Implementare unitară a tehnicilor de interacțiune în jocurile pe calculator

Timotei Dolean
Departamentul Calculatoare
Universitatea Tehnică din Cluj-Napoca
timotei.dolean@yahoo.co.uk

Dorian Gorgan
Departamentul Calculatoare
Universitatea Tehnică din Cluj-Napoca
dorian.gorgan@cs.utcluj.ro

REZUMAT

În ultimii ani au apărut diferite dispozitive noi de interacțiune cu calculatorul: de la cele cu *touch* până la cele gen Microsoft Kinect care interacționează cu ajutorul limbajului natural al corpului. Pe lângă aceasta, dispozitivele convenționale de genul mouse, tastatură sau *gamepad* sunt și ele folosite în aplicațiile curente. Astfel, apare problema integrării dispozitivelor noi în aplicațiile deja existente, sau crearea unora care să folosească multe din dispozitivele existente. Dezvoltatorii sunt confruntați cu nevoia de a implementa logică separată pentru fiecare din acestea, și nu de puține ori se întâmplă ca aceste implementări să fie ad-hoc sau re-implementate în fiecare nou proiect în funcție de necesitățile respective. Sistemul pe care îl propunem este unul care dorește să unifice și să abstractizeze accesul la datele primite de la diferite dispozitive prin oferirea unui nivel comun partajat de acestea.

Cuvinte cheie

Dispozitive de interacțiune, metode de interacțiune, touch, Microsoft Kinect.

Clasificare ACM

H5: Information interfaces and presentation (e.g., HCI), H.5.2: User Interfaces, H.5.m: Miscellaneous; B.4: Input/output and data communications, B.4.2: Input/output devices; D.1.5: Object-oriented programming

INTRODUCERE

Odată cu progresul tehnologic, apar constant metode noi prin care putem interacționa cu un sistem de calcul: începând de la metodele convenționale (e.g.: tastatură, mouse) și continuând cu cele mai noi (e.g.: Microsoft Kinect [1], Google Glass [2]). Dezvoltatorii de aplicații au astfel o miriadă de metode prin care pot să ofere interacțiune cu aplicațiile lor. Dar aceasta vine cu un dezavantaj: fiecare metodă are modul ei propriu de a furniza datele de ieșire care trebuie apoi procesate într-un mod aparte și fiecare metodă are un număr diferit de date de ieșire.

Nu de puține ori se întâmplă, mai ales în cazul jocurilor, să se (re) scrie anumite *wrapper-e* pentru a abstractiza unele informații de intrare, și să fie unificate pentru a le putea consuma mai apoi efectiv în joc. De aceea, acest proiect dorește să ofere o nouă metodă prin care dezvoltatorii să nu mai fie nevoiți să implementeze anumite funcționalități de bază pentru interacțiunea utilizatorilor cu aplicațiile lor, ci doar să folosească un API (*Application Programming Interface)* care să le ofere deja funcționalitatea necesară pentru diferite dispozitive de intrare, care devin din ce în ce mai complexe, puternice și mai naturale din punct de vedere al interacțiunii utilizatorului cu ele (e.g.: voce și imagine de la Kinect).

Chiar dacă există anumite implementări sau cercetări pe această temă, ele sunt nepotrivite pentru noile tehnologii care au apărut și continuă să apară în fiecare an. De aceea este nevoie de o analiză a cunoștințelor de până acum și regândite pentru noile provocări.

ALTE REALIZĂRI

Una dintre primele apariții ale conceptului de unificare sau abstractizare a metodelor de interacțiune a fost prezentă în cadrul ”Sistemului grafic al *kernel-ului”* (*Graphical Kernel System* – GKS) [3, 4]. Acesta a fost primul standard ISO de sisteme grafice pentru calculatoare de nivel de bază și, a fost foarte comun în anii `80 și `90. În cadrul acestui sistem, există o împărțire a intrărilor (*engl. input)* de la diferitele dispozitive fizice de intrare, în clase de dispozitive logice de intrare (e.g.: *locator*, *pick, choice, string*). Fiecare astfel de clasă reprezintă o metodă diferită de a trimite sistemului date de intrare. De exemplu, *string* reprezintă un șir de caractere, în timp ce *locator* reprezintă o pereche de coordonate X și Y, adică o poziție 2D (în două dimensiuni). Pe lângă acest nivel de abstractizare, mai există specificat și modelul de interacțiune, care poate fi de trei tipuri: la cerere, eșantionare și pe bază de evenimente. Dacă primul tip, pe bază de cerere, efectuează citirea datelor de intrare în funcție de necesități, ultimul tip, pe bază de evenimente, trimite datele în mod asincron, în funcție de cum utilizatorul folosește aplicația și activează anumite funcționalități. Modul de eșantionare este o combinație a celor două.

Taosong și Kaufman au propus și ei un sistem de unificare a interfeței dispozitivelor de interacțiune [5]. Sistemul propus este reprezentat printr-un protocol de comunicare între aplicații și dispozitivele de intrare, numit ”DUI (*Device Unified Interface)”*. Acest protocol implică transformarea datelor brute primite de la dispozitive cu ajutorul unor simulări și trecerea lor printr-o bază de informații a dispozitivelor. Pe baza acestor date aplicația va primi informații de la diferite dispozitive virtuale în funcție de configurarea acestora de către utilizator. Un aspect important al acestui sistem este faptul că el oferă în mod implicit posibilitatea de a personaliza modul în care se face maparea între dispozitivele fizice și logice - printr-un modul numit ”*Control Panel”* - precum și a modului în care se fac simulările. Informațiile folosite de către sistem

ca urmare a unificării, se descriu cu ajutorul unor așa numiți *tokeni* de informație. Ca și detaliu de implementare, acest sistem funcționează pe principiul *client-server*. Astfel, în momentul rulării aplicației se pornește un *server* care are sarcina de a verifica periodic ce date noi există, putând astfel să răspundă cererilor.

Pe parcursul timpului, odată ce dezvoltatorii au avut mai multe platforme de suportat, au apărut diverse *framework-uri*, gen "Object Oriented Input System" [6], care doresc să ofere acces direct, independent de platformă, la diferitele dispozitive de intrare.

După cum se poate vedea, există implementări și cercetări pentru anumite segmente din sfera aceasta a folosirii dispozitivelor de intrare, dar nu sunt unificate între ele. Proiectul de față își propune să aducă la cunoștință un sistem unificat care să ofere ambele abordări: accesul independent de platformă și unificarea metodelor de interacțiune, pentru a forma un tot unitar, într-o manieră cât mai flexibilă.

SENZORUL KINECT

Deoarece o parte importantă a acestui sistem este posibilitatea de a folosi senzorul Kinect prin nivelul de abstractizare, se vede necesară o introducere în ceea ce înseamnă și ce poate să facă acest dispozitiv. Microsoft Kinect este un senzor creat inițial ca și accesoriu pentru consola de jocuri XBOX 360, dar mai apoi oferit și pentru a fi folosit cu ajutorul unui calculator. Acesta este un dispozitiv care conține o cameră color VGA (rezoluție standard a imaginii cu o dimensiune de 640 per 480 de pixeli), o pereche de camere cu infraroșu pentru a calcula adâncimea 3D și două perechi de microfoane pentru o acuitate și precizie crescută. Senzorul este folosit pentru a detecta mișcarea unor persoane din fața lui și sunetele de jur împrejur [1]. Acesta se deosebește de majoritatea dispozitivelor de interacțiune cu calculatorul existente prin faptul că nu este folosit în mod fizic, ci doar cu ajutorul gesturilor corpului uman și a comenzilor vocale. În felul acesta, cu ajutorul acestui senzor, se pot crea aplicații interactive și inovatoare; mai ales în domeniul medicinei, unde o astfel de alternativă care nu necesită contact fizic este binevenită.

Pentru că procesul de detecție al persoanelor din zona vizibilă este unul complex și care nu este foarte precis, Microsoft a decis să ofere o anumită funcționalitate deja implementată într-un SDK (*Software Development Kit)* care a fost publicat și poate fi folosit de oricine împreună cu senzorul Kinect. Cu ajutorul acestui SDK, valabil pentru limbajele de programare C++ și C#, dezvoltatorii pot accesa într-un mod relativ simplu scheletul corpului uman (definit prin 12 puncte de interes). Totodată, ei pot folosi unealta de recunoaștere vocală, *Microsoft Speech*, pentru a recunoaște cuvintele spuse prin voce și auzite de senzor. În ultima versiune (ex. v1.7.0), s-a introdus suportul pentru detecția apăsării mâinii în aer precum și a încleștării acesteia și, construirea unui obiect 3D a persoanelor recunoscute, ca urmare a publicării unor rezultate de cercetare. Pe lângă acestea, ei au mai publicat un document cu ghiduri despre cum se programează folosind senzorul, și care sunt facilitățile și limitele tehnice ale acestuia [7].

O ALTĂ PERSPECTIVĂ

Dacă metodele de până acum au funcționat la nivel de dispozitiv, chiar dacă la un nivel mai abstract, soluția prezentată în acest articol propune o altă perspectivă, mai pragmatică. Pentru a oferi flexibilitate mai mare, e nevoie de un nivel de granularitate mai mare. Astfel, am decis să considerăm nivelul cel mai de jos al dispozitivelor, și anume: evenimentele. În momentul în care un buton pe tastatură se apasă, sau rozeta de la mouse se rotește sistemul primește o întrerupere externă prin care se anunță un eveniment extern. Aceste evenimente, prin întreruperi, stau la baza aplicațiilor actuale pentru a realiza diferite interacțiuni atât în cadrul lor cât și cu exteriorul.

De aceea, dacă această unificare o facem la nivel de eveniment, atunci ne este mai ușor să îndeplinim următoarele constrângeri:

- Compunerea unor metode sau tehnici de interacțiune complexe din unele mai simple
- Transformarea cât mai apropiată între evenimentele/acțiunile de bază și evenimentele interne ale noului sistem.

În cadrul sistemului există mai multe tipuri de evenimente de bază, numite "evenimente de intrare" (*engl.: input event)*. Acestea sunt împărțite pe categorii astfel încât să acopere o gamă cât mai variată de tipuri de intrări de la dispozitive.

Primul eveniment este cel de locație (*LocationEvent)*. Acest eveniment reprezintă o anumită poziție 3D în spațiu. El este generat luând ca sursă poziția cursorului mouse, sau în cazul senzorului Kinect, poziția mâinii.

Următorul eveniment este cel text (*TextEvent*), care este mapat pe tastele de la tastatură, sau comenzile vocale primite de la Kinect. Pe lângă aceste informații, mai ales în cadrul tastaturii, este nevoie să se transmită și anumiți modificatori (e.g.: SHIFT, ALT) pentru a se putea lua decizii corespunzătoare în aceste cazuri speciale. Pentru recunoașterea vocii, senzorul Kinect folosește un model bazat pe un dicționar de cuvinte. Cu anumite artificii se poate activa și modelul de dictare, care permite exprimarea liberă a unui text. Ca și rezolvare mai elegantă în contextul în care este necesar un text scurt (e.g.: introducerea unui nume), se definesc literele alfabetului, urmând să se pronunțe textul respectiv câte o literă pe rând. Ca și suport pentru diferite limbi, momentan doar câteva sunt recunoscute (engleză, germană, italiană, etc), dar limba română nu este printre ele.

Alt eveniment este cel care reprezintă gesturile (*GestureEvent*). Acest eveniment are ca și proprietăți: o poziție unde s-a întâmplat gestul și tipul gestului. Un exemplu de tip de gest este cel de "activare" – acesta este mapat pe click-ul de la mouse, apăsarea cu degetul pe un ecran tactil sau apăsarea cu mâna în aer. Tot aici intră și gesturile care sunt mai comune pe ecranele tactile de genul: *pinch*, *flick* sau *drag and drop*. Aceste gesturi sunt mapate foarte ușor și pe un dispozitiv Kinect. De exemplu, gestul de *drag and drop* este creat prin următoarele secvențe de evenimente gesturi: strângere mână, mutare mână (din punct de vedere al locației), eliberare strânsoare.

Ultimul eveniment, care reprezintă acțiunile nereprezentabile prin cele trei evenimente enunțate mai înainte, este cel definit de utilizator (*UserDefinedEvent*). Acesta este necesar deoarece dezvoltatorii pot avea nevoie de evenimente noi, care sunt specifice aplicației. Un alt motiv este acela că, în cazul dispozitivelor gen tastatură sau gamepad, multitudinea de butoane disponibile nu pot fi mapate dinainte pe anumite evenimente. Generarea unui astfel de eveniment se face în funcție de cum este configurată procesarea fiecărui dispozitiv de intrare. Pentru o flexibilitate maximă și pentru a se putea distinge identitatea fiecărui eveniment de acest gen, el are atașat un obiect generic de un tip de dată definit de utilizator.

Construind interacțiunile în felul acesta, unul din scopurile proiectului este atins cu succes: se pot folosi, la un moment dat mai multe dispozitive de intrare. În cadrul unui editor de nivele pentru un joc, utilizatorul poate folosi diferite dispozitive în felul următor: dispozitivul mouse pentru a selecta un element (un roboțel în cazul nostru), apoi îl rotește cu ajutorul tastelor, și în cele din urmă îl mută cu mâna folosind senzorul Kinect. Toate acestea sunt posibile fără a modifica codul în mod dinamic sau la compilare. Scenarii de genul acesta sunt favorabile în cadrul aplicațiilor care necesită diferite nivele de precizie. Un alt exemplu de genul acesta se poate întâlni la jocurile de pe console. Jucătorul poate schimba în mod aleatoriu oricând dorește modelul de interacțiune cu jocul: o jumătate de oră cu Kinect, apoi, când a obosit, folosește *gamepad-ul* să continue jocul.

ARHITECTURA SISTEMULUI

Sistemul propus (diagrama de ansamblu poate fi consultată în Figura 1) este structurat pe trei nivele: nivelul de abstractizare a platformei, nivelul de procesare și generare a evenimentelor și, agregatorul de evenimente. Primele două nivele au câte un modul pentru fiecare tip de dispozitiv de intrare suportat de sistem (e.g.: tastatură, mouse, Kinect).

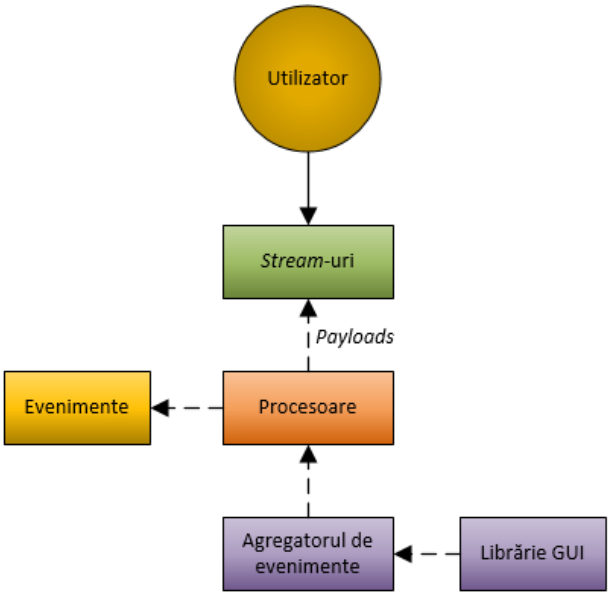

Figura 1. Diagramă de ansamblu a sistemului unitar

Abstractizarea platformei

Primul pas pentru a primi datele de intrare este de a le cere sistemului de operare. Aceasta depinde de platforma unde rulează, datele de intrare având anumite particularități în ceea ce privește accesarea lor. Nivelul acesta va asigura independența platformei, prin generarea unor așa numite încărcături (*engl.: payload)* care sunt folosite în interiorul sistemului. Modulele din acest nivel, numite ”*stream-uri”*, implementează fiecare o interfață comună, care conține un anumit tip de *payload* (e.g.: MousePayload, KeyboardPayload).

Procesarea intrărilor și generarea evenimentelor

Odată primite datele de la dispozitivele de intrare, urmează faza de procesare a acestora. Ca și la nivelul anterior, fiecare modul de la acest nivel, numit ”procesor”, implementează o interfață comună care are menirea de a asigura accesul la evenimentele generate de procesor. Modulele de aici primesc ca și intrare *payload-urile* de la nivelul anterior și, pe baza specificului fiecărui dispozitiv, generează o listă de evenimente.

În implementarea implicită a acestor procesoare, ele pot fi configurate astfel încât să genereze anumite evenimente definite de utilizator. Spre exemplu, în cadrul procesorului pentru tastatură, se poate genera un eveniment ”Mișcare dreapta”, în momentul în care se apasă tasta stânga sau „S”.

Un caz mai aparte aici este dat de situațiile în care este nevoie de a introduce text dar nu există module dedicate care fac asta (tastatură sau Kinect). Pentru a rezolva această problemă se folosește conceptul de tastatură virtuală. Aceasta este o zonă de pe ecran unde sunt afișate anumite caractere. Folosind un dispozitiv care poate deplasa cursorul, utilizatorul selectează anumite caractere și le activează, creând astfel un anumit text.

În Figura 2 se poate consulta detalierea primelor două nivele ale sistemului, pentru dispozitivele mouse, tastatură și Kinect.

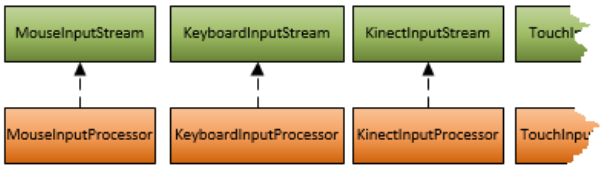

Figura 2. Primele două nivele: Stream-urile și procesoarele

Agregatorul de evenimente

Ultimul pas din cadrul sistemului este agregarea evenimentelor de la dispozitivele activate. Agregatorul are atașate o serie de procesoare, pe care la monitorizează pentru a vedea dacă sunt evenimente noi. Dezvoltatorul folosește direct acest agregator în aplicație, configurarea acestuia fiind necesară doar la început, când se atașează procesoarele suportate.

Nu doar aplicația în sine poate beneficia de acest sistem. Există anumite librării de interfețe utilizator care conțin controale predefinite (e.g.: buton, listă, casetă text). Unele

din ele nu folosesc în mod direct un anumit dispozitiv, cum ar fi mouse-ul, pentru interacțiune, ci sunt gândite în mod flexibil, și e necesar ca dezvoltatorul să injecteze anumite evenimente pentru a putea folosi controalele. În cazul acesta, se pot injecta evenimentele primite de la agregator direct în librăria cu controale, oferind mai multe posibilități de control și interacțiune cu acestea (e.g.: apăsarea unui buton cu mâna).

TESTAREA ȘI VALIDAREA SISTEMULUI

Sistemul prezentat în articol a fost implementat într-un joc 2D cu grafică 3D, de tip *platformer*. În felul acesta s-au putut face teste de performanță și validarea practică a sistemului.

Primul pas în procesul de validare a sistemului a fost verificarea evenimentelor generate în module. Pentru aceasta s-a realizat un document de analiză, prin care s-au mapat datele de intrare primite de la fiecare dispozitiv în parte, pe diferite tehnici de interacțiune. Acestea au fost mai apoi împărțite pe cele trei tipuri de evenimente definite în sistem. Au existat cazuri în care astfel de mapări nu puteau exista (e.g.: mouse-ul nu poate genera un eveniment de tip text). După ce au fost stabilite relațiile dintre datele de intrare și evenimente, s-au creat suite de teste (*engl: unit tests)* pentru a verifica implementarea.

Odată verificată generarea corectă a evenimentelor s-a continuat cu rularea jocului pe mai multe platforme. În acest sens, am folosit prima dată platforma Windows, pe arhitectura x86, adică un calculatorul uzual cu o configurație decentă (sistem de operare Windows 8 pe 64 biți, procesor Intel Pentium Dual-Core, 4GB RAM). Am testat jocul și editorul de nivele al acestuia cu ajutorul mouse-ului și tastaturii. Mai apoi am încercat folosirea Kinect-ului pentru a modifica un nivel existent. După mici ajustări ale algoritmilor de generare a evenimentelor interacțiunea cu jocul a devenit mai naturală și mai ușoară. Modulele de mouse, tastatură și Kinect fiind testate, a rămas testarea modulului *touch*. Realizarea testării s-a efectuat folosind o tabletă cu Windows RT, pe arhitectură ARM (tabletă ASUS Vivo Tab TF600, procesor NVIDIA Tegra, 2 GB RAM), deoarece în felul acesta puteam vedea cum se comportă sistemul pe o arhitectură diferită de cea de dinainte. În cadrul unei aplicații care folosește *touch* pentru interacțiunea cu utilizatorul, performanțele slabe ies mai rapid în evidență decât folosind alte dispozitive, deoarece *feedback-ul* primit este mai important și mai vizibil. Chiar și în această configurație, sistemul s-a descurcat excelent din punct de vedere al performanței. Merită menționat că modului Kinect nu a putut fi testat pe această configurație deoarece senzorul Kinect se poate conecta doar la un PC sau o consolă XBOX 360. Un ultim test a fost efectuat pe un sistem Linux (Ubuntu 12.10) și rezultatele au fost conform așteptărilor.

Ca în orice sistem software, un nivel de abstractizare aduce cu sine o anumită penalitate de performanță (procesor, memorie, etc). Pentru a măsura cât de mare este penalizarea, am realizat un scenariu de profilare (*engl: profiling)* a jocului în două situații: una în care se folosește sistemul unitar și alta în care accesează direct datele de intrare necesare. În Figura 3 se pot vedea rezultatele ca și procente ale nivelului de folosire a procesorului pe parcursul rulării, timp de aproximativ 16 secunde. Pe grafic, linia roșie de pe grafic reprezintă prima situație, iar linia albastră reprezintă cea de-a doua situație. Se observă că sistemul unitar consumă în medie sub 5% din capacitatea totală a procesorului.

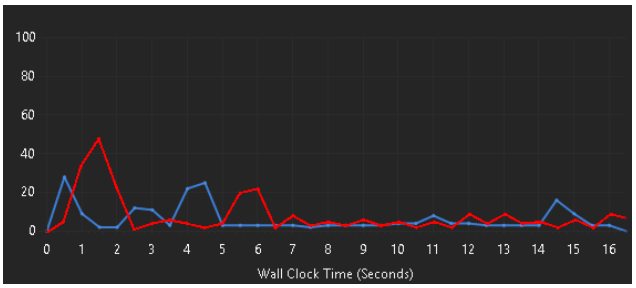

Figura 3. Rezultate sesiune de profilare

Fezabilitatea sistemului, implementat și prezentat, a fost confirmată prin câștigarea premiului întâi la secțiunea Games, din cadrul concursului *Imagine Cup 2013*, faza națională.

CONCLUZII

Marea majoritate a aplicațiilor din ziua de azi folosesc dispozitive de intrare pentru realizarea interacțiunilor utilizator. Odată cu acapararea unei piețe tot mai mari, dispozitivele mobile (*smartphone-urile)* sunt luate în vizor de către tot mai mulți dezvoltatori. De aceea, este necesar să existe o anumită bază comună tehnologică și logică care să poată fi portată pe platformele noi cu un minim de modificări. În acest sens, un sistem ca cel prezentat în acest articol este nu numai benefic, dar și necesar mai ales dacă se dorește încorporarea mai multor dispozitive total diferite – acest scenariu este cel mai folosit în cazul jocurilor.

REFERINȚE

1. W. Zeng; Microsoft Kinect Sensor and Its Effects, IEEE Multimedia, 2012.
2. Google Glass; http://www.google.com/glass/start/
3. F. R. A. Hopgood, D. A. Duce et al; Introduction to the Graphical Kernel System, Academic Press, 1986
4. D. A. Duce et al; An Approach to Hierarchical Input Devices, Computer Graphics Forum, 1990
5. Taosong He, Arie E. Kaufman, Virtual Input Devices for 3D Systems, IEEE Visualization, 1993
6. Object Oriented Input System, http://en.wikipedia.org/wiki/Object_Oriented_Input_System
7. Microsoft; Human Interface Guidelines – Kinect for Windows v1.7.0, Microsoft Corporation, 2013