ATTACHMENT 3

Interview on Game Development & Programming

With: van Hoeylandt, Ken

By: Drift, Beer van der

2/2/2012-2/27/2012

**DUTCH**

* Ken 🡪 Beer:

Beste Beer,

Je vader stuurde mij een brief met het verzoek om m'n hulp te verlenen

aan jouw profielwerkstuk over games ontwerpen/programmeren/produceren.

Zoals je waarschijnlijk al weet van je vader heb ik enkele jaren

gewerkt in de games industrie. (sinds augustus vorig jaar heb ik

echter een andere job, waarin ik niet enkel game-projecten afwerk)

Als je me vragen wil stellen dan mag je dat gerust doen naar dit

email-adres

:)

Groeten,

Ken

* Beer 🡪 Ken

Beste Ken,

Ja dit klopt! Ik ben bezig met een profielwerkstuk over het developen

van games. Hij vertelde mij inderdaad dat je aan een grote game werkte

in Amsterdam, waaruit ik de conclusie trok dat dit Killzone 2 was,

omdat dat rond deze datum was. Nou vraag ik me eerst af wat voor een

positie je hebt in de game industrie over het algemeen. Ben je

designer, programmeur, team manager, alpha tester of misschien wel van

de marketing. Dan kan ik hier mijn vragen op baseren, niet dat ik naar

dingen vraag die je niet kan weten.

Ik ben niet geinteresseert over speciale games maar in het algemeen

over het proces.

Hartstikke bedankt dat je hier even de tijd voor wilt nemen!

Groet, Beer.

* Ken 🡪Beer

Hoi!

Dat klopt, ik heb aan Killzone 2 gewerkt en alsook aan Killzone 3. (in

totaal 6 jaar voor beide projecten samen) Ook heb ik enkele maanden

meegewerkt aan een nog-niet-aangekondigd project, maar dat project heb ik

niet afgewerkt omdat ik van baan veranderd ben.

Ik werk dus nu niet echt meer in de games industrie (alhoewel ons bedrijf

wel games maakt, maar niet hoofdzakelijk) en ben nog steeds als programmeur

aan de slag (maar mijn takenpakket is nu wel breder). Ook bij Guerrilla

Games was ik programmeur en wel specifiek 'game coder'.

Dat wil zeggen dat ik met game designers aan tafel ging zitten om specifiek

gameplay voor hen te implementeren. Ik richtte me vooral op

wapen-gerelateerde code en de HUD, maar had ook met vele andere

gameplay-gerelateerde elementen te maken.

Ik deed echter geen tech code (basissystemen om graphics aan te sturen,

tools, bestandssystemen, etc.) of AI code (path planning, etc.).

Vragen over het proces zijn vast het interessantste :) Ik kan niet

garanderen dat ik alles kan beantwoorden (ik heb namelijk een NDA getekend),

maar de meeste vragen zullen vast geen probleem zijn.

* Beer🡪Ken

Hey!

Oke! Top, nou ik stel maar een paar vragen, kijk maar tot in hoe ver je ze

kan antwoorden.

1. Kan je het algemene proces van het maken van een game kort samenvatten?

(Idee --> Design --> Development --> Playtesten --> Publishen)

2. Hoe wordt het werk verdeeld, zoals teams, taken per team, zoals jij zei

van de wapens en de HUD wat jij deed. Doe je dat met een team of in je

eentje?

3. Welke software gebruikt men om te coderen, debuggen en te designen? Als

dit onder NDA valt: Gebruikt men zelfgemaakte software van de developer zelf

of wordt algemene (professionele) software gebruikt?

4. Als je programmeert aan een game, ben je dan de hele dag bezig als een

full time baan, of is het meer een werk-uit-huis-baantje?

5. Heb je plezier in het programmeren, of voelt het toch meer als werk?

6. Hoe wordt geordend de game in orde gehouden (updates, bugs oplossen)

nadat de game is losgelaten in het publiek?

7. Heb jij nog iets wat echt van belang is met het developen van games wat

meeste mensen over het hoofd zien? Een stap bijvoorbeeld?

Nou hopelijk lukt het antwoorden een beetje, en zijn de vragen niet

onduidelijk

Groet, Beer!

* Ken🡪Beer

Hier zijn de antwoorden!  
  
**1. Het proces:**  
Het proces hangt af van de grootte van het spel en ook afhankelijk van welke platformen je op publiceert. Bepaalde platformen zoals PS3, X-Box en iOS hebben immers een goedkeurings-fase, waarbij de app door de console-fabrikant moet worden gekeurd (en getest).  
In het algemeen klopt je opzet:  
**A.** Spel ontwerpen in grote lijnen (storyline, basis gameplay). Dit kan vooraf gaan aan brainstormsessies voor een nieuw spel.  
Tijdens deze stap gaan artists aan de slag om concepten uit te tekenen aan de hand van een vrij globaal idee van game designers.  
**B.** Speldetails ontwerpen:  
Als een spel met levels werkt, dan gaan level designers dit uitwerken. Hun stukje van het verhaal (en dus een level of een deel van een level) mogen zij uitwerken op zo'n manier dat het aansluit bij het overkoepelende design.  
Er kunnen designers zijn die zich specifiek richting op core mechanics zoals bvb hoe voertuigen zich moeten gedragen in het spel, of bijvoorbeeld hoe wapens werken.  
Bij kleinere games heb je natuurlijk minder designers en gaat een designer meer verantwoordelijkheid krijgen en minder gespecialiseerd werken.  
**C.** Prototyping:  Teams werken samen aan prototypes om game mechanics uit te testen. Zo zie je wat lekker speelt en wat niet.  
**D.** Productie: het design staat (min of meer) vast en programmeurs gaan aan de slag. Game designers werken nog steeds parallel en zullen tot relatief korte tijd voor de lancering van de game nog invloed blijven hebben op het productie-proces.  
**E.** Testen: Afhankelijk van de scope van het project zijn er verschillende test-fases (zoals alpha en beta tests) die al dan niet publiek of privé gebeuren of met een select publiek. Sowieso wordt er tijdens productie ook getest, maar dat is dan vooral binnen het bedrijf en met eigen testers.  
**F.** Publishen: Bij verschillende platformen is er een goedkeuringsproces. Bij Sony is er een TRC(Technical Requirements Checklist) en Microsoft heeft TCR (Technical Certification Requirements) (meer: <http://en.wikipedia.org/wiki/Game_testing> bij "Compliance testing")  
**G.** Patching: bugs fixen wanneer ze opduiken  
**H.** [expansion packs lanceren - en dus het proces grotendeels herhalen]  
  
**Meer** :  
<http://en.wikipedia.org/wiki/Software_release_life_cycle>  
<http://www.pearsonhighered.com/samplechapter/0672326922.pdf>  
[http://web.cs.wpi.edu/~id111x/c05/](http://web.cs.wpi.edu/%7Eid111x/c05/)  
<http://digitalworlds.wordpress.com/2008/04/10/the-process-of-game-creation-the-game-design-document/>  
  
**2. Werkverdeling**  
Bij Guerrilla waren er een verschillende takken. Hopelijk vergeet ik er geen, maar het komt neer op:  
- office (supplies, mensen ontvangen, etc.)  
- human resources (lonen, mensen aannemen)  
- designers  
    - level designers  
    - game (mechanics) designers  
- artists  
    - concept art  
    - texture art  
    - modelling  
    - animators  
- audio:  
    - music  
    - sound effects  
- code:  
    - tech code (basissystemen: graphics, sound, input processing, etc.)  
    - tools  
    - game code  
    - AI code (soms in 1 groep onder game code)  
- QA (Quality Assurance)  
- management:  
    - managers  
    - directors  
    - team leads  
Ik ben er vast een aantal vergeten, maar dit is het hoofdzakelijk.  
Hier staat overigens nog wat info: <http://en.wikipedia.org/wiki/Video_game_development#Development_team> (link heb je vast zelf al gevonden)  
Hoeveel mensen er voor welke rol beschikbaar zijn hangt af van het soort spel. Bij een klein spel gaat bijvoorbeeld misschien een artist én textures maken én modellen én animeren. Maar bij een groot spel worden die taken verdeeld.  
Idem met programmeren: bij minder complexe games (met dus minder werk) zal een programmeur meer taken op zich nemen.  
  
**3. Software**  
Voor **artists**:  
- Vaak Adobe producten (Photoshop, After Effects, etc.)  
Voor (gameplay) **designers**:  
- Photoshop?  
- Een office pakket  
Om te **programmeren:**Hangt af van het platform. Bij console games (PS3, X-Box) wordt vaak(meestal?) geopteerd voor Visual Studio. Bij iOS ben je redelijk gelimiteerd tot xcode.  
Bij Android development werk je meestal met Eclipse, maar in principe kan je ook met andere editors werken. De meeste Android-tools integreren netjes met Eclipse, dus Eclipse is de logische keuze.  
Verder heb je als programmeur vaak ook Photoshop nodig om dingen te controleren of snel nog even bij te werken als iets nét niet goed is.  
Als je een gameplay programmeur beschouwt, dan kan die namelijk in aanraking komen met animatie, geluiden, textures en in principe zowat alle content die in de game zit. Deze programmeur zal dan ook veel meer tools gebruiken dan bijvoorbeeld een tech coder die enkel geluid-gerelateerde systemen maakt en onderhoudt.  
**Verder**:  
- versiebeheerprogramma's (SVN, CVS, Perforce, etc.)  
- projectbeheer (wie wat doet en wanneer)  
- build monitoring (monitor om te zien of automatische tests falen, bvb CruiseControl)  
- etc.  
  
**4.**   
Bij commerciele projecten is het bijna altijd een fulltime job.  
Soms ontstaan er natuurlijk ook hobby-projecten die uitgroeien tot commerciele projecten. In zulk geval wordt er vooral in de vrije tijd ontwikkeld naast een reeds bestaande (al-dan-niet games-) baan.  
  
**5.**  
Als je geen plezier hebt in het programmeren dan hou je het niet vol. Het is immers materie dat een zekere mate van enthousiasme vereist. Meer dan met andere banen die ik heb gehad.  
Ik moet wel toegeven dat ik voor m'n programmeer-carriere meer enthousiast was over programmeren. Maar ik heb alsnog genoeg motivatie om naast m'n job aan eigen projectjes te werken, dus enthousiasme is er zeker nog.  
De focus kan wel verschuiven. (bvb meer gaan richten op UML modelling, meer meewerken aan het game design process, meer richten op systemen neerzetten zoals unit testing, etc.)  
  
**6.**- Crashreports: crashes worden vaak automatisch gelogd en naar de developer gestuurd. Op die manier worden bugs dus geraporteerd en kunnen de programmeurs het problemen proberen te reproduceren om het vervolgens daarna ook op te lossen.  
- Feedback: bugs/crashes worden omschreven door gebruikers en bvb via forums vermeld (kan ook via email of anders, afhankelijk van het project)  
Bij grotere games worden updates met fases uitgevoerd. Daarmee bedoel ik dat er niet elke dag een nieuwe update uitgebracht kan worden omdat er een testfase en goedkeuringsprocess is per update (en dat kost tijd en dus geld). Sowieso wil je als gebruiker niet elke dag gaan updaten voordat je het spel kan spelen.  
Bij kleinere games wordt er vaak makkelijker geupdate. Bij Android games is er bijvoorbeeld geen goedkeuringsproces. Dat zie je vaak ook wel aan de lage kwaliteit van sommige apps in de Android Market. (bvb dat games crashen op bepaalde type toestellen omdat ze daar niet op getest zijn)  
  
Bugs worden vaak bijgehouden in een 'bug tracker'. Dit is een systeem dat registreert wat er fout gaat (of welke features er nog in de game moeten), hoe het kan gereproduceerd worden, hoe vaak het probleem voorkomt, etc.  
Tijdens het ontwikkelprocess verandert de game continue. Elke wijziging wordt in een versiebeheersysteem bijgehouden(zoals SVN, CVS, Perforce, etc). Elke wijziging (in code en/of content) wordt hier in bijgehouden zodat op eender welk moment de game (of een deel ervan) kan worden teruggedraaid naar een oude versie. Het kan ook handig zijn om bugs op te sporen, om te zien wie bijvoorbeeld een stukje code heeft geschreven. Zo kom je er soms achter dat die zwaar irritante bug ontstaan is door een stukje code dat je zélf 5 jaar terug hebt geschreven.  
  
**7.**   
Voor de ontwikkelaars is het belangrijk dat ze beseffen dat het spel vooral LEUK moet zijn. En dan niet enkel voor de ontwikkelaar zelf, maar ook voor de doelgroep van het spel.  
Bij kleinere projecten wordt de testfase niet voldoende serieus genomen. Vaak is er niet voldoende tijd voor een grondige testfase.  
  
Als je toelichting wil of nog vragen hebt, laat het dan gerust weten!

* Beer🡪Ken

Hey!

Echt enorm bedankt dat je het zo uitgebreid hebt verteld, heeft veel geholpen! Gewoon persoonlijke interesse, maar bij Visual Studio, gebruikt men C#, C, Basic? Misschien wel XNA?

Nou top! Zou je nog mij kunnen inlichten over hoe jullie het programmeren aanpakken, of begin je gewoon met een gepland idee, en dan maar aan de slag en door middel van trial & error de uitkomst krijgen die gewild is?

Groet, Beer.

* Ken🡪Beer

Graag gedaan en mooi dat je er wat mee kon :)  
  
Zelf gebruikte ik C++ met Visual Studio voor PS3/PC programming bij Guerrilla.   
Voor cross-platform console games zal er bijna altijd C/C++ geprogrammeerd worden, omdat dit op de alle 'grote' consoles wordt ondersteund (PS3, X-Box 360, Wii).  
Als je enkel voor X-Box 360 ontwikkeld kan je inderdaad voor C# opteren, al dan niet via het XNA framework.  
(XNA is geen taal, maar een framework gebaseerd op ".NET")  
  
Hoe je programmeren aanpakt hangt af van persoon tot persoon en van feature tot feature. Grotere en complexere problemen zal ik eerst ontwerpen in UML(<http://en.wikipedia.org/wiki/Unified_Modeling_Language>)  
Meestal werk ik niet een heel UML model uit(je kan daar heel ver in gaan qua details), maar slechts 1 of enkele 'diagrammen' (er zijn een 15tal soorten diagrammen). Dit kan op zich al erg tijdrovend zijn bij complexe systemen.  
Eventueel zal ik het ontwerp daarna bespreken met een collega om zijn mening te vragen (dat levert vaak interessante input op).  Bij Guerrilla was er een "code review" fase, waarbij het gewoonweg verplicht was om je code door iemand willekeurig in het team na te laten kijken.  
Daarna begin ik met implementeren. Dat gaat meestal erg snel als je eerst een UML diagram hebt gemaakt.  
Bij kleinere opdrachten zal ik echter geen UML diagram maken, omdat ik van die kleinere opdrachten makkelijker een ontwerp in m'n hoofd kan bedenken.  
Waar ik nu werk heb ik echter geen code reviews meer, maar toch zal ik alsnog aan collega's af en toe om input vragen mbt de aanpak van het oplossen van bepaalde problemen.  
  
Trial&error wil je absoluut vermijden:  
Als je een probleem(bug) aanpakt is het belangrijk dat je niet enkel het symptoon aanpakt, maar het kernprobleem zelf.  
Ook met ontwerpen van nieuwe code wil je zeer zeker weten wat je wil maken en waarom het gaat werken: als je immers je code beschikbaar maakt voor de rest van het team, dan gaan zij er potentieel mee aan de slag en zullen zij er verder bovenop bouwen. Als je dan een structurele fout hebt gemaakt in je ontwerp, dan heb je dus een groot probleem, want dan moet je iets opnieuw ontwerpen terwijl er door vele mensen al verder op gebouwd is.  
Hobbyprojectjes en prototyping zijn natuurlijk uitzonderingen, daar kan je prima trial&error dingen doen.