

Progetto di Sistemi Operativi - A.A. 2011/2012

Giulio Pellitta

15 maggio 2012

1 Funzionalità del programma

Si deve implementare un'architettura client-server per un RPG testuale multiplayer (“real-time”, non “a turni” o altro). Il progetto comprende la realizzazione sia del server che del client. Il client fornisce l'interfaccia per consentire all'utente di interagire col server. Al primo avvio il client deve concordare con il server i parametri di funzionamento (nome-utente, password). L'interazione successiva avviene attraverso comandi immessi dall'utente. Il server gestisce la creazione di nuove utenze, l'interazione dei personaggi giocatore (nel seguito “PG”) tra di loro e con entità gestite dal server stesso (nel seguito “mostri”) e in generale con il mondo virtuale.

I PG uccidendo mostri ottengono denaro ed esperienza, informazioni di cui il server deve tenere traccia (memorizzando i dati dei PG su disco). Un PG non ha bisogno di bere, mangiare o dormire; non invecchia (né muore di vecchiaia). Il server è concorrente, quindi può gestire più PG contemporaneamente. Il mondo virtuale in cui si svolge l'azione è diviso in stanze, in genere due entità interagiranno tra loro solo se si trovano nella stessa stanza (es. un PG non può attaccare un mostro in una stanza diversa dalla sua). Non è previsto il combattimento tra PG (il server non è “PvP”), l'unica interazione possibile tra di loro è la chat (comando “talk(who,msg)”). Quando un utente si collega al server, il suo PG parte da una stanza predefinita e poi si sposta per il mondo cercando altri PG con cui attaccare i mostri presenti. Lo stato delle stanze (mostri ed oggetti presenti) viene resettato periodicamente: ad intervalli di tempo predefiniti (es. ogni 10 min) i mostri uccisi sono ricreati (in seguito “respawn”), tranne nel caso ci sia un combattimento in corso (bisogna aspettare almeno 5 min dalla fine del combattimento prima del reset, per dar tempo ai PG di prendere gli oggetti). Invece, quando invece un PG viene ucciso, ovvero se i suoi punti ferita scendono a zero, viene spostato in una stanza speciale denominata “tempio” e viene riportato in vita, ma subisce una penalità sui punti esperienza accumulati (l'esperienza può scendere al massimo a zero, non è prevista la perdita di livelli).

All'avvio il server deve leggere un file di configurazione dove vengono fissati i parametri di funzionamento e caricare le informazioni sull'ambiente virtuale. Chi opera il server (in seguito Game Master o GM) può mandare un messaggio in chat ad un PG o a tutti (comando “talk(who,msg)” o “talkall(msg)”) e visualizzare l'elenco degli utenti connessi (comando “who()”), le informazioni relative ad un PG (comando “show(who)”). Il GM può manifestarsi nel mondo virtuale (comandi “appear(room)” e “vanish()”) o osservare cosa succede senza essere presente (comando “peek(room)”). Il GM non può ferire le altre entità o essere ferito; invece può bandire un PG, cioè disconnettere un utente ed impedire che si ricollegli per un certo intervallo di tempo (comando “ban(who,time=1)”), per default un'ora. Infine, il GM ha un comando “quit()” per interrompere il servizio. Le funzionalità vanno implementate scegliendo quale sintassi adottare (in particolare, non è detto che ci sia corrispondenza 1:1 tra funzionalità/comandi e metodi).

2 Entità

Lo stato di un'entità (PG o mostro) è caratterizzato principalmente da due valori, i punti ferita (PF/HP) e i punti magia (PM/MP), che rappresentano la sua salute e (solo per maghi e chierici) la sua concentrazione. Quando un'entità viene ferita (resp. usa la magia) i suoi punti ferita (resp. magia) calano, ma vengono recuperati nel tempo. Maghi e chierici recuperano contemporaneamente entrambi, ma a ritmo dimezzato rispetto alle entità che non usano la magia.

Le unità sono caratterizzate anche da forza, attacco e difesa. Il danno di ogni colpo è $XD3+Y$, dove X è il valore della forza, $XD3$ indica la somma di X variabili casuali scelte uniformemente in $\{1,2,3\}$ ed Y il bonus dell'eventuale arma. La difesa è data da un valore base dipendente dalla classe (vedi sotto) più il bonus dell'armatura (si sottrae ai danni subiti per dare i danni effettivi, s'intende che il danno è non-negativo).

Ogni unità possiede una certa quantità di monete d'oro (nel seguito "MO") ed un inventario (gli oggetti che possiede: un rubino, una spada, ecc...). Se un oggetto è un'arma l'unità può equipaggiarlo. Un'arma aumenta il danno inflitto ad ogni colpo di una certa quantità, mentre uno scudo o simili assorbono danni (tutti o in parte, fino ad un certo limite) ad ogni colpo.

2.1 PG

Un PG corrisponde ad un personaggio del mondo virtuale controllato da un utente. Se un giocatore vuole avere diversi PG dovrà avere diverse utenze. Quando viene creato un PG si trova al 1° livello di una classe a scelta tra guerriero, chierico, mago. I chierici hanno una magia curativa (comando "heal(who)"), i maghi una offensiva (comando "fireball(who)"), mentre i guerrieri nessuna. La magia curativa può ripristinare un numero di PF pari al livello del chierico per 3, mentre quella offensiva causa un danno pari a 5 volte il livello del mago. Usare la magia curativa costa 10 PM, quella offensiva 9 PM. Quando si usa un comando per lanciare una magia, s'intende che l'attacco fisico/diretto successivo viene saltato ed al suo posto ha effetto la magia (è anche possibile iniziare un combattimento usando la magia offensiva).

Per passare dal livello k al livello $k+1$ un PG deve ottenere $2^k \cdot 1000$ punti esperienza, si può raggiungere al massimo il 20° livello. Quando un PG aumenta di livello i suoi PF massimi vengono aumentati di 5 se è un guerriero e di 3 se è un chierico e di 2 se è un mago; i chierici acquisiscono altri 3 MP e i maghi 4. Al 1° livello i guerrieri partono con 12 PF, i chierici 8 PF e i maghi 6 PF; i chierici partono con 10 MP, i maghi 15 MP. I guerrieri hanno forza pari a 2 volte il livello, i chierici 1.75 (arrotondato per difetto) e i maghi 0.75 (arrotondato per eccesso). La difesa è 4 per i guerrieri, 2 per i chierici e 1 per i maghi.

Per equipaggiare un'arma, uno scudo, ecc... si usa il comando `equip(item)`, dove s'intende che `item` deve essere presente nell'inventario. Per togliere un oggetto equipaggiato si usa un analogo comando `unequip(item)`. Si usa `look(who)` per osservare l'equipaggiamento indossato da un'entità (nella stessa stanza) e `inventory()` per vedere l'equipaggiamento indossato e il denaro posseduto. Per semplicità si assume che un PG possa trasportare una quantità illimitata di oro e oggetti; tuttavia un PG può indossare una sola armatura e utilizzare un solo scudo e una sola arma (per semplicità assumiamo che tutte le armi siano a una mano, niente archi o spadoni a due mani — l'alternativa sarebbe che anche questi sono utilizzati come armi a una mano).

I PG possono cooperare nell'uccisione dei mostri, ma senza costituire dei veri e propri gruppi organizzati (il "gruppo" dura il tempo di un combattimento). Si può attaccare un mostro già in combattimento con altri PG: se il mostro viene ucciso i suoi punti esperienza sono divisi equamente tra tutti i PG che hanno attaccato il mostro e che non sono morti in combattimento.

2.2 Mostri

Definire una o più tipologie di mostri (es. Orso/Lupo; Guardia/Bandito; Scheletro/Zombi; Arcimago/Drago; ...) contro cui i PG possano confrontarsi (stats ed equipaggiamento a piacere, non sono particolari rilevanti). Per ognuno definire stati appropriati ed eventuali oggetti posseduti (oro, armi, ecc...). Il progetto deve prevedere almeno una stanza con almeno due mostri di cui almeno due dello stesso tipo (e che abbiamo sia oro che equipaggiamento).

2.2.1 Combattimento

Le entità che con cui si confrontano i PG si comportano in modo elementare: se attaccati si difendono (per semplicità non è necessario considerare che un mostro attacchi per primo). Inoltre i mostri non si muovono di stanza in stanza, ma stanno fermi dove sono. Per iniziare il combattimento basta che l'utente usi il comando `attack(who)`: lo scambio di colpi continua fin quando uno tra il mostro e il PG non muore. Se il mostro attaccato dal PG non è l'unico della stanza, il comando `attack` fa iniziare il combattimento automaticamente anche per gli altri mostri dello stesso tipo lì presenti (es. se in una stanza ci sono due mostri "guardia", attaccando il primo anche il secondo reagisce). Se viene ucciso il PG che ha iniziato l'attacco, i mostri passano poi ad eventuali altri PG dello stesso "gruppo". Il combattimento quindi è tutti contro tutti, fino alla morte. Durante il combattimento un PG può decidere di cambiare mostro da attaccare, sempre usando il comando `attack(who)`.

Durante il combattimento, ogni PG della stanza deve vedere cosa succede. Esempio:

```
Una guardia attacca Slick.  
Slick attacca una guardia.  
Una guardia attacca Mithe.  
Mithe attacca una guardia.  
...
```

Un PG impegnato nella mischia non può lasciare la stanza finché è in corso il combattimento, ovvero finché i mostri contro cui combatte il "gruppo" non sono tutti morti.

2.2.2 Bottino/Esperienza

Quando un mostro muore lascia a terra l'oro e gli oggetti che possiede (equipaggiati o meno). Il PG che lo ha ucciso guadagna i punti esperienza corrispondenti.

Se il PG che uccide il mostro è in "gruppo", l'oro del mostro viene automaticamente diviso tra i PG del gruppo impegnati nel combattimento (contro un qualsiasi mostro dello stesso tipo nella stanza) e presenti nella stanza (gli oggetti invece vanno comunque a terra, i PG scelgono manualmente come spartirseli). Attenzione a gestire cosa succede se più mostri vengono uccisi nello stesso momento.

3 Mondo

Il mondo in cui si svolge l'azione è costituito di una serie di "stanze" (si tratta di un qualcosa di finito e discreto che si può pensare come un grafo orientato, vedi sezione §3.1). I PG possono spostarsi da una stanza all'altra indicando la direzione in cui spostarsi (per semplicità bastano `nswe`, a scacchiera insomma — non è richiesto avere mosse in diagonale). L'azione che si svolge in una stanza è indipendente da ciò che avviene altrove. L'unica eccezione a questa regola è la possibilità di mandare un messaggio a tutti i

giocatori contemporaneamente (comando "talkall(msg)"), azione che può essere permessa al più ogni 10 minuti ad ogni PG (per evitare spamming).

3.1 Stanze

Una stanza è una porzione ben definita dell'ambiente virtuale. La stanza va pensata come una sorta di ambiente zero-dimensionale (un punto, insomma): non ci si muove all'interno della stanza ma solo da stanza a stanza, l'interazione con oggetti o entità nella stanza (es. combattimento o svuotare contenitori) quindi avviene senza bisogno di avvicinarsi/calcolare distanze/ecc... S'intende che in combattimento un colpo inferto ad un'entità colpisce solo quella particolare entità e non altre (in particolare un PG non attacca più mostri alla volta e non colpisce anche se stesso o eventuali altri PG presenti).

Quando un PG entra in una stanza l'utente deve poter leggere dal client una breve descrizione della stanza e di eventuali oggetti o contenitori presenti. Qualcosa come questa

```
+-----+
|Stanza del tesoro|
+-----+
```

Forzieri stracolmi d'oro sono disposti ad ogni lato di questa stanza. In aggiunta il pavimento e' ricolmo di gemme e gioielli di ogni genere.

```
[Forziere] 100000 MO
[Forziere] 100000 MO
[Forziere] 100000 MO
[Forziere] 100000 MO
[Forziere] 100000 MO
[A terra] 20 diamanti, 10 rubini, 5 anelli, 2 collane
```

oppure questa

```
+-----+
|Tunnel|
+-----+
```

Il tunnel che stai percorrendo continua ancora, non si vede la fine. L'unica scelta e' continuare ad avanzare o tornare indietro.

o ancora questa

```
+-----+
|Foresta|
+-----+
```

Intorno a te non ci sono che alberi. La borscaglia e' molto fitta e non e' semplice orientarsi. La direzione del sole e' il tuo unico punto di riferimento.

Naturalmente per il progetto sono sufficienti descrizioni molto semplici e stringate (non c'è bisogno di essere scrittori o poeti).

Il nome della stanza non deve essere univoco. Ad esempio, se il PG attraversa molte stanze tutte uguali denominate “Tunnel” o “Foresta” si dà così l’impressione che stia percorrendo una lunga distanza. Da notare però che una stanza può essere molto piccola (una stanza di un edificio, un piccolo corridoio, un ingresso) o molto grande (una piazza, una zona boscosa, un’area desertica) quindi la distanza percorsa è relativa.

Non ci interessa modellare lo scorrere del tempo, quindi si può pensare che sia sempre giorno. L’utente può leggere sempre la descrizione della stanza, anche se entra in un buio tunnel (per semplicità non prevediamo l’uso di torce/lanterne e simili meccanismi).

Dopo la descrizione della stanza, l’utente deve leggere l’elenco degli eventuali PG già presenti in quella stanza seguito da quello dei mostri. Ad esempio:

```
+-----+
|Armeria|
+-----+
```

Lance e spade dell’esercito che protegge la cittadella sono custodite qui. La stanza e’ costantemente sorvegliata da guardie ben armate.

```
[Porta-lance] 5 alabarde, 10 lance
[Cassa] 30 spade, 25 archi
```

```
Slick (PG)
Mithe (PG)
Una guardia
Una guardia
```

I PG possono interagire con gli oggetti presenti nella stanza con i comandi “get(container,what)” e “getall(container)” (assumiamo che un contenitore non abbia lucchetti o simili e che sia sempre aperto). Un contenitore può essere una vera e propria cassa/cesta/...oppure può essere qualcosa di più particolare (ad esempio, è sempre presente in ogni stanza il contenitore “terra”). Se una stanza contiene più contenitori con lo stesso nome, ci si riferisce agli altri (ad esempio) aggiungendo un numero alla fine (es. get(forziere2,anello)). Analogo sistema per i mostri, es. guardia e guardia2 (s’intende che se si uccide la prima guardia, l’unica rimasta si può indicare semplicemente “guardia” — i numeri vanno a scalare).

Se l’utente prova a muovere il suo PG in una direzione non consentita (c’è un muro o altro ostacolo nella stanza) deve ricevere un messaggio di errore appropriato (l’utente comunque deve vedere l’elenco delle uscite nel prompt, vedi sezione §4).

Per gli scopi del progetto assumiamo che si può andare dalla stanza A alla stanza B se e solo se si può andare dalla stanza B alla stanza A. Non è necessario considerare situazioni particolari per cui un PG entra in una stanza e poi non può tornare indietro (es. stanze trappola, precipizi, ecc...).

3.2 Stanze speciali

La piazza principale è la stanza di inizio/fine sessione; c’è un comando logout disponibile solo in quella stanza. In tal modo l’esecuzione distribuita è semplificata: le informazioni sul PG dell’utente vanno memorizzate solo nella macchina principale e non vanno duplicate o trasmesse a/da altre macchine.

Il tempio è la stanza di risurrezione. Quando muore il PG (PF a 0 o meno) si ritrova trasportato qui automaticamente. Il PG non perde oro/equipaggiamento quando muore e si ritrova al tempio con 1 PF e i PM che aveva prima (ma perde punti esperienza pari al 10% di quelli necessari per passare di livello).

C'è una stanza emporio con comandi buy (scorta illimitata di spade, lance, scudi, . . .) e sell (gli oggetti venduti non entrano a far parte dell'inventario del negozio); il negozio ha fondi illimitati per comprare da PG. Non c'è bisogno che i PG interagiscano con un entità negoziante (simili entità sono note in gergo come PNG, personaggi non giocatore), basta che siano disponibili le funzionalità buy/sell.

4 Client

Appena avviato il client legge un file di configurazione che determina alcuni parametri di funzionamento:

- indirizzo e porta del server a cui connettersi;
- nome utente e password per l'accesso al server.

Dopo l'inizializzazione il client si connette al server usando il nome utente e la password letti dal file di configurazione. A questo punto l'utente può usare il client per eseguire le varie operazioni che comandano il suo PG. Il client deve mostrare un prompt con tutte le informazioni rilevanti (almeno livello PF/PM, uscite della stanza, percentuale punti esperienza acquisiti per passare di livello, oro). Il prompt va ristampato a video in combattimento ogni volta che il PG subisce un attacco (anche da 0 danni). Il client esegue tutte le operazioni (tranne la stampa dell'help) interagendo col server, poichè tutte le informazioni di stato (stats dei PG, composizione delle stanze, . . .) risiedono sul server. Il client deve anche comunicare al server la sua intenzione di terminare. Le interazioni tra utente e client avvengono tramite input/output da tastiera (la scelta delle modalità esatte è lasciata agli studenti). L'interfaccia del client può essere semplicemente di testo, non si deve implementare un'interfaccia grafica. Invece è ammesso usare ASCII ART o stampe colorate a schermo (NB: la loro implementazione di per sé non dà diritto ad una valutazione più alta). Prevedere anche un comando map() che, interagendo col server, restituisca uno schema delle stanze principali adiacenti a quella iniziale. Si può tenere la mappa come ASCII ART sul server e poi comunicarla al client su richiesta, ad esempio:

```
+-----+
|Castello|
+-----+
  |
  |
+-----+ +-----+
|Locanda|---|Emporio|
+-----+ +-----+
  |
  |
+-----+
|Porta sud|
+-----+
```

Nell'implementazione del client vanno gestite le possibili situazioni di errore, quali terminazione del server, comandi errati nel file di configurazione o durante l'interazione con l'utente e altro. Eventuali errori

non devono, ove possibile, compromettere il funzionamento globale del programma, e devono comunque essere diagnosticati all'utente da opportuni messaggi d'errore chiaramente interpretabili e specifici (Errore è troppo generale...). Data la natura "ludica" del progetto, l'utente dovrà leggere a video solo i messaggi di errori gravi, tali da compromettere il funzionamento corretto dell'applicazione; altri messaggi di errori vanno preferibilmente scritti in un file di registro (un file .log) memorizzato sul disco del PC dell'utente. Non devono essere considerate nel progetto problematiche legate alla sicurezza (ad es., crittografia per le password).

5 Server

Il server è configurabile tramite un file che specifica (almeno) la porta di ascolto del server. "quit" causa la terminazione del servizio. Per tenere traccia delle informazioni sui PG non è richiesto fare uso di database o simili, ma basta usare un semplice file di testo (ad esempio in formato csv, xml o simili) anche se è consentito l'uso di file binari per scrivere gli oggetti direttamente su file. I dati sulle descrizioni delle stanze invece devono essere scritti (almeno la maggior parte delle informazioni) in file di testo, in modo che siano modificabili dai gestori del server senza bisogno di comandi appositi (l'idea è permettere ai GM di apportare modifiche alle stanze modificando direttamente un file testuale in un certo formato). Il server riceve richieste di connessione dai client. I client sono identificati da login e password (si assume che i login siano tutti diversi tra loro). Dopo il login il server gestisce le richieste del client, come descritto nella parte relativa al client. In caso di terminazione (richiesta tramite il comando "quit") le connessioni in corso vengono interrotte. I client devono ricevere un messaggio di disconnessione opportuno (lo stato del PG va salvato come se si fosse recato nella zona iniziale ed avesse usato il comando logout). Avvisare inoltre gli utenti almeno un minuto prima della terminazione effettiva e relativa disconnessione.

6 Esecuzione distribuita

Come già sottolineato l'azione che si svolge in una stanza è indipendente da ciò che succede nelle altre stanze. Questo significa che ogni stanza può essere gestita da una macchina separata. Gli unici due casi in cui un'azione può riguardare più stanze sono se un PG o un GM usa il comando talkall oppure se un PG si sposta da una stanza ad un'altra.

- Nel primo caso, gestire la cosa nel modo più semplice possibile. Se due utenti usano il comando talkall da due stanze diverse, potenzialmente su macchine diverse, non è importante se l'effetto delle due invocazioni del comando non arriva in ordine ovunque (a meno che le due invocazioni non partano dalla stessa stanza).
- Nel secondo caso bisogna che la macchina che gestisce la prima stanza metta in comunicazione il client dell'utente con la macchina che gestisce la seconda stanza. Durante lo spostamento il PG dell'utente deve continuare a risultare nell'elenco degli utenti connessi (quello del comando who) e se in contemporanea con lo spostamento qualcuno usa il comando talkall l'utente non deve perdersi il messaggio.

Ove necessario andranno quindi utilizzati opportuni meccanismi di sincronizzazione.

All'attivazione del server principale, questo dovrà attivare le stanze sulle altre macchine. Viceversa, quando il server deve essere terminato deve prima disattivare l'esecuzione sulle altre macchine.

Prevedere due server “slave” oltre al server principale. La suddivisione delle stanze tra i vari server può essere fatta in modo fissato, gli unici vincoli sono che la stanza iniziale sia sul server principale e che sia possibile muoversi tra una stanza su di un server slave ed una stanza sul server master e tra stanze di server slave diversi (in particolare, ogni server deve gestire almeno una stanza).

7 Gruppi

I gruppi dovranno essere composti da minimo 3 - massimo 5 persone. Una volta che un gruppo si è formato, i singoli componenti devono necessariamente iscriversi all’appello di esame all’indirizzo: <http://esami.int.nws.cs.unibo.it/cgi-bin/Main.php> (specificando il gruppo di appartenenza). Da notare che l’appello è unico, indipendentemente dalla data di consegna del progetto e dalla data effettiva della discussione. Coloro che non daranno comunicazione della loro condizione di gruppo entro il 15 MAGGIO NON POTRANNO FARE IL PROGETTO. Pertanto chi non riuscisse a trovare un gruppo lo deve necessariamente COMUNICARE PER EMAIL (pellitta@cs.unibo.it) specificando i loro dati (nome-cognome-matricola-email) ed eventuali preferenze legate al luogo di lavoro e ai tempi di lavoro, per aiutarmi a creare gruppi uniformi. Provvederò io a raggrupparli. I gruppi così creati terranno conto delle preferenze espresse solo ove possibile, e non sarà possibile modificarli. Anche qualora i gruppi così creati includano persone residenti in posti diversi, tali persone si devono organizzare per gestire il lavoro di gruppo.

8 Modalità di consegna

Si devono consegnare due file: un archivio `.tar.gz` o `.zip` che contenga tutti i file relativi al progetto e un file `README` che specifichi passo per passo come, a partire dall’archivio, si possano eseguire tutte le operazioni necessarie per compilare e eseguire sia il server che il client (meglio ancora sarebbe allegare anche uno script per la compilazione ed uno per l’esecuzione, che vi conviene scrivere anche per voi stessi per risparmiare tempo in fase di sviluppo automatizzando queste operazioni). L’archivio deve contenere le seguenti directory: `src`, contenente i sorgenti del progetto. Si ricorda che i sorgenti devono essere scritti seguendo opportune regole di coding style e devono essere commentati in maniera chiara e non banale. Es. di commento banale:

```
i++; // incremento della variabile
```

Es. di commento appropriato:

```
// ricerca della prima posizione libera
while ((tree[firstFreePosition] != null) && (i < size))
{
    firstFreePosition = (firstFreePosition + 1) % size;
    i++;
}
if (tree[firstFreePosition] != null)
{
    firstFreePosition = -1;
}
```

`classes`, contenente i file `.class` con il bytecode risultante dalla compilazione dei sorgenti `doc`, contenente la relazione (circa 8–10 pagine) sul progetto (in formato DOC o PDF) nel file `project.doc` o `project.pdf`, dove vengono descritte le scelte progettuali effettuate (per esempio: uso di una certa struttura dati piuttosto che un'altra, descrizione per sommi capi della struttura del progetto, del codice sorgente, dei metodi più importanti e considerazioni sui problemi principali riscontrati). Specificare nella relazione i componenti del gruppo e le relative email. Per la consegna dei file si deve inviare una email con subject “consegna progetto <nomeGruppo>” e in attachment i file richiesti a `pellitta@cs.unibo.it`. Nella email vanno specificati chiaramente i componenti del gruppo con le relative email; va specificata inoltre l'email di riferimento per comunicazioni successive al gruppo. La documentazione deve essere consegnata anche in formato cartaceo lasciandola nella casella di posta del prof. Sangiorgi (che si trova al piano terra del dipartimento di informatica, poco prima dell'ufficio). Vanno consegnate in forma cartacea i seguenti files:

1. `project.doc` (o `.pdf`);
2. tutti i file sorgenti stampati in orizzontale (sotto linux si può usare `enscript -r nomefile`).

Ogni file deve essere graffettato nell'angolo in alto a sinistra. In seguito verrà fissata la data e l'ora in cui presentarsi per la discussione del progetto. **NOTA:** la relazione del progetto è molto importante e progetti con relazioni scarse (ad esempio di una facciata) possono essere anche solo per questo motivo giudicati insufficienti. La relazione non deve contenere un semplice elenco di classi o peggio la ripetizione delle specifiche, ma le idee e le scelte progettuali, soprattutto per quelli considerati punti chiave. In questo senso è gradita anche la sintesi: relazioni di 60 pagine sono ugualmente discutibili di quelle di 2 pagine. Una buona idea può essere di scrivere la relazione durante lo sviluppo del progetto, segnando di volta in volta le cose importanti di cui si discute, e mettendola a posto alla fine.

9 Date di consegna

Ci sono due date di consegna:

- all'inizio di Luglio, alle 24 di Lunedì 2 Luglio 2012: chi consegna il progetto entro questa data farà presumibilmente la discussione entro il mese di Luglio (dipendentemente dai tempi di correzione).
- alla fine di Settembre, alle 12 di Venerdì 21 Settembre 2012

In seguito a tali date verrà fissata la data e l'ora in cui presentarsi per la discussione del progetto e il referente del gruppo riceverà una comunicazione via email.

10 Valutazione del progetto

Il progetto verrà valutato mediante una discussione di gruppo, con tutti i componenti del gruppo presenti. La discussione, e il testing del progetto, sarà fatta nei laboratori ERCOLANI. Si richiede quindi che il progetto sviluppato funzioni sulle macchine dei laboratori Ercolani sulle quali verrà testato. **INVITO QUINDI TUTTI A TESTARE IL PROPRIO SOFTWARE SU TALI MACCHINE** e ad imparare a lavorare e a muoversi anche su di esse. Non è possibile per i componenti di un gruppo effettuare la discussione in incontri separati. Al termine della discussione ad ogni singolo componente verrà assegnato un voto in base all'effettivo contributo dimostrato nel lavoro di progetto. La valutazione del progetto è indipendente dal numero di persone che compongono il gruppo.

11 Note importanti

1. Il progetto deve funzionare: non devono essere presentati progetti non funzionanti, perchè non verranno presi in considerazione.
2. Il progetto deve fare “qualcosa”: un progetto funzionante che scrive solo “hello world” non può essere accettato.
3. Non si accettano (e fanno pessima impressione) email o richieste di eccezioni sui progetti con motivazioni legate a esigenze di laurearsi o di non voler pagare le tasse per un’altro anno: chi ha tali esigenze deve pensarci prima.
4. Chi copia o fa copiare, anche un sola parte del progetto, si vedrà invalidare completamente il progetto senza possibilità di appello. Dovrà quindi rifare un nuovo progetto l’anno successivo.

12 Domande sul progetto

Per le domande sul progetto si consiglia di usare il newsgroup del corso: `unibo.cs.scienzeinternet.so`. Siete invitati a rispondere alle domande dei vostri colleghi sul newsgroup. Risposte corrette verranno positivamente valutate in sede di esame. Solo in situazioni eccezionali è possibile chiedere informazioni mandando email direttamente a me (`pellitta@cs.unibo.it`) o venire a ricevimento (previo appuntamento preso via mail, che specifichi la motivazione della richiesta).