Robin Hjelte, 2012-12-11

Teknik och spel

- Betydelse och konsekvenser i utvecklingsprocesser.

Innehållsförteckning

Introduktion

Att utveckla digitala spel blir allt mer tillgängligt för amatörer genom kraftfulla spelmotorer som är enkla att använda. I dessa får förenklade högnivå-programmeringsspråk en framträdande roll med omfattande bibliotek och ofta många tydliga grafiska verktyg. Detta ger utvecklarna ett färdigt och tydligt arbetsflöde samt mer tid att engagera sig till den faktiska spelutvecklingen istället för att från grunden utveckla teknik för rendering, fysik och andra avancerade system.

I utbyte mot färdiga system ger utvecklarna bort sin totala kontroll över dessa. De förlorar möjlighet att på ett grundläggande plan gå in och optimera eller bestämma över de delar som finns inbyggda. I många fall är tredjepartsprogramvara och spelmotorer såpass komplexa att det krävs oerhörd kompetens bara för att greppa dem och ibland blir de snarare ett hinder och något man måste arbeta runt snarare än en hjälp.

Tillgängligheten hos dessa verktyg har accelererat och bekräftat den marknad som autonoma och individuella (“indie”) utvecklare exploaterat och således slutit en cirkel där ett nytt marknadssegment bundet till dessa inide-utvecklare trätt fram. Komplett med efterfrågan av både produkter och tjänster, som driver utvecklingen av än mer avancerad men förenklad teknik.

Spel är oavsett hur enkel strukturen är, komplexa i den mening spelarna interagerar med dem. Något som ger oförutsebara resultat en speldesigner bara kan gissa sig till. Spel är dessutom ett av de mest avancerade områdena inom programmeringens ingenjörsskap idag. Detta ger att förhållandet mellan spel och teknik blir oerhört komplext, och då specifikt betydelsen av teknik i kontexten mediet och branchen, samt valen av teknik och konsekvenser av dessa val i enskilda företag.

I den här texten kommer jag utforska hur förhållandet ser ut idag och vad företag bör tänka på i valet av teknik samt kort om vilken inverkan detta kan ha på branchen och dess framtid i stort. Jag vill också föra ett resonemang kring kriterier för valet av teknik där ekonomiska, prestandamässiga, syftesmässiga och processmässiga aspekter kommer vägas in.

Forskningssammanställning och relevans

Ämnet är högintressant och har relevans i en värld där spelmediet är i ständig, oerhört snabb utveckling. Spel är större än någonsin och industrin är en av de snabbast växande världen över och den absolut största inom mediaindustrin (Caron, 2008; Baker 2011) vilket öppnat upp för en differentiation av arbetsområdena (vidareuvecklat senare i texten), nya arbetsplatser, nya kunder och nya aktörer. Många nya företag startar och hamnar i en situation där det finns mängder av alternativ och alla val är kanske inte alltid självklara, men har stor inverkan för hela företagets framtid.

Forskning inom spelutveckling är en stor ny del av akademin, men främst ser man till utformning av rent tekniska, specifika områden (implementation av högnivåspråk i spelmotorer, rendering osv.) eller på genrespecifika trender (Massive Multiplayer Games, Serious Games, Social Games osv.). Mer relevant data hittar man däremot i analyser av mediet och dess industriella utveckling.

Martin och Deuze (2009) beskriver i sin avhandling vad som kännetecknar indie-utvecklare. De definieras som inviduella och autonoma företag som ofta består av ett fåtal individer där produkten står i fokus och målet ofta är kopplat till uttryck, kulturspridning och underhållning snarare än ekonomiska drivkrafter (även om dessa också är en grundpelare). En allt större del av den globala spelmarknaden erövras av indierörelsen och den verkar vara här för att stanna. De kommer fram till slutsatsen att indierörelsen inte hade kommit såpass långt utan framförallt teknik som möjliggör enkel digital distribution, men också i stor utsträckning enkla och kraftfulla produktionsverktyg så som gratis open source verktyg och kraftfulla, lättillgängliga spelmotorer.

Värdet av prototyper tidigt i spelprojekt har belysts vid ett flertal gånger (Schell, 2008; Koivisto och Suomela, 2007) och itterativt arbete vid innovation ger både ett bättre slutresultat samt att teamet får större motivation, flexibilitet och kreativitet (Takeuchi och Nonaka, 1986). Något som återspeglas i moderna spelmotorer (Unity är ett praktexempel, men även Unreal har liknande funktioner, om än med något högre tröskel) och ger utrymme för användande av flera verktyg i samma projekt för olika roller i produktionen (dvs. en för prototypande och testande, samt en för själva slutproduktionen).

Blow (2004) skriver om vinsterna och förlusterna med tredjepartsprogramvara och menar att man inför varje inköp av externt producerade moduler eller hela motorer måste göra väldigt noggranna vinst- och förlustkalkyler för att undvika fall där arbetet mot motorn tillsammans med dess kostnad blir större än egenutvecklad teknik. Kontexten Blow befann sig i 2004 är oerhört annorlunda jämfört med nu. Affärsmodellerna då byggde på en väldigt dyr licensiering innan man fick tillgång till att arbeta med motorn över huvud taget, medan den idag ofta bygger på en provisionsmodell där själva utvecklingen och produktionen kan göras gratis. Detta öppnar upp för helt nya förhållanden med stor tillgänglighet för enskilda utvecklare.

Blow beskriver också problemet med avsaknad av spelspecifika verktyg. Många verktyg som återfinns i spelindustrin är tagna mer eller mindre direkt från närliggande brancher, likt film- och mjukvarubrancherna. Exempel på detta är verktyg för 3D-modellering och programmering där dessa ofta saknar egenskaper som efterfrågas av spelutvecklare som att snabbt kunna kompilera och testa sina spel, eller att kunna arbeta flera samtidigt i en fil, eller att kunna bygga terräng optimerat osv.

Denna avsaknad och adaption av verktyg gäller inte enbart mjukvara utan också formella sätt att behandla text och processer (Kreimeier, 2002). Exempel på verktyg som är direkt tagna från filmbranchen är screenplay och story-board vilka fyller viss funktion men inte inkluderar alla aspekter som hör till spelmediet. Begreppet användarvänlighet är också vanligt som

kvalitetssäkring för spel, men mäter egentligen inget annat än hur intuitivt spelet i fråga är.

Kreimeier föreslår Alexandriska mönster som ett samlat verktyg för skrift kring spel och det har hållits resonemang kring hur man kan använda termer som “spelbarhet” eller “spelarvänlighet” med definitioner knutna till spelspecifika egenskaper, som ett bättre mått än användarvänlighet (González Sánchez, Padilla Zea and Gutiérrez, 2009). Skulle dessa standardiseras skulle de antagligen ge uphov till programvara som enklare låter utvecklare applicera verktygen på deras projekt. I nuläget är detta inte aktuellt och varje företag och varje projekt har egna normer för vad som passar det specifika projektet bäst. Detta är en enkel följd av mediets komplexitet och unga ålder; att man inte klart lyckats definiera vad som platsar inom ramarna för spel.

Vad är vad och hur fungerar det?

Nivå på programmeringsspråk

Man beskriver språk i nivåer där man med låga nivåer menar att språket är nära maskinspråket (ettor och nollor) medan ett högnivåspråk är närmare mänskligt språk. Den definitiva skiljelinjen mellan hög och lågnivåspråk finns egentligen inte då den ständigt pressas uppåt med utvecklingen av nya högnivåspråk och hårdvara som låter dessa arbeta utan fördröjning. Dessutom är det inte alltid själva språket som avgör nivån på programmeringen, utan också vilka bibliotek och API som är utvecklade till det projekt man arbetar med.

Det allmänna fallet är att utveckling i lågnivåprogrammering är mer tidskrävande och kräver högre kompetens och förståelse. Alltså dyrare som arbete, men att det arbetar snabbare rent prestandamässigt på datorerna. Högnivåprogrammering å andra sidan är mycket smidigare att arbeta med och går lättare att greppa, men är mer krävande för datorn att klara av.

C++ är ett språk som äger både hög- och lågnivåegenskaper och kan ses som var gränsen går idag. Allt ovan C++ är definitivt högnivå, och alltså inte lika abstraherat som de språk som finns under det.

Det är mer eller mindre standard att man använder lågnivåspråk för krävande uppgifter i stängda system (såsom rendering och fysik) och implementerat stöd för högnivåspråk eller APIer för gameplayprogrammering och scriptning.

Spelmotorer

En spelmotor är sammansatta moduler av källkod som inte direkt interagerar med spelets beteende, mekanik eller miljö, samt generella fysik- och dynamiska system. Det kan röra sig om rendering, fysiksimulation samt andra mer tekniska aspekter. Inom ramarna av en spelmotor applicerar man sedan gameplaykod och scriptning (Lewis och Jacobsen, 2002) vilka definierar själva spelet.

Det är inte en perfekt definition men en tillräckligt god sammanfattning för den här texten. Det är mer eller mindre standard att integrera kraftfulla utvecklingsverktyg i spelmotorer och då också göra dem mer eller mindre genrebestämmda.

Gameplayprogrammering och scriptning

Gameplayprogrammering är den del av programmet som beskriver spelet (dess mekanik) och miljön som spelet utspelas i. Scriptning är enkel programmering som beskriver specifika händelser under spelets gång. Härefter samlas de båda inom begreppet gameplayprogrammering.

Gameplayprogrammering är betydligt enklare än motorprogrammering då det inte är lika abstraherat och ofta (som ovan nämnt) implementeras med ett högnivåsspråk. För att hantera den här typen av programmering krävs inte lika stora kunskaper inom avancerad matematik, men istället behövs en större förståelse för spelmekanik och upplevelsen hos slutanvändaren.

Även om gameplayprogrammering ofta är enklare finns även komplexa moduler inom detta område. Exempelvis AI, som hittills inte lyckats lanseras kommersiellt som generell (läs: tredjeparts) modul.

Processmetodik inom spelutveckling

Det finns egentligen lika många processmetoder som företag, men de allra flesta följer en tydlig modell i grunden och har sedan anpassat den efter behov. Främst är fallet att man anpassat processmodeller som används inom andra industrier till spelmediumet, med vissa skillnader för att passa de situationer som är domänspecifika. Exempel på dessa är vattenfallmetoden och SCRUM. Den första har delat arbetet i moduler, där alla har en deadline och varje modul påbörjas vid den förras färdigställande. Den andra är mer anpassad till hur faktisk mjukvaruproduktion fungerar, och tillåter en stor flexibilitet där arbetet emellan de olika modulerna kan anpassas efter nya data.

Intressant är vad de olika processmodellerna har gemensamt då de behandlar spel, och vad som anses vara essentiella delar av spelutveckling.

Generella inslag

Spelutveckling kan delas upp i ganska definierade steg:

1. Idégenerering.
2. Förproduktion.
3. Produktion.
4. Implementering.
5. Test av spelmekanik och innehåll.
6. Test av teknik.

Förutom dessa tillkommer också Marknadsföring och Lansering, men dessa brukar och bör ligga parallellt med resten av produktionen för att dra ner den tid som behövs innan själva laneringen av spelet.

1. Idégenerering är uppbyggd beroende på syftet med produktionen. Ofta ligger en kreativ idé (vi kan kalla den grundidén) som bas och därefter byggs speldesignen upp i samförstånd med en affärsmodell för spelet i fråga. Det är inte ovanligt att grundidén föregås av en marknadsundersökning man bygger spelet på.

2. Förproduktionen är processen till att nå en färdig design på spelet och tydligt lista allt som ska produceras och implementeras till slutprodukten.

3. Den faktiska produktionen där allt som ska ingå i spelet framställs. Kopplat till en tydlig tidsplan med tydliga milstolpar. Tidigare än så här är det oerhört svårt att tidsbestämma och planera sitt spelprojekt (som refererat av Schell 2008 s.94-95) även om man har en uppskattad plan redan innan.

4. Implementering av producerat innehåll. Där allt sätts samman.

5. Testfas för att se att innehåll och spelmekanik förmedlar tema och estetik enligt syftet med produktionen.

6. Testfas för att se att projektet inte innehåller buggar och att tekniken fungerar syftesenligt.

Marknadsföring och Lansering är arbetet med att sälja spelet och inkluderar allt från att uppmärksamma sin publik om spelet eller företaget, till att ta kontakt och skriva avtal med förläggare och distributörer.

Föredragna strategier

Då spel är väldigt komplexa, och att konsekvenserna av många val är svåra att förutse, krävs en hög flexibilitet och förmågan att snabbt göra ändringar i sin design. Detta ger att det är föredraget att låta de olika stegen inom utvecklingen överlappa varandra (Takeuchi och Nonaka, 1986) och framförallt då hålla en ständig kvalitetssäkring med tester parallellt med resten av utvecklingen. Där är både prototyper och itterativa processer viktiga verktyg (Schell, 2008).

Flexibla processmetoder med team nedbrutna i mindre autonoma grupper är att föredra för så hög och flexibel arbetsinsats som möjligt.

Hantera teknik i spelutveckling

Där Blow delar in programmeringen i två större grupper, menas också två processer. Den där spelmotorn utvecklas och den där spelet i sig utvecklas. Produktionesmässigt innebär

detta att en teknisk grund för spelet måste produceras innan själva spelet i sin tur ska produceras (även om de till viss del kan överlappa varandra). Planen för teknikutveckling måste grundas i designval för spelet, och således ske på sin höjd parallellt med förproduktionen. Teknikutvecklingen bör göras i så dynamiska och flexibla moduler som möjligt för att snabbt kunna göra ändringar när produktionen av spelet kräver det.

Spelmotorer idag

Idag finns det flera segment på marknaden. Å ena sidan finns de dyra och oerhört komplexa och kraftfulla spelmotorerna som också ofta har sitt ursprung som in-house-motorer för egenutvecklade spelprojekt och sedan utvecklats till att blir kommersiella motorer som andra kan ta del av och utveckla sina spel med. Exempel på sådana är Unreal Engine och Cryengine.

Å andra sidan finns mindre spelmotorer som ofta utvecklats antingen som open source, freeware projekt eller som rent kommersiella motorer som möjliggör oerhört enkel utveckling av spel. Dessa är ofta mer anpassade för prototypande, inlärning och amatörer eller så är de tungt genreanpassade. Exempel ur den här ganska omfattande gruppen är GameMaker, Wintermute, Visionare, Adventure Game Studio, HGE osv. Det finns såklart kommersiellt gångbara projekt från dessa motorer också, varav ett av de senaste i raden är Cactus' Hotline Miami (2012), utvecklat i GameMaker.

Till sist finns de motorer som utvecklats i kommerisellt bruk samtidigt som de kliver bort från genrespecifika begränsningar. Dessa har istället anpassats för att passa generella arbetsflöden och underliggande system som ses återkommande i spelproduktioner. Här till räknar jag Unity och Unigine, även om motorer som UDK och Cryenginge börjar röra sig åt denna grupp av motorer.

Fristående moduler

Om beslut är taget om att utveckla en egen spelmotor kan man underlätta arbetet genom att köpa licenser för fristående moduler som exempelvis ljudhantering. Här finns många valmöjligheter inom många områden, men alla har inte nått god kommersiell gångbarhet (Blow, 2004). Mer lyckade områden inkluderar lågnivåhantering av ljud och rendering, samt animationsstöd och scriptspråkimplementering.

Generella drag hos tredjepartsmotorer

Gemensamt hos de flesta av spelmotorerna är deras sätt att hantera kompilering. Även om det finns vissa skillnader är de centrerade åt att på något sätt låta kompilation gå fort och användaren enkelt navigera till rätt punkt i spelet snabbt.

Enkla scriptspråk är implementerade i de flesta motorer, och där det inte finns ersätts de av högnivå APIer och bibliotek som tillåter för snabbare programmering.

De är, med ett par undantagsfall (exempelvis Unity, Unigine och GameMaker), byggda för en viss genre av spel. Vissa av dessa motorer är mer eller mindre anpassade för att klara andra typer av spel, men har fortfarande stor del av sin funktionalitet knuten till ursprungsgenren. Exempelvis är det mer jobb att göra ett spel som inte håller sig till förstapersonsvy i Unreal än tvärtom.

Trender

Om vi först ser åt spelbranchen i helhet pågår ett par trender. Den första är indie-företagens uppvaknande som kan liknas vid en längtan tillbaka till spelmediets rötter (utvecklare skapar spel för likasinnade och delar sitt material med dem). Den andra är en förlängning av den första och innefattar introduktion av spelargenererat material. Spel som Little Big Planet (Sony Computer Entertainment Europe, 2008), Mod nation racers (Sony Online Entertainment, 2010) och The Elder Scrolls V: Skyrim (Bethesda Softworks, 2012) har som en del i deras respektive USP valt att låta stort fokus ligga på spelarnas utnyttjande av medföljande verktyg för att skapa mer innehåll till spelen.

Trender i teknik går åt samma håll och är starkt sammankopplade till branchen i stort. För det första har kraftfulla verktyg blivit oerhört lättillgängliga. Motorer som tidigare var väldigt dyra är nu gratis att använda genom hela produktionstiden. Istället betalar man provision på försäljningen av sina kommersiellt lanserade titlar. För det andra går även kommersiella motorer åt att skapa ett community kring spelskapandet. Unity är ett värdigt exempel på detta som med införandet av deras Asset Store (år 2010) där utvecklare kan dela innehåll (modeller, script, plug-ins osv.), antingen till försäljning eller som fritt material utan avgifter.

Sammanfattning och diskussion

Man kan egentligen sammanfatta texten med ett par meningar: Kommersiella och professionella spel är oerhört komplexa att utveckla från grunden. Den ökade efterfrågan på spel och det ökade intresset för mediet har medfört att det finns fler aktörer på marknaden med nya behov. Den tekniska utvecklingen börjar tillsammans med behovet för nya produkter ge en större lättillgänglighet av teknik, både i införskaffandet och i kompetensen som krävs för att använda dem.

Spel har mångdubblats i sin komplexitet över de senaste decennierna, och många konflikter har uppstått på vägen där verktyg inte fungerar optimalt mot utvecklarnas växande behov. I och med mediets resning (högre intresse för att både att spela och utveckla spel) har en större marknad öppnat upp: en marknad för utvecklingsverktyg. Dessa har tidigare varit något som ofta saknas för utvecklare och något de själva måste ta fram för att kunna bruka. Ofta har då också dessa verktyg stannat inom studiorna och aldrig hittat ut till en större publik.

Det vi ser är en differentiering av marknaden, där utvecklingen delas upp i olika segment. En där ingenjörer utvecklar och säljer spelmotorer och underliggande system, och en där spelutvecklarna koncentrerar sig på själva spelutvecklingen. Det finns fördelar med att utveckla

en egen motor till sina spel, så som att man kan optimera och rikta den precis som man behöver, men i de fallen detta händer fungerar det ungefär likadant, med två olika team som arbetar separat för att först göra en spelmotor och sedan ett spel. Alternativt samma team, fast två faser. Det kräver oerhörda resurser då processen är både komplicerad och tidskrävande.

Ingenjörer har dock fortfarande en roll i själva produktionen, men den är då i arbete mot plug-ins, verktyg och gameplayprogrammering inom spelmotorer som är mer riktade mot det faktiska spelet studion väljer att utveckla. Detta ger för teknisk innovation även inom spelmotorer, även om den ibland är omöjligt med förutbestämda ramar. Är rent teknisk innovation målet med projektet, kanske en tredjepartsmotor är mer ivägen än till hjälp.

Växande tillgänglighet av teknik, då framförallt produktion och lansering, samt initiativ för communitybaserad utveckling (läs: open source och modeller som Unity) har satt igång en utveckling som kan liknas vid en demokratisering av spelmediet, något som i sin tur löser gamla konflikter och ställer nya krav på branchen i helhet.

Detta ger företag många nya möjligheter, där den största är att man väldigt enkelt kan välja kraftfulla verktyg och börja producera från dag ett. En annan stor fördel man får av nuvarande utveckling är tillgänglighet till flera spelmotorer. Man har lätt att hitta en som passar det projekt man arbetar emot (genremässigt, syftesmässigt osv.), och framförallt, man kan använda flera inom samma projekt.

Teknik är precis som alla andra verktyg: De används för specifika endamål. En motor som är lämplig för slutprodukten på grund av optimering och enkelt workflow kanske inte är den mest optimala för all form av prototyparbete. Genom att välja en annan, enklare motor som snabbare kan nå testvärdiga prototyper kan skynda på arbetet ordentligt och höja kvaliten på slutprodukten utan att ge för stora ekonomiska avtryck för projektet.

Avancerade open source-produkter ger dessutom företag tillgång till verktyg av branchstandard för inga pengar alls. Man betalar dock ett annat pris. Open source projekt är inte knutna till att tjäna pengar och har ofta således inte samma krav att kunna tillgodose sin kund (i det här fallet spelutvecklaren) med varken den bästa produkten, med det bästa arbetsflödet, på marknaden eller att kunna ge kundtjänst över huvud taget. Detta till trots är spelmediet drivet av många entusiaster som gärna hjälper varandra genom just open source-projekt och de verktyg som finns är en central del i många projekt.

Slutsats och kriterier

Frågan allt kommer ner till är vad ett företag för tänka på i valet av teknik till spelprojekt. Det framgår ganska tydligt att det finns många aspekter av ämnet, men det hela kan förenklas ner till två frågor:

1. Vad kostar minst?

2. Förbättrar mitt val upplevelsen hos spelaren, dvs. blir mitt spel ett bättre spel?

För att kunna svara på dessa två frågor föreslår jag att man går igenom följande steg, listade utan inbördes ordning.

Kunskapsinventering: Vilken kompetens finns inom teamet både i fråga om programmeringsnivå och verktyg medlemmarna tidigare har arbetat med? Valet blir långsiktigt både för att kompetensen ökar inom verktyget och för att många motorer bara kräver en engångsbetalning för licenser.

Features: Vad som ska finnas med i spelet. Svara på följande frågor och väg dem mot hur svårt det är med den teknik som övervägs. Ska spelet vara i 2D eller 3D, ska fokus ligga på hyperrealistisk rendering eller en simplare grafikstil? Hur mycket behöver jag optimera mitt spel? Vad ska man kunna göra i spelet? Är spelet för fler spelare? I så fall, över nätverk? Vad krävs för att jag ska kunna implementera detta i spelet med den teknik jag överväger?

Vilka ska implementera? Genom att gå igenom teamet och se vilka som ska vara med och implementera producerat material direkt i spelmotorn kan man bestämma hur mycket tid implementation för någon oinsatt måste ta. Ska till exempel någon som jobbar med ljud- och musikproduktion själv ljudlägga delar av spelet är verktyg med tydliga grafiska element, ljudtekniska filter och intuitiva editorer bra verktyg att leta efter.

Genre: Vilken genre spelet har är ytterst avgörande då mycket teknik är riktad just till en specifik genre av spel, där man kan få mycket gratis. Man måste vara uppmärksam och se om de features man vill ha extra, kommer motarbetas eller hjälpas av motorn i fråga. Om de motarbetas bör man överväga fler alternativ.

Syfte med spelet: Med syfte menar jag om spelet ska vara kommersiellt eller i lärandesyfte eller ett konstprojekt etc. Beroende på vilket finns det olika argument till varför ett val är bra eller inte. Open source är till exempel något mindre säkert, men kostar inget och kan därför vara bättre i ett icke-kommersiellt syfte. Konst kan handla om att bryta konventioner och kan således ställa krav mot varandra och vända kriterier uppochned.

Vilka plattformar ska spelet byggas till: Alla motorer stödjer inte naturligt att bygga ett spel till olika plattformar. Fundera tidigt på om spelet i fråga ska finnas på mobila plattformar, eller på PC / Mac eller på konsol eller på flera av dessa. Se också till hur optimerad tekniken i fråga är till de riktade plattformarna om den skulle klara av att bygga till dem.

Vilka motorer och verktyg kan jag använda i utvecklingsprocessen: Se tidigt till att veta hur prototypandet av spelet kan gå till och vilka verktyg som behövs i processen. Välj teknik som underlättar arbetet så mycket som möjligt och snabbar på processer som är svåra att tidsuppfatta (främst förproduktionen).

Vad kostar hjälpen och vad kostar processen? Se hur mycket hjälp du kan få från ett inköp eller inlärning av tredjepartsprogramvara. Omsätt den hjälpen till faktiska mantimmar det hade tagit att på egen hand få en ekvivalent upplevelse hos spelaren. Är kostnaden mindre hos tekniken som tänkts köpas in än hos den tid det hade tagit att bygga all programmering från grunden är det antagligen värt att köpa in tekniken. Glöm inte att ta i beaktande den tid det tar att lära sig det nya verktyget eller motorn för att få det resultat man önskar, och glöm inte att det ofta finns andra vägar till liknande upplevelser och tänk noga igenom dessa alternativ.

Spel är komplexa. Varje spel kräver sina egna verktyg och närmanden i form av produktionsmodeller och tekniska val. Dessutom är alla företag olika och har olika kompetensområden, affärsplaner och mål med sitt arbete.

Mitt förslag är att man väger samman de ovan nämnda faktorerna för att på så sätt skaffa sig en tydlig och omfattande förståelse och bild över det arbete man tänkt utföra och på den grunden göra sitt val för att fortsätta med den faktiska spelutvecklingen. Då med en större förståelse för processen och tydligare egna kriterier för vad som krävs till följande produktioner.

Referenser

- Blow J. (2004) *Game Development: Harder Than You Think,* QUEUE.
- Baker B. L. (2011) *Factbox: A look at the $65 billion video games industry,* Reuters. Tillgänglig på internet: http://uk.reuters.com/article/2011/06/06/us-videogames-factbox-idUKTRE75552I20110606
- Caron F. (2008) *Gaming expected to be a $68 billion business by 2012,* Ars Technica. Tillgänglig på internet: http://arstechnica.com/gaming/2008/06/gaming-expected-to-be-a-68-billion-business-by-2012/
- Koivisto E. M. I & Suomela R. (2007) *Using Prototypes in Early Pervasive Game Development,* the Association for Computing Machinery, inc.
- Kreimeier B. (2002) *The Case For Game Design Patterns,* Gamasutra.
- Lewis M. & Jacobson J. (2002) *Game Engines In Scientific Research,* Communications of The ACM.
- Martin C. B. & Deuze M. (2009) *The Independent Production of Culture: A Digital Games Case Study,* SAGE.
- Sánchez J. L. G., Zea N. P. and Gutiérrez F. L. (2009) *From Usability to Playability: Introduction to Player-Centred Video Game Development Process,* Software Engineering Department, University of Granada.
- Schell J. (2008) *The Art of Game Design: A Book of Lenses,* Morgan Kaufmann Publishers.
- Takeuchi H. & Nonaka I. (1984) *The new new product development game,* Colloquium on Productivity and Technology, Harvard Business School.