnen zo de meshes in ons 3D programma combineren vooraleer we ze exporteren. Op deze manier hebben we slechts 1 draw call voor de mesh en ~ 10 voor de textures.

Een andere oplossing is om via code alle meshes te combineren in een grote mesh via de Unity API, het enigste probleem is dat dit standaard geen rekening houdt met textures en bovendien redelijk moeilijk is om te programmeren zodat de textures ook aan elkaar vasthangen.

Tenslotte heeft Unity zelf een oplossing: Je kan objecten markeren als static, hierdoor zal Unity deze samen in een batch doorsturen voor te renderen, wat ook performanter is.

**2.4.2 Navigatiesysteem.**

Navigatiesystemen zijn 1 van de belangrijkste aspecten om een non playable character (NPC) te laten functioneren in een video game zodat deze in onze virtuele wereld kunnen rondlopen met weinig of geen problemen.

Bijna alle videogame engines bieden hun eigen navigatietools en systemen en die het voor de programmeur makkelijker maken. Het uiteindelijke doel is om een NPC van punt A naar punt B te laten stappen en rekening te houden met de verschillende colliders en objecten rond zich heen.

Natuurlijk zijn er wel enkele termen die vasthangen aan een NPC en navigatie. Een NPC die zal navigeren over een terrein noemen we de “Agent”, deze agent zal dankzij complexe algoritmes zoals bijvoorbeeld AStar (A\*) de kortste weg berekenen van punt A naar punt B. Het is wel belangrijk om te weten dat de A.I. hierdoor veel CPU power zal vragen aangezien hij ook de paden zal uitrekenen die op het einde dood lopen.

Een game environment kan immens groot zijn, hierdoor zal de computer redelijk wat rekenkracht nodig hebben om alles te berekenen, hierdoor zijn er meestal 2 manieren van een agent te laten navigeren.

**Navigation Mesh**

Een navigation mesh is een oppervlak waar de agent kan wandelen. Algoritmes zullen de nodige informatie van een environment selecteren en daarmee geoptimaliseerde routes berekenen. Helaas zijn er natuurlijk enkele nadelen aan dit systeem, zo kan dit enkel maar op een statische wereld gebruikt worden. Vanaf je een wereld hebt die telkens verandert zal je een andere oplossing moeten vinden.

**WayPoints**

Waypoints is een verzameling van punten die zodanig geplaatst is dat een punt minstens met 1 ander punt verbonden is, zo kan je dus van punt A naar B gaan door telkens de verschillende punten te verbinden, ook dit systeem kan enkel werken met een statische wereld.

**Unity’s Pathfinding System**

Unity heeft zelf ook een pathfinding systeem dat je zeer makkelijk kan gebruiken in je eigen videogame.

Unity’s systeem heeft 2 belangrijke componenten:

* Navmesh
* Agent

**Navmesh**

De navmesh of navigationmesh is een game object die we selecteren voor onze agent op te laten navigeren. We beginnen met de mesh te selecteren en dan vragen we aan Unity voor een Navmesh te bakken voor het geselecteerde object. Deze navmesh is statisch, wanneer onze wereld verandert van vorm zullen we deze opnieuw moeten bakken, dat gebeurt ook bij een dynamische wereld.

**Agent**

De agent in Unity wordt gekenmerkt door de NavmeshAgent component dat we vasthangen aan onze NPC, dit zorgt ervoor dat onze NPC toegang krijgt met de Navmesh en dan perfect weet hoe hij zich moet verplaatsen.

De verplaatsing zelf gebeurt door een Vector3 waarbij een 3-dimensionale vector opgeteld wordt en de NPC (de kubus) zich dan verplaatst naar de positie van de target (de cilinder).

A picture containing line chart

Description automatically generated

**Toepassing De Doop**

In de videogame is het niet de bedoeling dat de enemy AI door de muren en door tafels en stoelen kan lopen wanneer hij je achtervolgd, dus het is belangrijk dat we alle objecten een speciaal component geven waarmee de Navigation mesh rekening mee zal houden.

In de onderstaande afbeelding is het duidelijk te zien wat dit inhoudt voor het project. De Kubus of in dit geval de docent zal zich rond het obstakel navigeren om zo naar de speler (De cirkel) te lopen.

A screenshot of a computer

Description automatically generated with medium confidence

Maar we willen een realistische representatie hebben van een docent die de gangen patroeilleerd. Wanneer we deze AI laten doen zou hij overal navigeren zonder een doel of target te hebben. Om dit probleem op te lossen kunnen we interesse punten plaatsen op het speelveld en de AI bepaalde locaties geven die interessant zijn voor te bezoeken.

A screenshot of a video game

Description automatically generatedAls we even terug gaan naar het voorbeeld van Alien Isolation wordt dit duidelijk.

De alien AI weet welke locaties interessant zijn, en hij zal dan ook van de ene locatie naar de andere gaan en kijken of de speler ergens in de beurt is. Voor de alien zijn deze interessepunten terminals en objectives.

In mijn geval zijn de interessepunten de locaties waar de PI’s verstopt liggen.

In mijn onderzoek heb ik hier 2 verschillende manieren voor gevonden. Statische interesse punten en dynamische interesse punten.

**Statische interesse punten**

A picture containing text, monitor, indoor, screenshot

Description automatically generated

Statische interesse punten moeten handmatig in de scene geplaatst worden en de computer zal deze volgen op 2 verschillende manieren: Of de AI zal de lijst voorwaarts afgaan of achterwaarts. Deze AI heeft ook nog een paar instellingen waar je kan instellen dat de AI moet stoppen met wandelen en op de plaats zelf even moet stilstaan, dit is op zich handig voor de AI wat tijd te geven om rond te kijken, zo kan hij zijn hoofd roteren en via een “view cone” de omgeving verkennen om dan vervolgens opnieuw te patrouilleren.

**Dynamische interesse punten**

Dynamische interesse punten werken op een andere manier, je plaatst ze ook in de scene maar er is een link tussen de overlappende punten. Zo kan de AI zelf beslissen waar hij naartoe wil gaan door een willekeurig overlappend punt te verkennen. Dit zorgt ervoor dat de AI realistisch reageert en dit maakt het ook direct moeilijker voor de speler aangezien de AI geen vaste route heeft.

Het probleem met statische punten is dat de AI 1 bepaalde route zal volgen in een vooraf bepaalde volgorde. Deze volgorde is misschien wel willekeurig bepaald maar blijft consistent tijdens de game, iets wat niet voorkomt met dynamische punten.

A screenshot of a computer

Description automatically generated with medium confidence

Op de bovenstaande afbeelding staan de interessante punten met een gele cirkel. Deze cirkel toont aan dat alle rode punten met elkaar verbonden zijn. Dat betekend dus ook dat de A.I. Bij een kruispunt een wilekeurige richting zal volgen, en dus zo compleet willekeurig zal navigeren op de map. Deze zal natuurlijk ook rekening houden met obstakels en een omweg maken.

Het is natuurlijk niet de bedoeling dat de AI telkens hetzelfde pad afloopt, dit is simpel op te lossen met een kost toe te voegen aan iedere node. Zo zal de AI realistisch de omgeving verkennen.

**2.4.3 Raycasting**

Raycasting is voor bijna elke game een belangrijk onderdeel en het belang mag zeker niet onderschat worden, het is belangrijk voor zowel de speler als de NPC’s.

A picture containing indoor, control panel

Description automatically generatedRaycasting kan voorkomen onder verschillende vormen, het openen van een deur, het opnemen van een voorwerp of een AI die de speler “ziet” lopen. Laten we teruggaan naar Alien Isolation met een voorbeeld.

Dit is een computer die de speler moet gebruiken om een deur te openen in Alien Isolation, maar hoe weet het spel precies dat de speler die wilt openen? Of beter gezegd: hoe kan het spel weten dat de speler dicht genoeg is om die computer te gebruiken?

Dat gebeurt via een raycast.

In de doop vind je raycasts terug voor verschillende doeleinden. Het oppakken van een voorwerp, het detecteren van de speler en het openen van deuren voor zowel de AI als de speler.

**2.4.4 Player movement**

De speler moet zich kunnen bewegen in de 3d ruimte van de Doop. Buiten de beweging op de X en Z as zal de speler ook automatisch trappen kunnen oplopen (op de Y as) en kunnen weglopen van de A.I.

In de game is het dan ook belangrijk dat we onderzoek doen naar het mappen van keyboard/controller toetsen met de beweging in game, inclusief de extra toetsen voor de zaklamp, het oprapen van voorwerpen, het bovenhalen van de brochure en de toets voor snel weg te lopen.

Het grootste probleem is het detecteren van AZERTY en QWERTY-toetsenborden. De traditionele bewegingsknoppen op QWERTY en AZERTY verschillen helaas.

A picture containing white, keyboard

Description automatically generated  
In België en Frankrijk zijn we helaas het buitenbeentje i.v.m. de rest van de wereld.   
  
Het is belangrijk dat er een manier is zodat de gebruiker de toetsen kan aanpassen naar wat hij wil, helaas is dat niet zo zelfsprekend in het input systeem van Unity. Gelukkig heeft Unity een nieuw input systeem dat het verplaatsen van toetsen wel toelaat, de implementatie is wel volledig anders maar het nieuwe systeem is veel gebruiksvriendelijker.

**2.4.5 Story Narrative**

Een game heeft net zoals een film of boek een verhaal, je kan de speler moeilijk in het spel zetten zonder een duidelijk doel, of zonder een reden waarom die speler zich daar bevindt in de eerste plaats.

De meest gebruikte manier van het verhaal uit te leggen is door middel van een cutscene, deze kan ondersteund worden door een audio fragment of tekst.  
  
Helaas vragen cutscenes veel tijd en creativiteit om te maken, voor deze game zal ik het dus redelijk simplistisch houden, maar genoeg zodat de speler weet wat hij moet doen. De game zal een geluidsfragment afspelen dat een dialoog is tussen de student en de studentenclub. Op de muur zal ik dan ook de input opties plaatsen.

**2.4.6 Audio**

Audio is een cruciaal deel van een game. Ook al is de gameplay geweldig, zijn de Graphics prachtig kan de muziek en audio een verkooppunt zijn voor een game,

Het is dan ook de bedoeling dat de audio een impact heeft op de speler, wanneer hij iets vastneemt, of een deur opendoet.

Maar audio kan ook gebruikt worden om de speler een signaal te geven van een bepaald gebeurtennis, zo gebruikt deze videogame audio om de speler duidelijk te maken wat de AI doet.

Wanneer de AI zijn patrouille fase uitvoert speelt de theme song af, maar wanneer de AI iemand achtervolgd of de omgeving onderzoekt zal zijn gedrag verschillen en dus ook de muziek veranderen.

**2.4.7 Collection system**

Het verzamelsysteem is wellicht een van de belangrijkste delen van het spel, aangezien het de bedoeling is dat spelers de lay-out van de school leren kennen zullen de te verzamelen objecten op willekeurige plaatsen te vinden moeten zijn, op deze manier zal de speler meerdere keren kunnen spelen en het nog steeds interessant maken.

Natuurlijk is het ook de bedoeling dat de speler weet waar hij deze voorwerpen kan vinden, hiervoor heb ik de brochure gedownload en enkel de vakken van GDM erop geplaatst. Mijn game zal de speler ook naar belangrijke locaties sturen om de student zo kennis te leren maken met alle faciliteiten.

**2.4.8 AI systeem**

De AI is ongetwijfeld de grote ster van deze BAP, hij patrouilleert de school en zal de speler achtervolgen wanneer hij die hoort, ziet of weet dat hij een voorwerp gevonden heeft.

Maar een belangrijke vraag bij deze situaties is: Hoe weet de AI of de speler een voorwerp gevonden heeft? Of een deur opent?  
  
Er zijn verschillende systemen om dit te implementeren, een State machine, een behaviour tree of zelf hard coded.

Unity heeft gelukkig een animator tool die ook dient als Finite State Machine. Een FSM is een systeem in de engine waar je states kan aanmaken en bepalen welke gebeurtenis of actie verantwoordelijk is voor de transitie van de ene state naar de andere state.

De AI van “De doop” heeft 4 states die hij kan nemen.

**Patrol State:**

De patrol state is wanneer de AI de verschillende navigatiepunten afloopt, de AI zal punt per punt afgaan en telkens een kost koppelen aan de punten die hij al genomen heeft. Hierdoor zal de AI rekening houden met de paden die al eens genomen heeft en zal hij telkens een ander pad nemen.

Er zijn slechts 2 signalen die de AI kan opvangen om uit deze state te gaan, ten eerste kan hij de speler rechtstreeks zien bewegen en deze dan achtervolgen door in de “Chase State” te gaan. De tweede optie is dat de AI ergens een geluid opvangt door de speler die te dicht bij de AI loopt of een geluid opvangt van de speler zelf.

**Chase State:**

In de “Chase State” zal de AI rekening houden met een variabele die bijhoudt of de speler in het gezichtsveld van de AI zit. Zolang dat het geval is zal hij de speler achtervolgen. Als de speler niet meer in het gezichtsveld zit zal hij de laatste locatie gebruiken en overgaan naar de “Investigate State”.

**Investigate State:**

In de “Investigation State” zal de AI rondlopen en kijken of hij de speler ziet of als hij een geluid hoort. Na een aantal seconden zoekt hij het dichtst bei zijnde navigatiepunt en gaat hij terug naar de “Patrol State”.

**Investigate Sound State:**

In de “Investigate Sound State” zal de AI naar de locatie gaan waar hij het geluid gehoord heeft. Wanneer hij daar is zal hij een aantal seconden rondkijken en dan terug naar de “Patrol State” gaan.