Concepte privind dezvoltarea jocurilor electronice

Alexandru Angelescu
Universitatea "Alexandru Ioan Cuza" Iaşi
Facultatea de Informatică
Str. General Berthelot, nr. 16, 700483, Iaşi
alexandru.angelescu@info.uaic.ro

Sabin-Corneliu Buraga
Universitatea "Alexandru Ioan Cuza" Iaşi
Facultatea de Informatică
Str. General Berthelot, nr. 16, 700483, Iaşi
busaco@info.uaic.ro

REZUMAT

Lucrarea reprezintă un studiu de caz ce oferă o imagine de ansamblu asupra principalelor concepte privind dezvoltarea jocurilor electronice şi a modalităţilor prin care acestea sunt puse în practică de către dezvoltatorii de jocuri independenţi. Aspectele prezentate vor fi evidenţiate cu ajutorul unui proiect original, Beneath a Crimson Sky, un simulator de zbor dezvoltat utilizând *framework*-ul Microsoft XNA.

Cuvinte cheie

Jocuri electronice, Microsoft XNA, dezvoltarea jocurilor video, *indie games*, *game design*.

Clasificare ACM

H5.1. [Information Interfaces and Presentation]: Multimedia Information Systems – *Artificial, augmented, and virtual realities.*

INTRODUCERE

Vânzările de jocuri electronice au totalizat numai în Statele Unite ale Americii peste nouă miliarde de dolari în fiecare dintre ultimii trei ani [12]. Industria software a jocurilor video este responsabilă pentru mai mult de şase miliarde de dolari din acest total. În perioada 2008-2010, rata de creştere a profitului a fost de cel puţin 10%, iar în 2009, contribuţia industriei software pentru amuzament (*entertainment software*) la venitul intern brut al Statelor Unite a ajuns la 4,9 miliarde de dolari [10, 12].

Succesul de amploare înregistrat de industrie a creat o piaţă de desfacere uriaşă şi a condus la apariţia unei oportunităţi pentru dezvoltatorii de jocuri electronice independenţi de a crea şi distribui propriile titluri fără ajutorul unei companii de renume. Cu toate că există şi firme mari care crează jocuri independente, acestea sunt de obicei dezvoltate de către o singură persoană sau de către echipe mici de până la zece persoane, în funcţie de complexitatea proiectului. Durata de timp necesară creării unui joc independent poate varia de la câteva ore sau zile până la câţiva ani, în funcţie de complexitate, numărul de participanţi şi de obiectivele propuse.

O dată cu apariţia canalelor de distribuţie digitală, precum Microsoft Xbox LIVE Community Games, dezvoltarea de jocuri electronice a fost rapid democratizată. Dezvoltatorii care aveau nevoie de contracte de publicare pentru ca jocurile lor să ajungă pe piaţa consolelor de jocuri, au putut să-şi distribuie creaţiile fără ajutor extern, iar acest lucru a condus la apariţia unui nou val de independenţă şi creativitate în industrie, numit mişcarea *indie games* (din engl. *independent games*).

În trecut, dezvoltatorii independenţi se simţeau constrânşi în munca lor, deoarece marile companii de distribuţie a jocurilor electronice contractau exclusiv titluri cu şanse mari de a aduce profit. Mai mult, pentru a garanta investiţia, erau preferate proiectele asemănătoare cu titlurile anterioare, care înregistraseră deja succes pe piaţă. Această lipsă de inovaţie în design-ul jocurilor electronice a reprezentat un alt factor determinant în succesul obţinut de către dezvoltatorii *indie* [8].

În prezent, ne aflăm într-o perioadă extrem de favorabilă pentru jocurile electronice: grafică 3D optimizată prin algoritmi de procesare complecşi şi prezentată în *High Definition*, conţinut bogat şi imersiv, *multiplayer* axat pe socializare şi creare de conţinut de către jucători, modalităţi de interacţiune bazate pe recunoaşterea gesturilor şi a comenzilor vocale. Piaţa jocurilor video este asaltată de titluri cu un standard calitativ deosebit de ridicat în fiecare lună. Cu toate că acest lucru este de dorit, în special pentru utilizatorii finali, această dezvoltare rapidă poate fi uşor descurajantă pentru dezvoltatorii de jocuri aspiranţi, care au ambiţii ridicate şi, uneori, experienţă insuficientă.

Cu ajutorul Microsoft XNA, pentru prima dată, un dezvoltator de jocuri neprofesionist poate crea jocuri *single player* şi *multiplayer* ce pot rula atât pe PC-uri sau pe console Xbox 360, cât şi pe dispozitive Zune sau Windows Phone 7. Doar cifrele de mai sus pot fi un motiv suficient de bun pentru ca cineva să fie interesat să înveţe XNA şi să devină un dezvoltator de jocuri, încercând astfel să obţină o cotă din această piaţă mai profitabilă decât cea a industriei de filme de la Hollywood.

Unele lucruri nu trebuie ascunse însă: din păcate, există puţine oportunităţi în acest domeniu, existând un singur post de programator de jocuri pentru fiecare o mie de posturi de programatori „din viaţa reală". Mai mult decât atât, în medie, industria jocurilor video îşi plăteşte programatorii cu mai puţin decât alte industrii.

Există şi veşti bune pentru cei care încă mai cred că o carieră de dezvoltator de jocuri este interesantă şi profitabilă: o dată ce Microsoft a deschis serviciul LIVE Marketplace, permiţând oricui să-şi vândă propriile jocuri

către alţi membri ai LIVE, s-a creat o piaţă de desfacere pentru jocurile independente cu zece milioane de potenţiali clienţi (ianuarie 2008).

Când Microsoft a lansat XNA în decembrie 2006, a devenit imediat clar că această nouă tehnologie va avea un impact major asupra posibilităţilor dezvoltatorilor de jocuri. XNA a fost creat de la început cu uşurinţa utilizării în minte, fără a sacrifica în acelaşi timp performanţa şi capacităţile de a atinge acest ţel. În cursul unui singur an, în jurul XNA a luat naştere o largă comunitate de utilizatori, astfel că cineva interesat poate găsi exemple de cod pe un număr vast de situri, poate pune întrebări pe unul dintre forum-uri sau poate să se întâlnească într-un XNA User Group cu alţi dezvoltatori care îi împărtăşesc pasiunea.

Figura 1. Ecranul principal al aplicației Beneath a Crimson Sky

MICROSOFT XNA FRAMEWORK

XNA este un joc de cuvinte, fiind o prescurtare de la "**X**NA's **N**ot **A**cronymed" (XNA nu este un acronim). Conceptul inovator propus de XNA, de a oferi unui utilizator obişnuit puterea de a-şi crea propriile jocuri pentru PC, Xbox 360, Zune și dispozitive Windows Phone este o importantă inovație tehnologică, care este însoţită de eforturile multiple din partea Microsoft de a pune bazele unei comunităţi active pentru creatorii de jocuri (reunind astfel comunitatea din domeniul programării de jocuri pentru Windows sau Zune cu cea pentru consola Xbox 360, și, o dată cu cea mai recentă versiune, cu cea pentru Windows Phone 7) şi de a înfiinţa programe academice care să sprijine instituţiile care doresc să creeze cursuri folosind capacităţile acestei console [9].

Aceste eforturi devin evidente prin prisma faptului că Microsoft XNA Game Studio poate fi descărcat gratuit de pe situl oficial [1]. Compania oferă gratis, de asemenea, conţinut pentru jocuri, incluzând tutoriale video, *starter kits* (jocuri complete ce pot fi personalizate liber), mostre şi alte tipuri de conţinut ajutător în cadrul App Hub [1], portalul oficial al comunității utilizatorilor XNA.

Ultimul pas în popularizarea serviciului Microsoft LIVE ca un "YouTube pentru jocuri" a fost oferirea posibilității de a *upload*-a jocurile create în cadrul Xbox LIVE şi de a le distribui sau de a le vinde oricărei persoane din lume ce posedă o conexiune LIVE. Având această oportunitate, se explică entuziasmul crescând al comunității programatorilor de jocuri neprofesionişti, atât pentru lansarea XNA Game Studio, cât şi pentru actualizările frecvente constând în conţinut nou de pe situl App Hub (fostul XNA Creator's Club).

Marele secret din spatele succesului XNA este simplitatea sa, mai substanţială decât a altor interfeţe de programare a aplicaţiilor (*Application Programming Interface* sau *API*) pentru software pentru console şi chiar a API-urilor pentru programarea de jocuri sub Windows, mulţumită abstractizării pe care o oferă pentru unele detalii de implementare, care, în alte API-uri, trebuie tratate de programator. XNA foloseşte acelaşi mediu de dezvoltare integrat (*Integrated Development Environment* sau *IDE*) – XNA Game Studio Express – şi acelaşi *framework* pentru dezvoltarea jocurilor, atât pentru platforma Windows cât şi pentru consola Xbox 360 sau pentru dispozitivele Zune sau Windows Phone 7, asigurând astfel un grad ridicat de compatibilitate. Cu toate acestea, există unele diferenţe în nivelurile de bază; de exemplu, consola rulează o versiune compactă a *framework*-ului .NET, astfel încât nu toate funcţiile disponibile în Windows vor rula şi pe Xbox 360.

BENEATH A CRIMSON SKY

Jocul Beneath a Crimson Sky (**Figura 1**) a fost creat pentru a ilustra concret potenţialul *framework*-ului XNA şi pentru a exemplifica o serie de noţiuni privind dezvoltarea jocurilor video. Aplicația a fost concepută pentru a surprinde cât mai multe dintre conceptele de programare a jocurilor 3D, cât și pentru a demonstra că XNA poate fi utilizat în realizarea de jocuri cu un grad ridicat de complexitate și cu un nivel tehnic al execuției apropiat de cele realizate de dezvoltatorii profesioniști de jocuri.

Aplicaţia reprezintă un simulator de luptă aeriană, plasat în timpul celui de-al Doilea Război Mondial, în care jucătorii au posibilitatea de a pilota avioane modelate după aparate de zbor reale şi de a îndeplini misiuni având diferite obiective. Jocul poate fi considerat un simulator de zbor de tip „*arcade*”, deoarece gradul de realism în redarea mecanicilor de zbor este menţinut la un nivel mediu; astfel, atenţia jucătorului este concentrată în permanenţă asupra intensităţii luptelor aeriene şi mai puţin pe detaliile legate de comportamentul aparatelor de zbor.

Această tematică a fost aleasă pentru că dezvoltarea unui simulator de zbor permite înglobarea unui număr mare de componente tehnice ce pot fi dezvoltate în XNA (teren tridimensional, sisteme de particule, utilizarea tehnicii *billboarding*, manipularea obiectelor 3D), dar și pentru că jocurile ce simulează luptele aeriene nu sunt foarte numeroase în peisajul jocurilor dezvoltate în XNA.

Componenta *single player* (joc individual) (**Figura 2**) [2] este reprezentată de modul *Campaign* format din două campanii, una de partea Aliaţilor şi una de partea Axei, fiecare cuprinzând câte patru misiuni cu diverse obiective de îndeplinit (doborârea tuturor inamicilor, supravieţuirea pentru o anumită perioadă de timp ş.a.). Jucătorii pot alege oricare dintre cele două campanii, pot selecta o misiune disponibilă (acestea sunt deblocate pe măsură ce misiunile existente sunt îndeplinite), unul dintre cele trei grade de dificultate (pilot amator, experimentat sau as) şi îşi pot alege avionul cu care să parcurgă misiunea dorită.

Când rulează pentru prima dată aplicaţia, fiecare jucător îşi va putea crea un profil de pilot, care va memora diferite statistici despre performanţele sale (timpul petrecut în joc, numărul de inamici doborâţi, acurateţea tirului etc.). Prin îndeplinirea obiectivelor din cadrul misiunilor, jucătorii vor acumula experienţă (care se traduce prin dobândirea unui grad militar ce conferă dreptul de a debloca noi avioane) şi *Upgrade Points* (moneda din joc care permite cumpărarea de îmbunătăţiri [3] pentru toate avioanele deblocate de jucător).

După îndeplinirea unei misiuni în modul *Campaign*, aceasta va putea fi rejucată la un grad mai avansat de dificultate pentru a câştiga experienţă sau *Upgrade Points* suplimentare. Progresul jucătorilor în joc este marcat şi prin dobândirea de medalii pentru îndeplinirea misiunilor şi a unor *achievement*-uri (diverse obiective ce pot fi îndeplinite în joc – distrugerea unui anumit număr de avioane inamice, spre exemplu) [2], asemănătoare cu cele din jocurile pentru consolele Xbox 360 sau PlayStation 3.

Figura 2. Interfața aplicației în modul single player

Jocul are, de asemenea, şi o componentă *multiplayer* (joc cu mai mulți participanți) [2]: jucătorii pot lua parte în modul *Split Screen* (imaginea de pe ecran este împărţită în jumătate pe orizontală) în dueluri aeriene (*dogfights*) împotriva unui alt jucător uman folosind acelaşi calculator (**Figura 3**). După ce fiecare își alege tipul de avion dorit, cei doi vor participa într-un meci unu la unu, compus din mai multe etape, în care fiecare trebuie să-şi doboare

adversarul. La finalul meciului, jucătorul care a acumulat cele mai multe victorii în acea partidă câştigă.

Aplicația Beneath a Crimson Sky este însoțită de manualul jocului [3] şi de un videoclip de prezentare detaliat [2], compus în întregime din secvențe din joc.

ARHITECTURA APLICAȚIEI

Proiectul prezentat utilizează un motor de joc (*game engine*) special creat, numit Crimson Engine, care implementează arhitectura *Model View Controller* (sau MVC). Acesta beneficiază de o structură compartimentată, asemănătoare cu soluția implicită propusă de XNA, care oferă o serie de avantaje în construcția aplicației: extensibilitate, robustețe, flexibilitate şi posibilitatea reutilizării componentelor în alte proiecte. Crimson Engine este format din:

- motorul grafic al jocului (*rendering engine* sau *renderer*) care administrează toate obiectele necesare creării universului jocului: camerele de luat vederi, sursele de lumină, decorul tridimensional şi modelele 3D care îl populează;
- sistemul de administrare a stării jocului (*game state management system*) care se ocupă de tranzițiile între stările automatului jocului (diferitele meniuri și ecrane ale jocului);
- motorul fizic al jocului (*physics engine*) ce administrează deplasarea obiectelor şi detectarea sau procesarea coliziunilor;
- componentele ce țin de logica aplicației (scenariile misiunilor, inteligența artificială etc.);
- sistemul de memorare a datelor necesare jocului (profilurile jucătorilor, caracteristicile avioanelor, informații privind misiunile, medaliile, obiectivele și gradele militare sau fișierele de configurare).

Terenul tridimensional

Deoarece într-un simulator de zbor acțiunea se desfășoară, de obicei, într-un ritm alert, numărul de cadre afișate pe secundă (*frame rate*) de motorul grafic trebuie să fie foarte mare. Pentru a obține performanțe mai ridicate, a fost eliminată o parte din suprasarcina computațională creată de redarea unui teren de mari dimensiuni prin folosirea unui *quadtree* (o structură de date de tip arbore, în care fiecare nod are exact patru descendenți) [7]. Utilizarea unui *quadtree* este o tehnică des întâlnită în programarea jocurilor, ce presupune împărțirea repetată a terenului în câte patru porțiuni egale (*quads*), până când se atinge o anume dimensiune pentru fiecare subdiviziune. Principalul avantaj apare la redarea terenului, deoarece vor fi desenate doar subdiviziunile care se află în raza vizuală a camerei de luat vederi. Similar cu tehnica selectării fațetelor invizibile (*backface culling*) existentă în XNA [11], în componenta de teren a fost implementată o metodologie pentru selectarea automată a *quad*-urilor care nu sunt vizibile pentru obiectivul camerei.

Implementarea defectuoasă a unui *quadtree* poate crea însă probleme serioase de performanță: timpi de încărcare semnificativ mai mari, un număr foarte mare de apeluri către metodele care desenează primitivele sau desenarea multor porțiuni din teren atunci când camera care filmează este poziționată paralel cu terenul, foarte aproape de el.

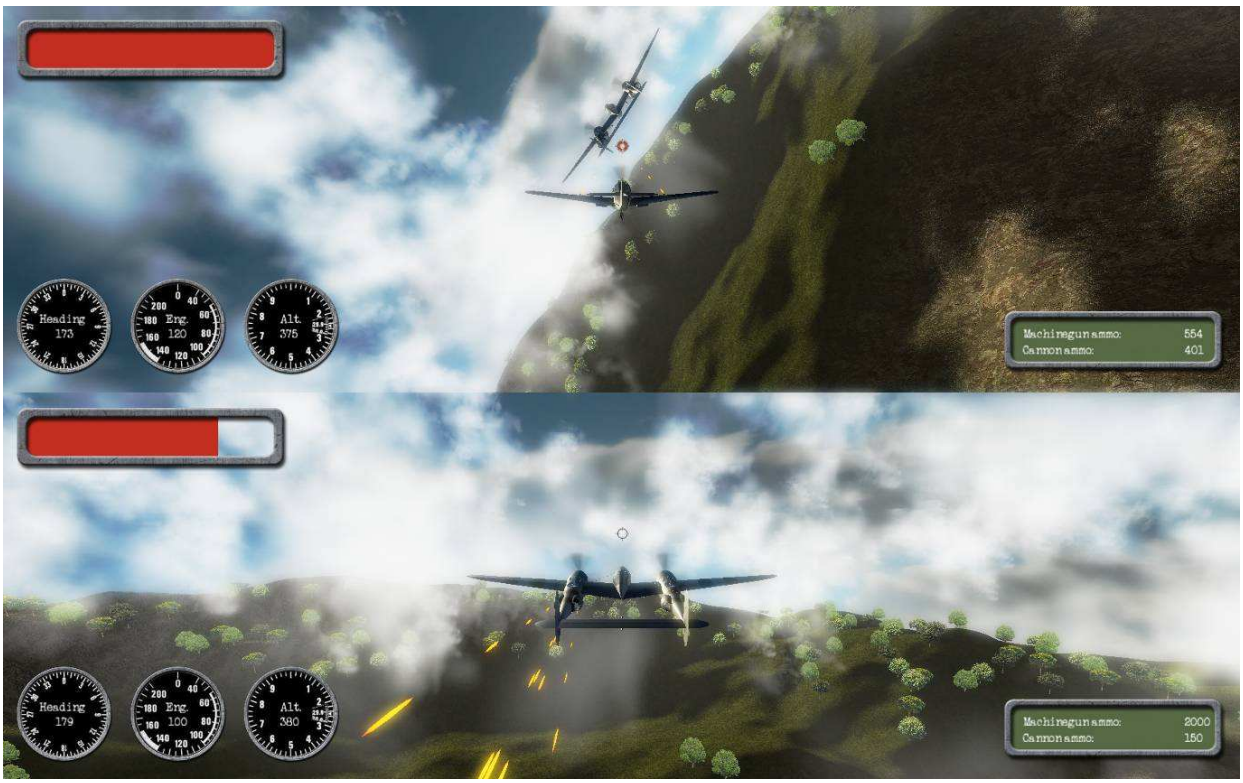

Figura 3. Interfața aplicației în modul split screen

Terenul utilizat în aplicație este generat pe baza unui fișier ce memorează o hartă de înălțimi (*height map*). Deoarece, la construirea unui teren dintr-un *height map* pot apărea porțiuni cu mici discrepanțe de înălțime, după generarea sa

este aplicat un proces de netezire a terenului (*terrain smoothing*) pentru a conferi naturalețe reliefului obținut. Numărul de iterații ale procesului este stabilit, de obicei, în urma unui compromis între aspectul final al terenului și pierderea de performanță cauzată de execuția procesului [9].

Pentru a imprima terenului un grad mai mare de realism, acesta beneficiază de iluminare difuză, reflectantă și ambientală (inclusiv simularea unui ciclu zi – noapte cu durată variabilă), iar umbrele create de teren răspund la iluminarea dinamică în timp real. La desenarea terenului sunt folosite, de asemenea, tehnicile *normal mapping* și *multitexturing* [11] pentru a îmbunătăți rezultatul final.

O altă componentă importantă a motorului grafic al aplicației este cea care desenează întinderile de apă ce însoțesc un teren tridimensional. Crimson Engine implementează efecte de reflexie și refracție (avioanele care se deplasează deasupra apei, picăturile de ploaie sau fulgii de zăpadă sunt reflectați pe suprafața ei, iar un avion prăbușit în apă va fi văzut refractat). Caracteristicile apei pot fi modificate printr-o serie de parametri: viteză, direcția vântului, vâscozitate sau apariția undelor pe suprafața ei.

Utilizarea tehnicii billboarding

Fiind un simulator de zbor, acțiunea se desfășoară într-un spațiu deschis care, pentru a părea cât mai real, trebuie să fie populat cu cât mai multe detalii. Unul dintre elementele de decor utilizate în aplicația de față este vegetația. Însă, pentru ca o pădure să pară autentică, ar trebui desenați sute sau mii de copaci. Nu se pune însă problema desenării atât de multor copaci ca obiecte 3D, deoarece astfel s-ar încetini enorm aplicația. Această problemă este ușor rezolvată prin tehnica *billboarding* [7], în care un obiect 3D este înlocuit cu o imagine 2D (*billboard*) pentru a adăuga detalii vizuale fără a încetini aplicația. Problema principală în utilizarea unui *billboard* apare atunci când camera de luat vederi este poziționată lângă el, pentru că privitorul își dă seama că privește o simplă imagine 2D (**Figura 4)**.

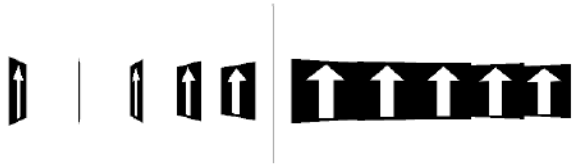

Figura 4. Imagini 2D într-o scenă 3D: plasate aleator (stânga) sau prin tehnica billboarding (dreapta) [7]

În principiu, pentru a rezolva această problemă, se definesc câte două triunghiuri în spațiul 3D pentru fiecare *billboard* afișat, care vor fi rotite în permanență în concordanță cu poziția camerei, astfel încât imaginea 2D este întotdeauna îndreptată către obiectiv (partea dreaptă a **Figurii 4**) [7]. *Framework*-ul XNA conține funcționalități care calculează pozițiile celor șase puncte ce formează cele două triunghiuri pentru fiecare imagine rotită.

Această abordare solicită însă intensiv procesorul și este destul de ineficientă din punctul de vedere al utilizării resurselor plăcii grafice. Deoarece CPU-ul este un procesor având atribuții generale, acesta nu este optimizat pentru calcule cum sunt cele legate de rotirea triunghiurilor ce formează un *billboard*. Astfel de sarcini paralizează activitatea procesorului și limitează accesul altor procese critice pentru joc la el.

Pentru a se obține un spor de performanță semnificativ, în aplicația Beneath a Crimson Sky, punctele ce formează triunghiurile *billboard*-ului sunt stocate în memoria statică rapidă a plăcii grafice, iar calculele au fost efectuate de către GPU prin intermediul unui program de tip *vertex shader* [9]. Dintre beneficiile câștigate, menționăm:

- traficul redus pe magistrala PCI-Express sau AGP;
- timpul necesar calculelor este semnificativ redus (GPU-ul este optimizat pentru acest tip de calcule);
- nici o instrucțiune necesară tehnicii *billboarding* nu este executată în CPU, astfel că procesorul este disponibil pentru alte sarcini mai importante.

Tehnica *billboarding* este utilizată în aplicația creată pentru a popula terenul de joc cu diferite tipuri de vegetație (iarbă, copaci, arbuști) și pentru a simula un sistem de nori volumetrici (**Figura 5**). Trebuie menționat că pentru a obține iluzia de profunzime și transparență a norilor, tehnica *billboarding* a trebuit combinată cu folosirea unor imagini 2D punctiforme (*point sprites*). Norii aflați mai aproape de cameră vor fi desenați folosind *billboard*-uri, iar cei aflați în depărtare vor fi *sprite*-uri punctiforme. Astfel s-au obținut rezultate mai bune, atât din punct de vedere estetic, cât și din punct de vedere al performanței (tehnica *billboarding* este de cele mai multe ori puțin mai costisitoare din punctul de vedere al performanței) [7]. Pentru a ușura procesul de desenare, distanța la care se face trecerea de la o tehnică la alta poate fi setată de programator sau există posibilitatea de a folosi doar una dintre cele două tehnici pentru întregul sistem.

Figura 5. Vegetație și nori creați prin tehnica billboarding

Sisteme de particule

Utilizarea unui sistem de particule reprezintă o tehnică prin care se simulează diferite fenomene vizuale (foc, explozii, fum, ceață, ploaie, zăpadă) care nu pot fi reproduse ușor prin metode convenționale pentru desenare [11].

La fel ca în cazul folosirii tehnicii *billboarding*, atunci când se afișează un număr mare de particule, jocul poate fi încetinit considerabil. Volumul de procesare necesar actualizării sistemului de particule și transferarea celor mai recente poziții ale particulelor către GPU spre desenare pot conduce la pierderi mari de performanță. De aceea, animarea particulelor se efectuează, de obicei, în

GPU, pentru ca sarcina procesorului central să rămână relativ redusă, indiferent de dimensiunile sistemului de particule prezent în aplicație.

La crearea unei noi particule, procesorul introduce într-o structură de date de tip punct, poziția, viteza și momentul de timp la care particula a fost creată. După ce acest punct este încărcat în GPU, procesorul nu va mai interacționa niciodată cu el. În fiecare cadru, atunci când GPU-ul desenează particulele, el calculează vârsta fiecăreia, comparând momentul creării ei (memorat anterior) cu momentul curent de timp. Cunoscând poziția de start, viteza și vârsta, se poate calcula poziția curentă și se poate desena un *sprite* punctiform în acea locație.

Noile particule sunt întotdeauna adăugate la finalul *buffer*-ului de puncte, iar cele vechi sunt eliminate de la începutul său. Deoarece toate particulele au aceeași durată de viață, toate particulele active vor fi întotdeauna grupate împreună într-o regiune contiguă a *buffer*-ului și de aceea, pot fi desenate toate într-un singur apel. Procesorul este responsabil pentru adăugarea de noi particule la finalul *buffer*-ului și pentru eliminarea celor învechite de la începutul său, dar nu are nici o informație privind partea de mijloc, activă a *buffer*-ului. Chiar și în cazul jocurilor cu mii de particule afișate simultan pe ecran, în fiecare cadru sunt create sau distruse foarte puține, astfel încât sarcina CPU-ului este foarte redusă [11].

Motorul Crimson Engine folosește sistemele de particule pentru a simula câteva efecte pirotehnice ce pot apărea în luptele aeriene: dâre de fum (atunci când un avion este ușor lovit), flăcări (atunci când motorul unui avion a fost atins) sau explozii (când avionul a fost distrus în aer sau când a avut loc o coliziune cu terenul) (**Figura 6**).

Figura 6. Sisteme de particule utilizate pentru a simula dâre de fum și flăcări

Particulele 3D sunt utilizate și în simularea condițiilor meteorologice (zăpadă, ploaie și ceață). Jocul beneficiază de un generator de condiții meteo care permite modificarea diferiților parametri (tipul precipitațiilor, intensitatea, viteza de cădere), iar particulele sunt afectate de gravitație, vânt și de condițiile de iluminare.

Modelele 3D

Pentru a reprezenta avioanele ce sunt simulate în aplicație, au fost utilizate obiecte tridimensionale create de artiști 3D independenți în programul de modelare Google SketchUp [4] și disponibile gratuit (sub licență ne-comercială) pentru *download* pe portalul 3D Warehouse [5]. Pentru a putea fi însă folosite în joc, modelele au suferit câteva modificări:

- formatul SKP nu este suportat de *framework*-ul XNA și de aceea s-a apelat la o serie de *plug-in*-uri gratuite dezvoltate de utilizatori XNA pentru a converti modelele în formatele FBX sau X acceptate de XNA;
- pentru a nu crea probleme de performanță, numărul de poligoane utilizat în crearea modelelor a fost redus, iar texturile au fost redimensionate conform specificațiilor XNA (este recomandat ca o textură să aibă dimensiuni cu valori puteri ale lui 2 pentru a se asigura compatibilitatea cu cât mai multe dispozitive grafice);
- pentru un plus de realism, s-au aplicat avioanelor modelate texturi transparente pentru zonele vitrate (carlingă sau hublouri), iar elicele au primit texturi semi-transparente care se rotesc pentru a simula mișcarea palelor.

Stocarea datelor

Pentru a putea urmări progresul jucătorului în timpul utilizării aplicației, este nevoie de o modalitate de a memora statisticile legate de acțiunile efectuate în joc. Cea mai facilă metodă este utilizarea fișierelor XML pentru a stoca parametrii profilului jucătorului (numele profilului, experiența câștigată, numărul de *Upgrade Points* obținute, victorii, înfrângeri etc.). Salvând aceste date în afara executabilului aplicației, se obține o separație între codul legat de logica jocului și datele statistice. Având datele în exterior, se pot adăuga sau actualiza fișierele ce le conțin fără a modifica vreo parte a codului aplicației. De asemenea, dacă se folosesc fișiere XML simple pentru configurarea jocului, se facilitează modificarea unor parametri de rulare fără a fi nevoie de reconstruirea proiectului XNA.

Există două metode de a folosi conținut XML în cadrul XNA: prin serializare în fișiere XML simple sau prin fișiere XML de tip XNAContent.

Principala diferență între cele două maniere de lucru constă în faptul că fișierele XnaContent XML permit detectarea erorilor la construcția proiectului XNA, spre deosebire de fișierele XML simple care pot genera erori la rularea aplicației. Desigur, se pot construi documente XSD pentru validarea fișierelor XML ce au fost serializate de aplicație sau se poate merge pe încrederea în corectitudinea creării lor.

A doua diferență este că documentele XnaContent XML vor fi convertite și compresate de către magistrala XNA Content Pipeline [9, 11] în formatul XNB la construcția proiectului, în timp ce fișierele simple XML nu vor suferi nici o optimizare.

Proiectul descris utilizează ambele metode pentru integrarea conținutului XML. Pentru stocarea informațiilor care nu se modifică o dată ce proiectul a fost construit, precum parametrii tipurilor de avioane, datele legate de misiuni, medalii sau achievement-uri, se folosesc fișiere XnaContent XML ce sunt trecute pentru validare prin Content Pipeline. Pentru stocarea profilurilor de utilizator sunt folosite fișiere XML simple, deoarece datele din aceste documente nu rămân nemodificate după construirea

proiectului, fiind actualizate de mai multe ori pe parcursul unei singure rulări a jocului.

Sistemul de administrare a stării jocului

Orice joc, indiferent de complexitatea sa, are nevoie de un sistem de administare a stării jocului (*game state management*). Aplicația memorează o stivă cu una sau mai multe instanțe ale clasei `GameScreen` din XNA. Manager-ul ecranelor de joc coordonează tranzițiile de la un ecran al jocului la altul și se ocupă de redirijarea comenzilor de la utilizator către ecranul care se află pe prima poziție a stivei.

Fiecare clasă ce definește un ecran de joc (inclusiv ecranul ce prezintă acțiunea propriu-zisă a jocului) moștenește clasa `GameScreen`. Aceasta din urmă oferă metode pentru actualizare, desenare și tratarea comenzilor de la jucător, plus logica pentru administrarea stării de tranziție. Clasa `GameScreen` nu implementează propriu-zis efectele pentru redarea tranziției, ci oferă informații privind starea procesului de tranziție. Clasa derivată va decide modul de folosire a acestor informații la momentul desenării ecranului. Astfel se facilitează substanțial implementarea diferitelor efecte vizuale necesare ecranelor în cadrul unei aceleași infrastructuri pentru tranziții.

DIRECȚII VIITOARE DE DEZVOLTARE

În versiunile următoare ale aplicației, pentru a crește gradul de satisfacție în joc pe partea de *multiplayer* se va trece de la modul *Split Screen* la modul de joc în rețea, astfel fiind posibilă reunirea mai multor jucători simultan într-o partidă, fiecare rulând aplicația pe computerul propriu. O îmbunătățire importantă ar putea consta în crearea unei aplicații server care să facă posibilă memorarea profilurilor de utilizator la nivel global. Astfel datele utilizatorilor nu vor mai fi stocate pe mașina locală, ci vor putea fi vizualizate și comparate de către toți jucătorii din comunitate (se vor putea construi clasamente pe diferite criterii: cele mai bune rezultate într-o anumită misiune, cei mai buni jucători în modul *multiplayer* etc.).

Pe partea de *single player* se vor introduce mai multe misiuni în cadrul celor două campanii și se va mări numărul tipurilor de avioane ce pot fi utilizate de către jucător. Vor putea fi introduse mai multe tipuri de obiective principale pentru misiuni, cât și obiective secundare, a căror îndeplinire să fie opțională (în schimbul finalizării unui obiectiv secundar, jucătorul va primi un plus de experiență sau un număr de *Upgrade Points*). Modul *Campaign* va beneficia și de un fir narativ bine conturat, care să mențină interesul jucătorului pe parcursul celor două campanii.

În ceea ce privește motorul jocului, se dorește introducerea unui micro-sistem de vizualizare a pagubelor produse asupra unui avion (atunci când este lovit, avionul va suferi fragmentații ale diverselor componente) pentru a crește nivelul de imersivitate al acțiunii. Pentru a obține un plus de realism, scena 3D ar putea beneficia de creșterea dimensiunilor terenului, de diversificarea tipurilor de teren și al obiectelor de decor (vegetație, tipuri de nori, clădiri, drumuri) și a efectelor speciale (trasoare pentru reprezentarea proiectilelor, așchii și fragmente pentru ilustrarea impactului gloanțelor asupra avionului, alte condiții meteo etc.) sau de creșterea numărului de avioane prezente într-o scenă simultan. Toate aceste îmbunătățiri vizuale vor trebui însă alese și dezvoltate atent pentru ca performanțele aplicației să nu aibă de suferit. O creștere a *frame rate*-ului este întotdeauna o îmbunătățire prioritară, în special în cazul simulatoarelor de zbor.

O ultimă direcție de dezvoltare posibilă are în vedere rularea aplicației pe consola Xbox 360, devenind astfel posibilă distribuirea ei prin intermediul serviciului Microsoft LIVE Marketplace. Deoarece *framework*-ul XNA a fost construit special pentru dezvoltarea de jocuri atât pentru Windows cât și pentru consola Microsoft, portarea aplicației pentru Xbox 360 este relativ facilă.

Prin implementarea acestor îmbunătățiri, proiectul Beneath a Crimson Sky ar atinge un nivel calitativ similar cu producțiile firmelor de renume din domeniu; astfel, prin tematica abordată și execuția tehnică și conceptuală, s-ar remarca și mai mult între creațiile independente dezvoltate în *framework*-ul XNA.

CONCLUZII

Popularitatea crescândă a platformei XNA era greu de anticipat la sfârșitul anului 2006, când aceasta a fost lansată. La acea dată se punea problema dacă într-adevăr jocurile pot fi scrise în *managed code* și dacă se pot obține aceleași beneficii ca în cazul dezvoltării de programe Windows standard. Membrii comunității programatorilor de jocuri erau îngrijorați în legătură cu viteza de execuție a codului *managed* şi foarte mulţi credeau că nu este posibil să creezi jocuri „reale” folosind XNA [9]. Cu trecerea timpului însă, multe persoane au început să realizeze că există multe avantaje în folosirea de *managed code* şi că temerile privind viteza de execuţie erau exagerate. Se poate spune că potențialul consolei Xbox 360 și chiar al sistemului Windows nu este experimentat la maximum decât atunci când sunt folosite pentru a rula propriile jocuri create, iar folosind *framework*-ul XNA nu mai există decât o singură barieră, cea a imaginației proprii.

Din perspectiva educațională, datorită simplității sale, XNA reprezintă o alegere excelentă pentru cei ce doresc să învețe limbajul de programare C#. Dezvoltarea de jocuri oferă o bază deosebită pentru colaborarea între elevii și studenții în informatică și cei care lucrează în domeniul artelor, muzicii și al design-ului. XNA a devenit o tehnologie atât de importantă pentru Microsoft, încât compania a decis în 2008 să creeze o categorie nouă pentru dezvoltarea jocurilor în cadrul faimosului concurs Imagine Cup [6].

Pe parcursul acestei lucrări au fost prezentate o parte dintre conceptele ce stau la baza programării jocurilor electronice, precum și modul în care ele se regăsesc aplicate într-un proiect XNA. Având o infrastructură robustă, *framework*-ul oferă perspective nelimitate în ceea ce privește dezvoltarea jocurilor de orice tip, atât pe platforme clasice (Windows sau Xbox 360), cât și pe dispozitive mai puțin folosite în *gaming* (Zune sau Windows Phone 7). Beneficiind de o comunitate de dezvoltatori independenți importantă și de sprijinul tehnic al Microsoft, XNA permite deja realizarea de jocuri complexe, atât din punct de vedere al execuției tehnice, cât și conceptual, ce pot rivaliza cu titluri consacrate, apărute

doar cu câțiva ani în urmă. Tehnologia XNA reprezintă un pas esențial în micșorarea distanței între industria jocurilor electronice și dezvoltatorii independenți.

REFERINŢE

1. App Hub. Develop for Windows Phone & Xbox 360 http://create.msdn.com/en-US/
2. Beneath a Crimson Sky. Gameplay Footage. http://www.youtube.com/watch?v=33J_3XrK3XI
3. Beneath a Crimson Sky. Game Manual. http://www.scribd.com/doc/50388141/Beneath-a-Crimson-Sky-Game-Manual
4. Google SketchUp http://sketchup.google.com/
5. Google 3D Warehouse http://sketchup.google.com/3dwarehouse/
6. Imagine Cup 2011 http://www.imaginecup.com
7. Riemer Grootjans - *XNA 3.0 Game Programming Recipes: A Problem-Solution Approach*. Apress, 2009.
8. Mary Jane Irwin – *Indie Game Developers Rise Up*. Noiembrie 2008. http://www.forbes.com/2008/11/20/games-indie-developers-tech-ebiz-cx_mji_1120indiegames.html
9. Alexandre Santos Lobão, Bruno Evangelista, José Antonio Leal de Farias şi Riemer Grootjans – *Beginning XNA 3.0 Game Programming: From Novice to Professional*. Apress, 2009.
10. NPD Group http://www.NPD.com
11. Aaron Reed – *Learning XNA 3.0*. O'Reilly Media, 2009.
12. Stephen E. Siwek – *Video Games in the 21st Century. The 2010 Report.* Entertainment Software Association, 2010.