L’introduzione dei videogiochi nel Corso di Laurea in Informatica

Costantini Giulia, Maggiore Giuseppe

Universitá Ca’ Foscari di Venezia

**INDICE**

1. Introduzione

2. Strumenti e tecnologie

3. Imparare riparando – Corso di programmazione del primo anno

3.1 Il corso di programmazione del primo anno

3.2 Descrizione delle esercitazioni

3.3 Esempi di esercitazioni

4. Imparare estendendo – Corso introduttivo di Ingegneria del Software

4.1 Un progetto strutturato

4.2 Tecnologie utilizzate

4.3 La reazione degli studenti

5. Prime conclusioni

6. Imparare creando – Attivitá di tesi

6.1 Descrizione del progetto

6.2 La scoperta della matematica

6.3 Carta e penna alla riscossa

6.3.1 Identificazione delle entitá

6.3.2 Caso base

6.3.3 Passo induttivo

6.3.4 Un semplice esempio

7. Risultati dell’attivitá di tesi

8. Riferimenti

**1. Introduzione**

In questo articolo vogliamo analizzare l’uso della grafica computazionale come strumento a supporto dell’insegnamento in corsi di laurea triennali di informatica. Nonostante ci siano tracce di precedenti esperimenti di questo genere [PM06], l’idea non ha ancora fatto completamente presa sulle “masse” ed é certamente soggetta a miglioramenti. Bisognerá prima di tutto capire a quali benefici tale tecnica possa portare, per decidere se vale la pena analizzarla e sperimentarla; inoltre dovremo trovare un modo per evitare i rischi collegati all’applicazione di questo approccio, che non sono pochi né poco rilevanti:

* confondere gli studenti con troppi concetti corposi
* modificare troppo il corso, vanificandone l’obiettivo originale
* richiedere troppo lavoro ai docenti e ai loro assistenti per preparare le lezioni

Un ruolo cruciale in questo esperimento viene svolto dalle tecnologie scelte: é fondamentale trovare strumenti informatici facili da usare, che permettano di ottenere velocemente buoni risultati e che si possano adattare al maggior numero possibile di contesti.

La grafica computazione é certamente un campo molto appassionante ed entusiasmante che puó contribuire a ridurre in modo decisivo la sensazione di noia che cosí spesso (purtroppo!) accompagna gli studenti durante il loro percorso universitario [GS02]. Al giorno d’oggi molti studenti si iscrivono ad un CdL in Informatica perché hanno collegato i computer al concetto di “cose belle”: animazioni 3D, film e applicazioni multimediali, interfacce grafiche coinvolgenti, e cosí via. Non possiamo non tenere conto di questo dato di fatto: una matricola di informatica é curiosa e il suo principale interesse é sapere come e perché un computer riesca a fare tutte quelle “cose belle”. Ma non dobbiamo dimenticare che gli studenti sono anche dei giovani adulti: la “vita vera” si avvicina e molti di loro cominciano a interrogarsi sul loro futuro, su cosa li aspetta lí fuori, oltre le mura accademiche, cosí sicure ma anche cosí spesse. Un giovane informatico sente quindi molto forte l’importanza di imparare tecniche e strumenti avanzatissimi dell’industria IT. Per riassumere, ció che uno studente desidera dall’universitá é studiare materie che lo aiuteranno a trovare un lavoro e nel contempo acquisire le abilitá necessarie per produrre applicazioni simili a quelle che tipicamente si vedono negli strumenti tecnologici moderni.

Dopo esserci messi nella prospettiva di uno studente passiamo dall’altro lato, ovvero mettiamoci nei panni di un docente universitario: cosa si puó guadagnare dalla tecnica che noi proponiamo? L’introduzione di grafica computazionale in un tradizionale corso di Informatica puó essere davvero quel “qualcosa in piú” che serve per trasformare studenti svogliati (che seguono le lezioni solo perché “devono”) in studenti genuinamente interessati e coinvolti dalle lezioni e dal corso. Sapendo quanto é importante per un docente parlare a una *audience* sveglia e attenta (piuttosto che a una informe massa di persone apatiche), crediamo che valga decisamente la pena sperimentare qualcosa di nuovo.

La grafica computazionale é un campo vastissimo e, nonostante una sua qualunque porzione possa essere adatta per raggiungere lo scopo che ci siamo prefissi, noi personalmente scegliamo i videogiochi. I videogiochi sono infatti l’area della grafica computazionale che é piú vicina all’esperienza quotidiana e agli interessi di uno studente (oltre, ovviamente, al fatto che é un’area con cui noi siamo familiarizzati). É molto probabile che gli studenti universitari giochino con i videogiochi, quindi l’idea di saperne di piú di questi affascinanti mondi virtuali li fará sentire abili, qualificati, capaci. E poi c’é una ulteriore ricompensa: dopo avere costruito un videogioco, lo potranno giocare!

Ora che abbiamo fissato le premesse, vediamo come é suddiviso l’articolo: nella sezione “*Strumenti e tecnologie*” vedremo l’insieme di tecnologie richieste per sperimentare nella pratica l’approccio che proponiamo. Poi analizzeremo due casi-studio presi dalla nostra esperienza personale all’universitá Ca’ Foscari: in “*Imparare riparando*” vedremo l’uso di videogiochi “rotti” come esercitazioni per il corso di programmazione del primo anno; in “*Imparare estendendo*” mostreremo come possiamo assegnare a un gruppo di studenti di un corso di ingegneria del software la produzione di un videogioco completo. Nell’esplorare i due casi studio, ci concentreremo su come riuscire a preservare la struttura originale del corso e come ottenere nello stesso tempo i maggiori benefici per studenti e docenti. In “*Prime Conclusioni*” analizzeremo i risultati ottenuti per un computo finale di vantaggi e svantaggi del nostro approccio. Poi approfondiremo ulteriormente il tema dei videogiochi al servizio della didattica: in “*Imparare creando*” illustreremo una tecnica da noi preparata per mostrare agli studenti l’utilitá della matematica tramite la produzione di videogiochi. In “*Risultati dell’attivitá di tesi*” tireremo le conclusioni a riguardo di tale approccio.

**2. Strumenti e tecnologie**

La grafica computazionale (e allo stesso modo i videogiochi) é un campo estremamente complesso, a cui sono associate tecnologie, metodi, strumenti e terminologie. Corriamo dunque un grosso rischio: rendere piú difficile il corso, chiedendo agli studenti di svolgere un doppio lavoro (imparare i contenuti del corso ma anche a produrre videogiochi!). Dobbiamo evitare a tutti i costi di aggiungere troppi concetti al corso e di distogliere l’attenzione dello studente dall’obiettivo principale del corso. Per questo é cosí importante trovare uno strumento che richieda poco sforzo per venire appreso e usato da degli studenti; inoltre, per poter applicare il nostro approccio al maggior numero possibile di corsi, vogliamo qualcosa che sia scalabile, qualcosa che ci permetta di aggiustare la difficoltá del compito a seconda delle abilitá possedute dagli studenti.

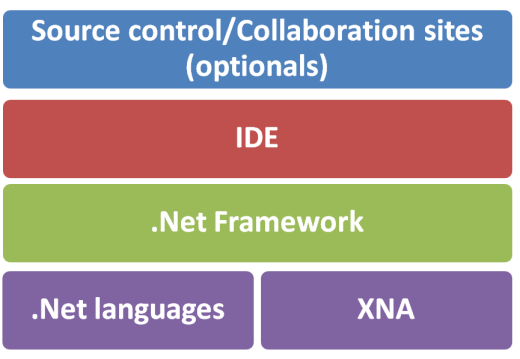
Consideriamo, come esempio, uno scenario in cui gli studenti non sanno ancora nulla di programmazione: non possiamo certo chiedere loro di sviluppare e produrre un intero videogioco, per il semplice motivo che non sono in grado di farlo. L’ideale sarebbe poter assegnare loro degli esercizi piccoli e specifici, focalizzati a svolgere un compito ben preciso, senza doversi troppo preoccupare del resto del codice che rischierebbe solamente di ingenerare confusione. In altri casi, peró, potremmo trovarci di fronte studenti con piú esperienza e con piú abilitá, ad esempio studenti prossimi alla laurea, capaci di programmare in diversi linguaggi e abituati a lavorare in gruppo. Questi studenti hanno le basi per poter (con un sistema di supporto adeguato e con gli strumenti giusti) produrre e sviluppare un intero gioco, lavorando in team di collaborazione.

Lo strumento che possiede tutte queste caratteristiche e che noi abbiamo scelto é XNA, prodotto da Microsoft. XNA é una libreria software per creare videogiochi; la sua prima versione risale al 2004, ma é rapidamente cresciuta e da poco é uscita la versione 3.1. XNA aiuta lo sviluppatore di un videogioco in tutte le aree rilevanti: i suoni, l’interazione con l’utente (il giocatore), la logica del gioco e, ovviamente, la grafica. Ció che ci ha indotti a scegliere XNA é la sua duplicitá d’uso: da un lato offre moltissimi scorciatoie per aiutare l’utente inesperto, dall’altro permette agli sviluppatori esperti di accedere a tutte le funzionalitá piú nascoste e complicate. Inoltre, non va trascurato anche il fatto che XNA permette di scrivere videogiochi che possono essere giocati sia sui normalissimi PC che sulla console Xbox360.

La scelta di XNA ci fornisce libertá nella scelta del linguaggio di programmazione da fare usare agli studenti. Infatti XNA fa parte di una struttura/impalcatura/architettura (il .NET framework) che supporta molti linguaggi. Quindi non siamo vincolati dal fatto che gli studenti abbiano studiato un particolare linguaggio nei corsi precedenti; all’interno del .NET framework troveremo sicuramente il linguaggio adatto alla situazione in cui ci troviamo.

Infine, lo strumento che tipicamente si usa per sviluppare con il .NET framework (cioé Visual Studio) ci permette di sfruttare un’altra interessante tecnologia Microsoft (Team Foundation Server), per aiutare fortemente gli studenti nel lavoro di gruppo. Questi strumenti permettono infatti la sincronizzazione del lavoro dei singoli membri del gruppo, sia che essi lo svolgano da casa o nel laboratorio dell’universitá.

La nostra architettura di tecnologie é la seguente:



**Figura 1:** *Architettura generale.*

**3. Imparare riparando – Corso di programmazione del primo anno**

In questa sezione consideriamo il primo dei nostri due casi studio.

**3.1 Il corso di programmazione del primo anno**

Il corso di cui parliamo in questa sezione é un corso introduttivo di programmazione della laurea triennale in Informatica. Il corso tratta un approccio funzionale alla programmazione, quindi in questo caso scegliamo un linguaggio funzionale dal paniere di linguaggi offerti dal .NET framework per sviluppare in XNA. Il corso si struttura in due lezioni settimanali frontali tenute dal docente e esercitazioni settimanali da risolvere durante le due settimane successive. Una volta alla settimana i tutor del corso sono presenti in laboratorio per aiutare gli studenti nello svolgere le esercitazioni. Le esercitazioni sono state divise in due parti: la tipologia tradizionale (esercizi numerici) e quella avanzata (collegata ai videogiochi). Poiché l’introduzione di videogiochi é stata un esperimento, gli studenti potevano scegliere se svolgere l’esercitazione tradizionale oppure quella avanzata.

**3.2 Descrizione delle esercitazioni**

Come abbiamo giá detto prima, non é pensabile chiedere a uno studente del primo anno di scrivere un videogioco da zero: non ne é capace. Tuttavia possiamo chedergli di svolgere un ruolo marginale nella costruzione di un videogioco: possiamo ad esempio fornire allo studente un piccolo videogioco che é stato “rotto” intenzionalmente da noi, introducendo errori nel codice o togliendo inter pezzi di codice. Le parti “rotte” del videogioco vengono ovviamente scelte in base agli argomenti svolti durante la settimana all’interno del corso. La cosa piú importante é che gli studenti devono completare una parte molto piccola (minuscola, quasi) di un videogioco ma allo stesso tempo fondamentale per farlo funzionare correttamente. Questo approccio dá loro la soddisfazione di risolvere un problema senza dovere peró possedere una vasta conoscenza di grafica computazionale o di altre aree complesse.

**3.3 Esempi di esercitazioni**

Vediamo due esempi di esercitazioni.

La primissima esercitazione del corso, svolta quando gli studenti erano quasi completamente digiuni del linguaggio di programmzazione, riguardava il costrutto *if-then-else*. L’idea alla base di questo costrutto é immediata: se (*if*) una certa condizione vale allora (*then*) si esegue una cosa, altrimenti (*else*) se ne esegue un’altra. Abbiamo preparato un insieme di esercizi tradizionali, dei quali possiamo citare un esempio: “*Dati un prezzo e uno sconto, calcola il prezzo scontato, tenendo conto del fatto che lo sconto si applica solo se il prezzo é sopra i 10000€*”. La soluzione é piuttosto banale: se (*if*) il prezzo é minore o uguale di 10000€ allora (*then*) ritorna quello stesso prezzo, altrimenti (*else*) ritorna il prezzo scontato dopo averlo calcolato. Insieme a questa esercitazione, abbiamo preparato anche la versione avanzata: un semplice videogioco in cui il giocatore (un triangolo blu al centro dello schermo) poteva sparare ai nemici (triangolini rossi sparsi nei dintorni del giocatore) e ucciderli (ovvero farli scomparire). Qual era il problema? Il gioco si comportava in maniera totalmente erratica: il giocatore sparava in continuazione in una unica direzione (mentre avrebbe dovuto sparare solo alla pressione di determinati tasti) e i nemici scomparivano nei momenti sbagliati (non corrispondenti alla collisione con proiettili). Il bello é che tutto il codice necessario per riparare questo gioco non richiedeva altro che il costrutto if-then-else (se l’utente preme un certo tasto allora spara un proiettile altrimenti non spararlo, etc). In questo modo gli studenti che provavano a svolgere l’esercitazione avanzata dovevano capire il costrutto if-then-else tanto bene quanto i loro colleghi meno impavidi (cioé quelli che sceglievano l’esercitazione tradizionale), ma alla fine del loro lavoro avevano anche ottenuto un piccolo videogioco funzionante, con la consapevolezza che il loro contributo era stato determinante.

Qui sotto possiamo vedere una immagine presa dal gioco:



**Figura 2:** *Immagine del gioco, esercitazione if-then-else.*

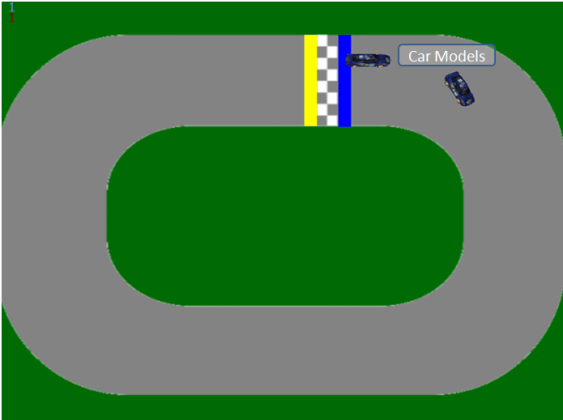
É importante notare l’aiuto dato da XNA (e il .NET framework in generale) nel rendere possibile una esercitazione di questo tipo:

1. come prima cosa, abbiamo nascosto la maggior parte del codice del gioco in un file chiuso, non visibile dagli studenti; questo era fondamentale per evitare di confonderli (ricordiamo che erano alla loro prima esercitazione universitaria in laboratorio!); le uniche parti di codice a cui gli studenti avevano accesso erano esattamente quelle che dovevano correggere con il costrutto if-then-else;
2. per fare sparare il giocatore in corrispondenza della pressione di certi tasti (ad esempio, il tasto Spazio) bisogna potere rilevare che una pressione del tasto é effettivamente avvenuta: grazie a XNA, ció é facilissimo. Possiamo infatti sapere se un certo tasto é stato premuto grazie alla seguente riga di codice: *Keyboard.IsKeyDown(Keys.Space)*, che chiede alla tastiera (*Keyboard*) se il tasto spazio (*Keys.Space*) é stato premuto (*IsKeyDown*).

Vediamo un altro esempio di esercitazione. L’ultima esercitazione del corso riguardava l’argomento “liste”. Un estratto dall’insieme degli esercizi tradizionali potrebbe essere il seguente: “*calcola il prodotto di tutti i numeri dispari di questa lista di numeri*”. Se la lista di numeri data allo studente fosse stata, ad esempio: [1; 5; 6; 3; 2; 10], allora il risultato richiesto sarebbe stato 15 = 1 \* 5 \* 3. Per l’esercitazione avanzata abbiamo invece usato un gioco di macchine (in cui la fisica delle macchine era stata parzialmente costruita dagli studenti durante le esercitazioni precedenti). Il difetto di questo gioco era che le macchine venivano rappresentate su schermo da cerchietti colorati: decisamente poco realistico! Lo scopo dell’esercitazione era dunque implementare il codice per renderizzare (visualizzare su schermo) i modelli 3D delle macchinine. Per capire il collegamento tra l’argomento “liste” e il compito assegnato per l’esercitazione avanzata, dobbiamo investigare meglio la forma di un modello 3D renderizzabile in XNA.

In XNA, un *Modello* puó rappresentare un qualunque tipo di entitá disegnabile su schermo. Un *Modello* é formato da un insieme (una lista) di *Meshes* che rappresentano singoli oggetti fisici. Ogni *Mesh* puó essere mossa indipendentemente dalle altre, cosí come puó essere disegnata su schermo separatamente dalle altre. Per fare un esempio pratico, il *Modello* di una macchina potrebbe contenere una *Mesh* per il corpo del veicolo e una per le ruote (le quattro ruote di una macchina sono esteticamente uguali, quindi serve una sola *Mesh*, che verrá poi peró disegnata 4 volte nelle giuste posizioni per rappresentare tutte le ruote). Un altro esempio potrebbe essere quello di un *Modello* che rappresenti una persona: in questo caso le *Mesh* che lo compongono potrebbero essere un busto, una testa, due braccia e due gambe. Si capisce meglio in questo secondo esempio l’importanza della possibilitá di muovere le *Mesh* indipendentemente l’una dall’altra: dobbiamo potere muovere le braccia senza necessariamente muovere anche le gambe, o viceversa. La *Mesh* é poi a sua volta formata da un insieme di *MeshParts*, che corrisponde a un sottoinsieme della *Mesh* con caratteristiche simili al suo interno. Per esempio, nel caso della *Mesh* “braccio” potremmo considerare come *MeshParts* la parte coperta dalla maglietta (che avrá appunto il colore della maglietta) e poi la parte di braccio scoperto (che avrá il colore della pelle umana). Abbiamo lavorato in modo da semplificare al massimo l’accesso e l’utilizzo di queste informazioni, in modo che gli studenti potessero (come al solito) concentrarsi sull’argomento principale dell’esercitazione (le liste) invece che su strani argomenti di grafica. Il risultato finale é stato che gli studenti, per disegnare un modello sullo schermo, dovevano scorrere la lista di *Mesh* che compongono un *Modello*; per ciascuna di queste Mesh, dovevano scorrere la lista dei suoi effetti grafici e per ciascun effetto dovevano impostare alcuni parametri necessari per il disegno.

Nell’immagine sotto, si puó vedere l’aspetto del gioco dopo avere renderizzato correttamente i modelli delle macchine:



**Figura 3:** *Immagine del gioco di macchine.*

Anche in questa esercitazione abbiamo fatto in modo di nascondere agli studenti tutto ció che non era fondamentale per la loro esercitazione.

**4. Imparare estendendo – Corso introduttivo di Ingegneria del Software**

Presentiamo ora il secondo case-study: un corso introduttivo di Ingegneria del Software, indirizzato a studenti del terzo anno della laurea triennale in Informatica. Gli studenti, ormai prossimi alla laurea, hanno tutte le capacitá di programmazione richieste dalla scrittura di un videogioco completo. Inoltre, poiché lo snodo cruciale del corso é sulle metolodogie di ingegneria del software e sul lavoro di gruppo, abbiamo deciso di dividere gli studenti in gruppi da 4 o 5 persone ciascuno.

**4.1 Un progetto strutturato**

Al fine di insegnare agli studenti come applicare metodologie di ingegneria del software, abbiamo chiesto agli studenti di costruire un semplice gioco di carte. Un videogioco di questa tipologia é tecnologicamente poco avanzato (non richiede grafica 3D di ultima generazione, per intenderci!) ma puó essere complicato in molti modi, per renderne la costruzione una sfida adatta al corso di Ingegneria del Software. L’obiettivo, quindi, non era un gioco graficamente e tecnologicamente avanzato, ma ordinato, estendibile, concepito in modo ordinato fin dall’inizio (cosa che raramente accade quando gli studenti creano le loro prime applicazioni), costruito in maniera raffinata seguendo patterns teorici noti nell’informatica e insegnati in corsi precedenti.

In particolare, l’obiettivo primario richiesto agli studenti riguardava l’estensibilitá, vista da due punti di vista diversi. Da un lato, abbiamo costruito e fornito agli studenti l’intelligenza artificiale (AI) per uno specifico gioco di carte (Machiavelli, una versione piú complicata di Scala 40/Ramino). L’intelligenza artificiale é (in parole povere) il giocatore computerizzato, che tipicamente ci si trova contro quando si gioca a carte al PC (tranne che nel solitario!). Gli studenti dovevano costruire il loro sistema in modo da usare tale intelligenza artificiale da noi fornita. Dall’altro lato, abbiamo anche chiesto loro di costruire il loro sistema in modo che rendesse possibile (con minime aggiunte al codice sorgente) giocare a diversi giochi oltre a Machiavelli. Quindi gli studenti avevano i seguenti obiettivi:

* creare una applicazione che permettesse di giocare a Machiavelli;
* usare la nostra intelligenza artificiale per il gioco Machiavelli;
* rendere possibile l’aggiunta di altri giochi di carte (Scala 40, Solitario, Poker, etc.) nel modo piú facile possibile (ovvero separando completamente le parti specifiche di un singolo gioco, cioé la logica dei turni, da quelle comuni a tutti i giochi, come il concetto di carta, di giocatore, di mazzo, etc.).

**4.2 Tecnologie utilizzate**

Abbiamo preparato delle lezioni aggiuntive (rispetto alle lezioni di un classico corso di Ingegneria del Software) per spiegare agli studenti le tecnologie che avrebbero dovuto imparare ad usare: XNA (per la costruzione del videogioco), Team Foundation Server (per lavorare piú comodamente in gruppo), SharePoint (per condivisione di documenti e comunicazioni interne).

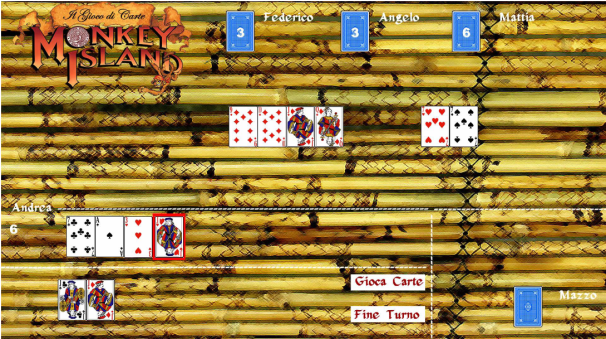
Per assicurarci che gli studenti si applicassero e prendessero confidenza con queste nuove nozioni, abbiamo assegnato loro un primo task che testasse la loro abilitá nell’utilizzo delle tecnologie menzionate sopra. L’obiettivo era costruire una semplice applicazione XNA in 2 dimensioni (niente grafica 3D per evitare complessitá inutili) in cui comparissero immagini, testo e venisse anche considerato l’input dato dall’utente. Questo task doveva esserci consegnato attraverso il Team Foundation Server. Contrariamente alle nostre aspettative, alcuni team ci hanno sorpresi introducendo parti non richieste esplicitamente dal task (e addirittura cose che non avevamo né spiegato né menzionato): sfondi che si muovevano, trasparenze, supporto per input utente non necessariamente proveniente da tastiera e mouse (ma dal controller dell’Xbox360), etc. Gli sforzi profusi in questo primo task ci hanno mostrato quale fosse il loro grado di coinvolgimento nell’attivitá da noi proposta.

Dopo questa introduzione agli strumenti, abbiamo cominciato ad affrontare argomenti piú vicini al cuore del corso e abbiamo spiegato agli studenti il funzionamento della nostra intelligenza artificiale e come costruire una applicazione XNA che la usasse. A quel punto la presentazione delle tecnologie e degli attori in gioco era terminata e gli studenti erano finalmente in grado di analizzare il problema. Senza queste conoscenze avrebbe infatti rischiato di immaginare un sistema troppo complesso da realizzare. In questa fase di analisi gli studenti dovevano definire chiaramente il problema di risolvere (sembra una banalitá ma non lo é affatto) e produrre una lista di requisiti che dovrebbero essere soddisfatti una buona soluzione al problema.

Il secondo task del corso era quindi focalizzato a capire cosa si doveva produrre; nel terzo task invece gli studenti dovevano capire come farlo. Affinché gli studenti avessero in mano tutte le conoscenze necessarie ad affrontare questo task, abbiamo presentato in maggior profonditá il funzionamento di XNA. La prima caratteristica riguarda la modalitá che Xna offre per tenere organizzato e pulito il codice, suddividendolo in moduli e sottosistemi. Poi abbiamo spiegato un modo per evitare che l’applicazione restasse completamente bloccata mentre l’intelligenza artificiale pensava alla mossa successiva da svolgere (va considerato che puó metterci anche 10-15 secondi ed é impensabile tenere l’applicazione ferma per tutto quel tempo). Abbiamo mostrato come la lettura/scrittura di dati su files possa essere fatta efficientemente e facilmente: questo é utile, per esempio, per potere memorizzare partite di carte in corso e per poterle ricaricare in un uso successivo dell’applicazione. Come ultima cosa, abbiamo spiegato come utilizzare parti di codice giá scritte da altri sviluppatori e “chiuse” in un formato specifico. Questa caratteristica é fondamentale ai fini dell’estendibilitá ad altri giochi di carte, la cui logica potrebbe essere implementata da qualcun altro, esterno al gruppo di studenti.

Il quarto e ultimo task del corso comportava di produrre del sistema progettato nei tasks precedenti e di testarne il buon funzionamento. Il voto finale é stato assegnato considerando tutte le fasi (l’analisi iniziale, la progettazione dell’applicazione, il piano di test da effettuare, il codice scritto, l’applicazione risultante). Siamo stati molto soddisfatti, poiché quasi tutti i gruppi sono riusciti a creare il gioco completo come richiesto. Inoltre alcuni gruppi hanno svolto un lavoro veramente egregio, non solo rispettando tutti i vincoli imposti ma anche raggiungendo una qualitá finale elevata.

Nelle immagini qui sotto vediamo i giochi prodotti dai gruppi migliori:

**



**Figure 4, 5, 6:** *Alcuni videogiochi prodotti.*

**4.3 La reazione degli studenti**

Alla fine del progetto, abbiamo proposto agli studenti di compilare un questionario sul corso. Il nostro obiettivo era capire quale era stata la loro reazione alll’uso dei videogiochi come case-study. I risultati del questionario sono riportati nella tabella 1 (valori medi basati sulle risposte di 44 studenti).

|  |  |
| --- | --- |
| Domanda | Punteggio  (1 = minimo, 10 = massimo) |
| Come valuti la potenza e la qualitá di XNA come strumento per lo sviluppo di videogiochi? | 7,86 |
| Quanto é stato interessante usare un videogioco come case-study? | 7,86 |
| Quanto ritieni utili le tecnologie usate all’interno del corso? | 7,07 |
| Pensi che le tue capacitá di team-work siano migliorate rispetto all’inizio del corso? | 7,76 |
| Valutazione complessiva dell’esperienza | 8,03 |

**Tabella 1:** *Risultati del questionario*

**5. Prime conclusioni**

Finora abbiamo analizzato il possibile ruolo dello sviluppo di videogiochi a supporto della didattica negli insegnamenti di Programmazione e Ingegneria del Software (di un corso di laurea triennale in Informatica). I case-study di cui abbiamo parlato hanno coinvolto un numero complessivo di circa 150 studenti.

Abbiamo illustrato quali benefici docenti e studenti possano trarre da un tale approccio: gli studenti vengono coinvolti molto di piú dalle lezioni perché i videogiochi sono entusiasmanti e gli strumenti necessari per produrli sono conoscenze preziose nell’industria IT. Il docente, d’altro canto, non deve stravolgere i contenuti del suo corso ma solamente adattare le esercitazioni (o il progetto), sapendo che con queste piccole modifiche avrá molta piú presa sugli studenti.

Come considerazione di chiusura relativa a questa prima parte, cerchiamo di capire cosa viene effettivamente richiesto a un ipotetico docente di Programmazione/Ingegneria del Software per ripetere questo esperimento. Nel primo caso (Programmazione), il docente dovrebbe produrre un semplice videogioco (di cui assegna piccole porzioni da correggere ad ogni esercitazione); nel secondo caso (Ingegneria del Software), il docente dovrebbe preparare circa 3 lezioni frontali per affrontare concetti specifici correlati alle tecnologie (principalmente a XNA) e decidere il soggetto dell’applicazione da creare (nel nostro caso era “un gioco di carte estendibile”, con tutte le sfide ingegneristiche che questo comportava). Quindi é certamente richiesto da parte del docente uno studio “a priori” di XNA ma, una volta superato questo ostacolo (nemmeno tanto grosso, a dire la veritá!), sperimentare questo approccio non é particolarmente piú faticoso rispetto all’organizzare un corso “tradizionale”, soprattutto se si tiene conto della differenza del risultato, in termini di coinvolgimento e preparazione degli studenti. Insomma, sembra che ne valga la pena!

**6. Imparare creando – Attivitá di tesi**

Alla fine del corso di Ingegneria del Software abbiamo offerto una interessante opportunitá agli studenti migliori: continuare a lavorare con XNA su videogiochi per la loro attivitá di tesi (in quanto studenti del terzo anno, la maggior parte di loro si avvicinava alla prova finale della laurea triennale). In questo modo tali studenti possono partecipare allo sviluppo di un videogioco piú complesso e piú vasto, il cui scopo é permettere di esplorare la nostra laurea in Informatica in un mondo virtuale. Questo progetto é volto a rendere familiare il giocatore (tipicamente, uno studente delle scuole superiori) con il curriculum del nostro CdL come se fosse una guerra, in cui ogni esame é una battaglia da combattere e da vincere.

**6.1 Descrizione del progetto**

Lo scopo del progetto era semplice: creare una applicazione rivolta agli studenti delle scuole superiori, per permettere loro di scoprire il Corso di Laurea in Informatica offerto dal nostro dipartimento (Universitá Ca’ Foscari). Oltre all’obiettivo primario (quello informativo) ne abbiamo imposto un altro: rendere l’applicazione divertente. Infatti, le informazioni “nude e crude” possono anche essere reperite sui siti web o recandosi alle segreterie delle universitá: noi volevamo creare qualcosa di nuovo, di diverso, che fosse uno strumento di informazione ma anche di piacevole utilizzo e quindi piú invitante. Come abbiamo giá accennato nell’introduzione, una calamita certa per attirare l’interesse dei ragazzi sono i videogiochi.

Ecco che é nata cosí l’idea di “DSI – the Game” (DSI é la sigla del nostro dipartimento): l’utente sceglie il CdL (triennale o specialistico) e quale curriculum esplorare. A questo punto, viene mostrato l’elenco degli esami da sostenere per laurearsi in tale CdL, divisi per anno. Ad ogni esame sono associate varie informazioni, tra cui il docente in carica, il numero di CFU, la modalitá di esame, l’obiettivo formativo, eventuali prerequisiti, il programma del corso, e cosí via. In questo modo lo studente puó avere una idea chiara di cosa si troverá ad affrontare se deciderá di iscriversi a quel CdL. Dove sta la parte di divertimento? É presto detto: ogni esame é associato ad un mini-gioco, e la vittoria nel mini-gioco corrisponde al superamento dell’esame corrispondente (con relativo voto in 30esimi). L’utente infatti possiede un “libretto virtuale”, in cui sono memorizzati gli esami superati e i voti ottenuti. Viene mostrata anche la “media ponderata”, cosí che lo studente delle superiori possa prendere piú confidenza con il meccanismo e il significato dei CFU, vedendone un riscontro tangibile nella media dei voti (che in alcune circostanze differisce fortemente da quella piú classica, la “media aritmetica”). Nell’utilizzare l’applicazione, lo studente ha molta libertá: puó limitarsi a leggere le informazioni offerte (senza giocare alcun mini-gioco), oppure puó provare a “laurearsi virtualmente”, affrontando ogni singolo mini-gioco proposto.

All’interno dell’applicazione abbiamo anche voluto dare corpo al nostro dipartimento, inserendo una galleria fotografica che mostrasse i luoghi in cui l’ipotetico studente si ritroverebbe a studiare se si iscrivesse: le aule, il laboratorio, il giardino, la biblioteca, etc.

Inoltre, come nel case-study di Ingegneria del Software, l’applicazione é stata costruita con in mente l’estendibilitá: al momento sono esplorabili solamente i CdL del dipartimento di Informatica, ma con uno sforzo minimo (l’inserimento dei dati relativi agli esami) é possibile aggiungere altri CdL a piacimento. L’ideale sarebbe estendere l’applicazione in modo che presenti in modo completo l’universitá Ca’ Foscari di Venezia.

**6.2 La scoperta della matematica**

Gli studenti di Informatica hanno sempre la tendenza, quando posti di fronte a una applicazione da costruire, di buttarsi a scrivere il codice, senza fermarsi a pensare con calma all’architettura generale del software che stanno costruendo, a quale é esattamente il problema che devono risolvere o a quali stratagemmi adottare per evitare errori nella soluzione che loro creano per il problema. Come abbiamo visto nel paragrafo 4, il corso di Ingegneria del Software si pone come obiettivo proprio quello di “estirpare” dalle giovani menti degli studenti questo approccio avventato ai problemi, sostituendolo con uno piú cauto, piú ragionato, piú strutturato. Tuttavia, spesso un singolo corso non basta per cambiare una mentalitá cosí radicata; é anche relativamente facile che lo studente pensi tra sé e sé: “In fondo questo é solo un corso universitario, nella vita vera poi si ragiona in modo diverso”. Per questo (e anche per dare piú corpo teorico alle tesi dei ragazzi, che altrimenti sarebbero state l’equivalente di uno stage) abbiamo deciso di presentare loro un approccio molto formale allo sviluppo dei videogiochi.

Questa tecnica si basa su particolari paradigmi di programmazione (la cosiddetta “programmazione funzionale”) ed é divisa in due parti:

1. la parte “*carta & penna*”, si fonda su concetti di logica e matematica e serve per descrivere in modo formale ció che accade nell’applicazione (il videogioco)
2. la parte “*dalla carta al codice*” é un modo facile e immediato per tradurre i concetti prodotti dalla parte (1) in codice sorgente funzionante. Ció significa che, dopo aver eseguito correttamente la parte 1 della tecnica, si puó ottenere l’applicazione in un batter d’occhio; non solo, sará anche ben funzionante.

Poiché la seconda parte é molto tecnica (e anche abbastanza diretta) non la esploreremo ulteriormente. Vediamo invece in maggior dettaglio in cosa consiste la prima parte.

**6.3 Carta e penna alla riscossa**

L’idea che sta alla base di questa parte é la seguente: pensa alla logica del tuo gioco, identifica le entitá coinvolte e come interagiscono tra di loro col passare del tempo.

Un videogioco consiste infatti di una serie di oggetti sullo schermo che compaiono, si muovono e scompaiono (non necessariamente in questo ordine). Inoltre un videogioco é composto da una collezione di schermate che si susseguono l’una dopo l’altra: il tempo non é continuo, ma discreto. Ció significa che il computer ogni tot secondi (anzi, per essere precisi, ogni tot millisecondi!) ricalcola lo stato del gioco in quel momento e lo disegna sullo schermo. Fino a che non viene calcolato lo stato successivo (e quindi ridisegnato), sullo schermo rimangono le informazioni dello stato precedente. In questo senso si puó capire meglio cosa vuol dire che un gioco é composto da una collezione di schermate: sono le schermate collegate agli istanti temporali in cui il computer ha calcolato lo stato del gioco e lo ha renderizzato (disegnato) su schermo. Questi istanti temporali vengono chiamati *frames*: quindi l’inizio del gioco (il primo stato renderizzato) sará il *frame 0*; il successivo sará il *frame 1*, poi verrá il *frame 2* e cosí via. Affinché il giocatore umano non si accorga di questa temporizzazione (ovvero affinché non si accorga che lo schermo cambia ogni tot millisecondi, ma abbia una sensazione di movimento fluido), i *frames* devono avere una frequenza compresa tra i 18 e i 60 Hz.

Per proseguire dobbiamo ora introdurre un concetto fondamentale nella matematica, il **principio di induzione**. Il principio di induzione é un enunciato sui numeri naturali che viene spesso usato per dimostrazioni matematiche. Esso asserisce che:

Se é una proprietá che vale per un certo numero (tipicamente ) e se, dato che che vale per il generico numero allora vale anche per il numero successivo (), allora vale per tutti i numeri naturali a partire da .

Una bella metafora per capire meglio quello che abbiamo appena detto é l’effetto domino: affinché le tessere da domino disposte lungo una fila cadano tutte, devono valere le seguenti condizioni:

* CASO BASE: deve cadere la prima tessera (vale per
* PASSO INDUTTIVO: ogni tessera deve essere posizionata in modo tale che cadendo provochi la caduta di quella successiva (se vale per allora vale per )

Il principio di induzione estende questa idea al caso in cui la fila sia composta da infinite tessere.

Ora che abbiamo spiegato il funzionamento alla base di un videogioco e il principio di induzione, vediamo come possiamo collegarli tra loro: se definiamo il *frame*  e come si possa passare dal *frame*  al *frame* , allora abbiamo automaticamente definito l’intera collezione di frames temporali (potenzialmente infinita).

I passi principali della nostra analisi sono dunque i seguenti:

1. *identificazione delle entitá*, cioé capire quali entitá compaiono nel nostro gioco
2. *caso base* dell’induzione (), cioé definire lo stato di tali entitá all’inizio del gioco (al *frame* )
3. *passo induttivo* (da a ), cioé capire come si modificano le entitá da un passo temporale (*frame* ) al successivo (*frame* )

**6.3.1 Identificazione delle entitá**

In questo primissimo passo di analisi vanno selezionati gli elementi di gioco. Anche se puó sembrare banale, é un passo importante perché si deve porre attenzione a non selezionare elementi inutili (complicherebbero l’analisi senza veri vantaggi) senza peró scartare elementi necessari.

Prima di tutto dobbiamo immaginare il gioco che vogliamo costruire e definire una lista di tutte le entitá che, prima o poi, compariranno all’interno del gioco. Poi dobbiamo scartare tutte le entitá non fondamentali alla logica del gioco. Per esempio, uno sfondo colorato che resta costante durante tutto il videogioco non fa parte della logica di gioco: é un elemento decorativo che viene piazzato una volta per tutte all’inizio del gioco e di cui poi non ci si preoccupa piú. Infine, dobbiamo identificare le caratteristiche principali di ciascuna entitá: ad esempio, é un oggetto singolo? Oppure é un insieme di oggetti? Ha una vita (che, per esempio, diminuisce quando gli si spara)? Ha uno schieramento (buono piuttosto che cattivo)? Se ad esempio volessimo definire in cui compaiono sciami di api che si muovono all’unisono, potremmo dover identificare l’entitá *ape* (che ha una certa vita ed é buona oppure cattiva) ma anche l’entitá *sciame* (che ha come caratteristica il numero di api che lo compongono).

Questa prima parte della nostra tecnica puó venire dunque riassunta nell’immagine seguente:

Figura 7 – Passi dell’identificazione entitá

**6.3.2 Caso base**

Nel passo precedente abbiamo identificato le entitá essenziali del nostro gioco: ora dobbiamo decidere i valori iniziali di tali entitá. Alcuni insiemi di oggetti potrebbero essere vuoti (ad esempio se all’inizio del gioco non ci sono api sullo schermo, in realtá avremo uno sciame vuoto, cioé con zero api), altre entitá potrebbero provenire da una descrizione del livello di gioco, e cosí via. Questa parte é molto semplice da descrivere in giochi in cui lo stato iniziale non deve essere calcolato, ma é predefinito.

**6.3.3 Passo induttivo**

Come abbiamo detto prima, un gioco é formato da una successione di schermate: in questo passo bisogna cercare di capire come le nostre entitá si modificano da una schermata alla successiva. Ció significa che dobbiamo descrivere una funzione che prenda come input le entitá al tempo e restituisca come output una delle tre seguenti possibilitá:

* WIN 🡪 il giocatore ha vinto la partita (il gioco é concluso)
* LOSE 🡪 il giocatore ha perso la partita (il gioco é concluso)
* Lo stato delle entitá al tempo (il gioco continua)

**6.3.4 Un semplice esempio**

Pensiamo ad esempio a un gioco semplicissimo: uno sciame di api si trova sul lato sinistro dello schermo. Il giocatore deve premere un certo tasto per spostarle gradualmente verso destra (dove si trova l’alveare). Se le api arrivano all’alveare entro un certo tempo (un minuto, per esempio) il giocatore vince la partita. Se invece alla fine del minuto le api sono ancora “in giro”, il giocatore ha perso. Le entitá rilevanti ai fini della logica di gioco sono:

* Sciame di api, caratterizzato da una posizione nello schermo
* Timer (numero, espresso in secondi)

I valori iniziali di tali entitá sono (case base):

* Sciame 🡪 posizionato nel lato sinistro dello schermo
* Timer 🡪 60 secondi

La funzione che passa da un istante temporale al successivo sará cosí costruita:

* Se il timer ha un valore minore o uguale a zero, restituisce LOSE (il giocatore ha perso)
* Se lo sciame ha una posizione corrispondente al lato destro dello schermo, restituisce WIN (il giocatore ha vinto)
* In tutti gli altri casi:
  + Se il giocatore umano ha premuto il tasto per muovere leggermente a destra le api, restituirá lo sciame di api con una posizione variata (un po’ piú a destra della precedente)
  + Se il giocatore non ha premuto il tasto, restituirá lo sciame invariato (nella stessa posizione di prima)
  + Sia che il giocatore abbia premuto o meno il tasto, il timer viene decrementato del tempo passato dal frame precedente a quello corrente (il computer ci fornisce questa informazione)

**7. Risultati dell’attivitá di tesi**

Il principale obiettivo di questa tecnica era mostrare agli studenti come la matematica e la logica possano aiutare nel risolvere problemi informatici “di tutti i giorni”.

I videogiochi che abbiamo fatto loro realizzare erano principalmente “remake” di classici arcade che tutti hanno visto almeno una volta nella vita (Pacman, Arkanoid, Puzzle Bobble, Space Invaders, Asteroids, etc) ma anche qualcosa di leggermente piú complesso come Risiko. La fase creativa nella produzione di un videogioco non va sottovalutata e in questo modo l’abbiamo saltata a pié pari, utilizzando concept giá affermati e testati. I risultati che abbiamo ottenuto sono stati molto promettenti: gli studenti hanno creato (nel lasso di tempo di un mese circa) tre piccoli giochi ciascuno, senza incontrare particolari problemi. É un risultato considerevole, visto che durante quel mese gli studenti avevano anche corsi da seguire e studiare).

Gli studenti hanno davvero apprezzato l’idea di sfruttare teorie matematiche finalizzate stavolta a produrre qualcosa di divertente ed esteticamente valido, invece del solito risultato numerico di un esercizio svolto in classe: le parti teoriche del loro corso di studi prendevano finalmente vita. Un ragazzo, una volta afferrato il cuore della tecnica, ha esclamato: “Ecco a cosa serviva tutta quella matematica!”.

Alcune immagini dei videogiochi prodotti dai ragazzi sono i seguenti:

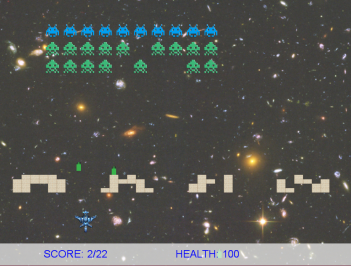


Figure 8, 9, 10 – Arkanoid, Risiko e Space Invaders

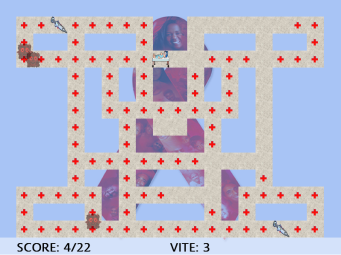


Figure 11, 12, 13 – PacMan, SuperPang e PacXon

****

Figure 14, 15 – Project Solar System e Puzzle Bobble

Come potete vedere nelle immagini dalla 8 alla 15, i videogiochi prodotti sono apprezzabili esteticamente; inoltre il codice sorgente sottostante (le “fondamenta del gioco”) é ben scritto (la logica é estendibile e tutto funziona perfettamente).

Ricordiamo peró che l’obiettivo finale era ottenere un prodotto che presentasse il nostro dipartimento all’esterno: quindi, oltre a fare sviluppare i minigiochi (corrispondenti ciascuno a un esame virtuale), abbiamo anche fatto creare a un ragazzo l’impalcatura esterna dell’applicazione, che gestisse tutti questi miniogiochi organicamente, che presentasse tutte le informazioni associate, che memorizzasse gli esami virtuali svolti dall’utente e cosí via. L’applicazione finale é venuta particolarmente bene; alcune immagini di tale applicazione sono le seguenti:

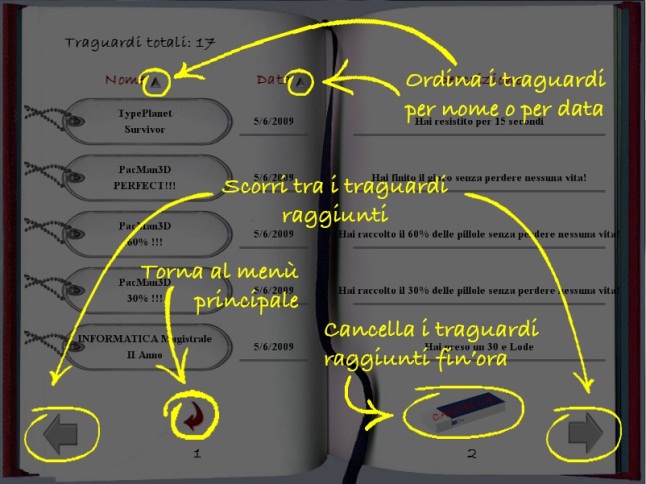


Figure 16, 17 – Il menu iniziale dell’applicazione e la guida delle schermate



Figure 18, 19 – Gli autori del gioco e la photogallery del dipartimento di Informatica

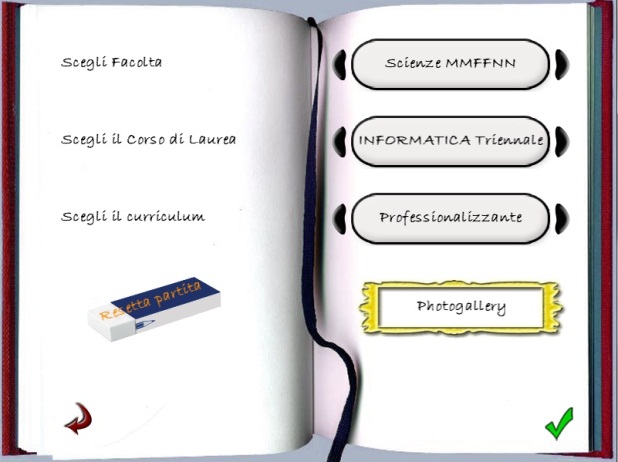
****

Figure 20, 21 – La scelta del CdL da esplorare e la schermata di vittoria in un gioco-esame

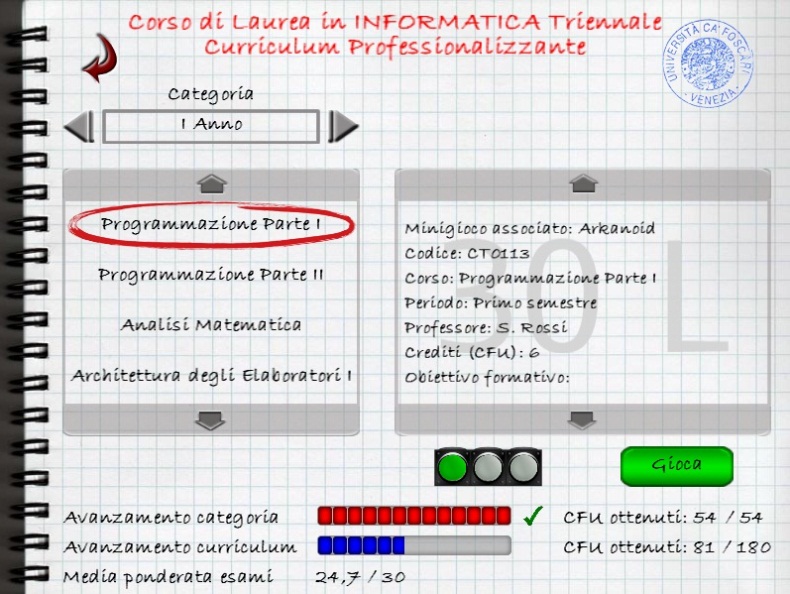
****

Figura 22 – La schermata di informazioni: a sinistra si vede l’elenco degli esami divisi per categoria (I anno, II anno, III anno, etc), sulla destra le informazioni dell’esame selezionato (docente, CFU, programma, etc), in basso l’avanzamento del giocatore e la media ponderata raggiunta

**Riferimenti**

[CC01] Curriculum Guidelines for Undergraduate Degree Programs in Computer Science,

<http://www.acm.org/education/education/education/curric_vols/cc2001.pdf>

[GS02] Guzdial M., Soloway E.: Teaching the Nintendo generation to program. *Commun. ACM, 45(4)* (2002)

[MG07] Maragos K., Grigoriadou M.: Designing an Educational Online Multiplayer Game for learning Programming, *Proceedings of IEEII 2007, Thessaloniki, Greece* 322-331 (2007)

[Mos97] Moser R.: A fantasy adventure game as a learning environment: Why learning to program is so difficult and what can be done about it. *Proc. Of ACM ITiCSE 97, Uppsala, Sweden* 114-116 (1997)

[PM06] Pedroni M., Meyer B.: The Inverted Curriculum in Practice. *Proceedings of SIGCSE 2006*, *ACM, Houston, Texas* (March 2006)

[Pre03] Prensky M.: Digital Game-Based Learning. *Computers in Entertainment, vol. 1(1)* (2003)

[Whi84] White B.Y.: Designing computer games to help physics students understand Newton’s laws of motion*. Cognition and instruction, 1*, 1, 69-108 (1984)

[XNA] XNA Official Site, <http://creators.xna.com/en-US>

Achten, P. 2008. Teaching functional programming with soccer-fun. In *Proceedings of the 2008 international Workshop on Functional and Declarative Programming in Education* (Victoria, BC, Canada, September 21 - 21, 2008). FDPE '08. ACM, New York, NY, 61-72. DOI= <http://doi.acm.org/10.1145/1411260.1411270>

Chakravarty, M. M. and Keller, G. 2004. The risks and benefits of teaching purely functional programming in first year. *J. Funct. Program.* 14, 1 (Jan. 2004), 113-123. DOI= <http://dx.doi.org/10.1017/S0956796803004805>

Costantini, G., Maggiore, G. and Cortesi, A. 2009. Learning by fixing and extending games. In *Proceedings of Eurographics* 2009 (Muenchen, Germany, March 30 – 3 April, 2009)

Harrison, R. 1993. The use of functional languages in teaching computer science. In *Journal of Functional Programming* (1993), 3:67-75 Cambridge University Press.

Hughes, J. 2008. Experiences from teaching functional programming at Chalmers. *SIGPLAN Not.* 43, 11 (Nov. 2008), 77-80. DOI= <http://doi.acm.org/10.1145/1480828.1480845>

Joy, M., Matthews, S. 1994. Some experiences in teaching functional programming. In *International Journal of Mathematical Education in Science and Technology*, 25 (2), 165-172.

Thompson, S. J. 1997. Where Do I Begin? A Problem Solving Approach in teaching Functional Programming. In *Proceedings of the9th international Symposium on Programming Languages: Implementations, Logics, and Programs: including A Special Trach on Declarative Programming Languages in Education* (September 03 - 05, 1997). H. Glaser, P. H. Hartel, and H. Kuchen, Eds. Lecture Notes In Computer Science, vol. 1292. Springer-Verlag, London, 323-334.