Relazione
“Farming Simulator”

Linda Fabbri, Federico Raffoni, Simone Rega, Montali Giacomo

21 aprile 2021

Indice

Capitolo 1

Analisi

1.1 Requisiti

Il gruppo si pone come obiettivo quello di realizzare "Farming Simulator", un software a scopo ludico basato sul videogioco "Hay Day" [1] sviluppato da Supercell. Il progetto mira alla realizzazione di un ambiente agricolo in 2D, in cui il giocatore interagisce con le colture e gestisce le proprie finanze. Lo scopo del gioco è arrivare all'apice della ricchezza, sbloccando tutti gli appezzamenti di terra e coltivando la maggior varietà di semi possibili.

Requisiti funzionali

- Questa applicazione deve permettere al giocatore di muoversi all'interno della mappa di gioco evitando di salire su zone di mappa non calpestabili.
- Il personaggio ha la possibilità di coltivare piantagioni di grano, patate, pomodori, carote, alberi di ciliege e altri frutti.
- Nella mappa deve comparire una zona per l'allevamento degli animali, grazie ai quali si possono raccogliere latte, uova e altri derivati.
- Farming Simulator mette a disposizione un negozio in cui è possibile vedere il proprio inventario, vendere i prodotti agricoli raccolti in cambio di monete virtuali e comprare nuovi semi da coltivare.
- Per aumentare la difficoltà di gioco vengono messi a disposizione appezzamenti di terreno coltivabili inizialmente bloccati, che si possono sbloccare tramite l'utilizzo delle monete virtuali.

[1] `https://supercell.com/en/games/hayday/`

Requisiti non funzionali

- Farming Simulator deve poter essere eseguito su dispositivi con schermi dotati di risoluzioni differenti.
- Il programma deve poter essere eseguito sia su Windows sia su sistemi operativi Unix

1.2 Analisi e modello del dominio

In Farming Simulator il soggetto principale è il personaggio virtuale, il quale si muove all'interno di una mappa, composta da diversi tipologie di blocchi. All'interno della mappa possiamo trovare blocchi coltivabili (Field Block), blocchi non calpestabili dal personaggio (Wall), blocchi estetici(Water), il quale unico compito è quello di abbellire la mappa, blocchi che compongono la stalla in cui possono muoversi gli animali(Stall), ed infine blocchi "locked"(Unlockable Block), ovvero che inizialmente sono bloccati e possono essere sbloccati durante la partita in cambio di monete virtuali.

Il personaggio (Player), grazie all'inventario (Inventory), è in grado di mantenere risorse al suo interno. Tra le risorse collezionabili possiamo trovare: il cibo (Food) ottenuto dalla raccolta di coltivazioni o di animali oppure i semi (Seed). Questi ultimi possono essere comprati nel negozio (Shop) del videogioco, dentro il quale è anche possibile vendere il cibo che si è raccolto.

Il personaggio può interaggire con blocchi e animali (Animals), in particolare può piantare e raccogliere i semi all'interno dei blocchi coltivabili (FieldBlock), sbloccare i blocchi non ancora accessibili (UnlockableBlock) e raccogliere i prodotti animali.

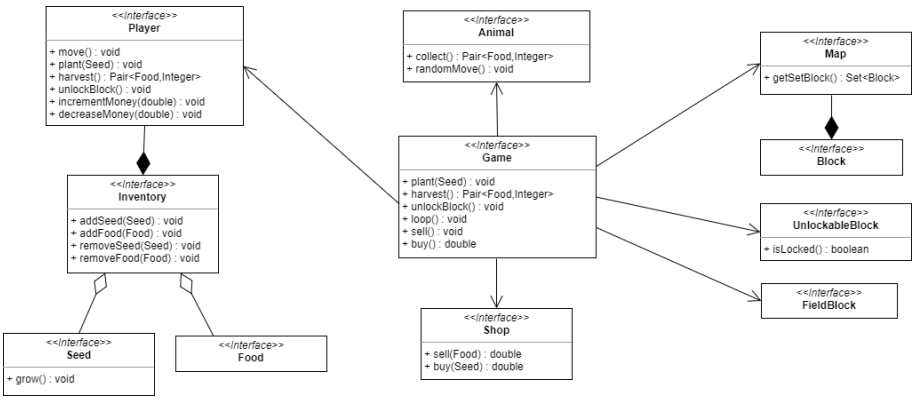

Figura 1.1: Schema UML dell'analisi del problema, con rappresentate le entità principali ed i rapporti fra loro

Capitolo 2

Design

2.1 Architettura

L'architettura di Farming Simulator è di tipo MVC. In particolare è stata sviluppata l'interfaccia Game che funge da controller e si posiziona tra view e model e gestisce eventuali interazioni tra componenti. All'arrivo di un input, game, si occupa di ordinare al Model il cambio di stato.

Gli input da tastiera provenienti dall'utente passano attraverso la classe KeyNotifier, la quale si occupa di informare il game dell'avvenuto input.

La parte di View è formata da diversi GameDrawer, i quali si occupano di interrogare il game e mostrare a video il gioco.

Quando viene catturato un input da tastiera, il GameDrawer avvisa il keyNotifier dell'avvenuta pressione, così facendo il KeyNotifier è un observable per GameDrawer. In questo modo il GameDrawer non si deve occupare della gestione degli input, spostando il carico di lavoro al KeyNotifier; questa astrazione ci permette di non dipendere dall'implementazione e poter cambiare il binding[1] o addirittura dispositivo di input (ad esempio un Joystick).

In Figura 2.1 è esemplificato il diagramma UML che descrive l'architettura MVC del progetto.

2.2 Design dettagliato

In questa sezione si possono approfondire alcuni elementi del design con maggior dettaglio. Mentre ci attendiamo principalmente (o solo) interfacce negli

[1]Binding = associazione di un tasto ad un determinato movimento e/o interazione visiva in gioco (ad esempio [WASD] per spostarsi nella mappa)

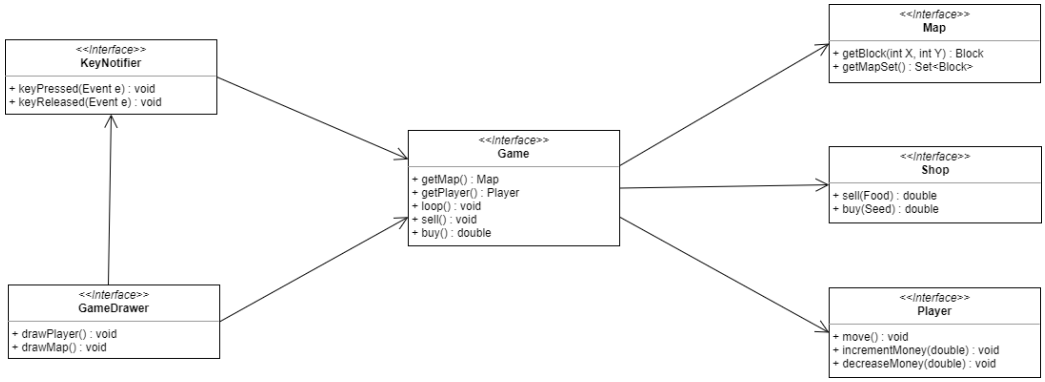

Figura 2.1: Schema UML architetturale di Farming Simulator, MVC.

schemi UML delle sezioni precedenti, in questa sezione è necessario scendere in maggior dettaglio presentando la struttura di alcune sottoparti rilevanti dell'applicazione. È molto importante che, descrivendo un problema, quando possibile si mostri che non si è re-inventata la ruota ma si è applicato un design pattern noto. È assolutamente inutile, ed è anzi controproducente, descrivere classe-per-classe (o peggio ancora metodo-per-metodo) com'è fatto il vostro software: è un livello di dettaglio proprio della documentazione dell'API (deducibile dalla Javadoc).

È necessario che ciascun membro del gruppo abbia una propria sezione di design dettagliato, di cui sarà il solo responsabile. Ciascun autore dovrà spiegare in modo corretto e giustamente approfondito (non troppo in dettaglio, non superficialmente) il proprio contributo. È importante focalizzarsi sulle scelte che hanno un impatto positivo sul riuso, sull'estensibilità, e sulla chiarezza dell'applicazione. Esattamente come nessun ingegnere meccanico presenta un solo foglio con l'intero progetto di una vettura di Formula 1, ma molteplici fogli di progetto che mostrano a livelli di dettaglio differenti le varie parti della vettura e le modalità di connessione fra le parti, così ci aspettiamo che voi, futuri ingegneri informatici, ci presentiate prima una visione globale del progetto, e via via siate in grado di dettagliare le singole parti, scartando i componenti che non interessano quella in esame. Per continuare il parallelo con la vettura di Formula 1, se nei fogli di progetto che mostrano il design delle sospensioni anteriori appaiono pezzi che appartengono al volante o al turbo, c'è una chiara indicazione di qualche problema di design.

Usare correttamente i design pattern in questa sezione è molto importante: se vengono utilizzati correttamente, è molto probabile riuscire a progettare il software in modo corretto, estensibile, e riusabile. Per ogni pattern utilizzato si presenti:

- almeno un paragrafo che spieghi come è reificato nel progetto (ad esempio: nel caso di Template Method, qual è il metodo template; nel caso di Strategy, quale interfaccia del progetto rappresenta la strategia, e quali sono le sue implementazioni; nel caso di Decorator, qual è la classe astratta che fa da Decorator e quali sono le sue implementazioni concrete; eccetera);
- almeno uno schema UML che grafichi quanto sopra descritto.

La presenza di pattern di progettazione *correttamente utilizzati* è valutata molto positivamente. L'uso inappropriato è invece valutato negativamente: a tal proposito, si raccomanda di porre particolare attenzione all'abuso di Singleton, che, se usato in modo inappropriato, è di fatto un anti-pattern.

Elementi positivi

- Ogni membro del gruppo discute le proprie decisioni di progettazione, ed in particolare le azioni volte ad anticipare possibili cambiamenti futuri (ad esempio l'aggiunta di una nuova funzionalità, o il miglioramento di una esistente).
- Si identificano, utilizzano *appropriatamente*, e descrivono come suggerito diversi design pattern.
- Ogni membro del gruppo identifica i pattern utilizzati nella sua sottoparte.
- Si mostrano gli aspetti di design più rilevanti dell'applicazione, mettendo in luce la maniera in cui si è costruita la soluzione ai problemi descritti nell'analisi.
- Si tralasciano aspetti strettamente implementativi e quelli non rilevanti, non mostrandoli negli schemi UML (ad esempio, campi privati) e non descrivendoli.
- Si mostrano le principali interazioni fra le varie componenti che collaborano alla soluzione di un determinato problema.
- Ciascun design pattern identificato presenta una piccola descrizione del problema calato nell'applicazione, uno schema UML che ne mostra la concretizzazione nelle classi del progetto, ed una breve descrizione della motivazione per cui tale pattern è stato scelto. Ad esempio, se si dichiara di aver usato Observer, è necessario specificare chi sia l'observable e chi l'observer; se si usa Template Method, è necessario indicare

quale sia il metodo template; se si usa Strategy, è necessario identificare l'interfaccia che rappresenta la strategia; e via dicendo.

Elementi negativi

- Il design del modello risulta scorrelato dal problema descritto in analisi.
- Si tratta in modo prolisso, classe per classe, il software realizzato, o comunque si riduce la sezione ad un mero elenco di quanto fatto.
- Non si presentano schemi UML esemplificativi.
- Non si individuano design pattern, o si individuano in modo errato (si spaccia per design pattern qualcosa che non lo è).
- Si utilizzano design pattern in modo inopportuno. Un esempio classico è l'abuso di Singleton per entità che possono essere univoche ma non devono necessariamente esserlo. Si rammenta che Singleton ha senso nel secondo caso (ad esempio `System` e `Runtime` sono singleton), mentre rischia di essere un problema nel secondo. Ad esempio, se si rendesse singleton il motore di un videogioco, sarebbe impossibile riusarlo per costruire un server per partite online (dove, presumibilmente, si gestiscono parallelamente più partite).
- Si producono schemi UML caotici e difficili da leggere, che comprendono inutili elementi di dettaglio.
- Si presentano schemi UML con classi (nel senso UML del termine) che "galleggiano" nello schema, non connesse, ossia senza relazioni con il resto degli elementi inseriti.
- Si tratta in modo inutilmente prolisso la divisione in package, elencando ad esempio le classi una per una.

Esempio minimale (e quindi parziale) di sezione di progetto con UML ben realizzati

In questa sezione ci si concentrerà sugli aspetti di personalità e sul funzionamento del reporting di GLaDOS.

Il sistema per la gestione della personalità utilizza il pattern Strategy, come da Figura 2.2: le implementazioni di `Personality` possono essere modificate, e la modifica impatta direttamente sul comportamento di GLaDOS.

Figura 2.2: Rappresentazione UML del pattern Strategy per la personalità di GLaDOS

Figura 2.3: Rappresentazione UML dell'applicazione del pattern Template Method alla gerarchia delle Personalità

Sono state attualmente implementate due personalità, una buona ed una cattiva. Quella buona restituisce sempre una torta vera, mentre quella cattiva restituisce sempre la promessa di una torta che verrà in realtà disattesa. Dato che le due personalità differiscono solo per il comportamento da effettuarsi in caso di percorso completato con successo, è stato utilizzato il pattern template method per massimizzare il riuso, come da Figura 2.3. Il metodo template è `onSuccess()`, che chiama un metodo astratto e protetto `makeCake()`.

Figura 2.4: Il pattern Observer è usato per consentire a GLaDOS di informare tutti i sistemi di output in ascolto

Per quanto riguarda il reporting, è stato utilizzato il pattern Observer per consentire la comunicazione uno-a-molti fra `GLaDOS` ed i sistemi di output. `GLaDOS` è observable, e le istanze di `Input` sono observer. Il suo utilizzo è esemplificato in Figura 2.4

Contro-esempio: pessimo diagramma UML

In Figura 2.5 è mostrato il modo **sbagliato** di fare le cose. Questo schema è fatto male perché:

- È caotico.
- È difficile da leggere e capire.
- Vi sono troppe classi, e non si capisce bene quali siano i rapporti che intercorrono fra loro.
- Si mostrano elementi implementativi irrilevanti, come i campi e i metodi privati nella classe `AbstractEnvironment`.
- Se l'intenzione era quella di costruire un diagramma architetturale, allora lo schema è ancora più sbagliato, perché mostra pezzi di implementazione.

- Una delle classi, in alto al centro, galleggia nello schema, non connessa a nessuna altra classe, e di fatto costituisce da sola un secondo schema UML scorrelato al resto
- Le interfacce presentano tutti i metodi e non una selezione che aiuti il lettore a capire quale parte del sistema si vuol mostrare.

Figura 2.5: Schema UML mal fatto e con una pessima descrizione, che non aiuta a capire. Don't try this at home.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

In questo progetto abbiamo deciso di testare prevalentemente la parte di Model poiché riteniamo sia la più propensa ad avere bug.

Per svolgere test completamente automatizzati sul progetto è stato usato JUnit5.

Le classi che sono state testate sono:

- **Player:** sulla classe Player sono state testate movimento all'interno della mappa, le relative collisioni; aumento e diminuzione dei soldi.
- **Shop:** sono state testate le funzionalità di Buy e Sell, controllando inventario e soldi del Player
- **Saver:** è stata testata la funzionalità di salvataggio della partita controllando lo stato di Player e Map
- **Interaction:** sono state testate le interazioni con:
  - Piante (plant[1] e harvest[2])
  - Animali (raccolta dei prodotti animali))
  - UnlockableBlock (sbloccaggio del blocco con e senza soldi)

[1]Plant = piantare il seme nel FieldBlock

[2]Harvest = raccogliere l'elemento Food dal FieldBlock

3.2 Metodologia di lavoro

Ci aspettiamo, leggendo questa sezione, di trovare conferma alla divisione operata nella sezione del design di dettaglio, e di capire come è stato svolto il lavoro di integrazione. **Andrà realizzata una sotto-sezione separata per ciascuno studente** che identifichi le porzioni di progetto sviluppate, separando quelle svolte in autonomia da quelle sviluppate in collaborazione. Diversamente dalla sezione di design, in questa è consentito elencare package/classi, se lo studente ritiene sia il modo più efficace di convogliare l'informazione. Si ricorda che l'impegno deve giustificare circa 40-50 ore di sviluppo (è normale e fisiologico che approssimativamente la metà del tempo sia impiegata in analisi e progettazione).

Elementi positivi

- Si identifica con precisione il ruolo di ciascuno all'interno del gruppo, ossia su quale parte del progetto ciascuno dei componenti si è concentrato maggiormente.
- La divisione dei compiti è equa, ossia non vi sono membri del gruppo che hanno svolto molto più lavoro di altri.
- La divisione dei compiti è coerente con quanto descritto nelle parti precedenti della relazione.
- La divisione dei compiti è realistica, ossia le dipendenze fra le parti sviluppate sono minime.
- Si identifica quale parte del software è stato sviluppato da tutti i componenti insieme.
- Si spiega in che modo si sono integrate le parti di codice sviluppate separatamente, evidenziando eventuali problemi. Ad esempio, una strategia è convenire sulle interfacce da usare (ossia, occuparsi insieme di stabilire l'architettura) e quindi procedere indipendentemente allo sviluppo di parti differenti. Una possibile problematica potrebbe essere una dimenticanza in fase di design architetturale che ha costretto ad un cambio e a modifiche in fase di integrazione. Una situazione simile è la norma nell'ingegneria di un sistema software non banale, ed il processo di progettazione top-down con raffinamento successivo è il così detto processo "a spirale".
- Si descrive in che modo è stato impiegato il DVCS.

Elementi negativi

- Non si chiarisce chi ha fatto cosa.
- C'è discrepanza fra questa sezione e le sezioni che descrivono il design dettagliato.
- Tutto il progetto è stato svolto lavorando insieme invece che assegnando una parte a ciascuno.
- Non viene descritta la metodologia di integrazione delle parti sviluppate indipendentemente.
- Uso superficiale del DVCS.

3.3 Note di sviluppo

Questa sezione, come quella riguardante il design dettagliato va svolta **singolarmente da ogni membro del gruppo**.

Ciascuno dovrà mettere in evidenza eventuali particolarità del suo metodo di sviluppo, ed in particolare:

- **Elencare** (fare un semplice elenco per punti, non un testo!) le feature *avanzate* del linguaggio e dell'ecosistema Java che sono state utilizzate. Le feature di interesse sono:
  - Progettazione con generici, ad esempio costruzione di nuovi tipi generici, e uso di generici bounded. Uso di classi generiche di libreria non è considerato avanzato.
  - Uso di lambda expressions
  - Uso di `Stream`, di `Optional` o di altri costrutti funzionali
  - Uso della reflection
  - Definizione ed uso di nuove annotazioni
  - Uso del Java Platform Module System
  - Uso di parti di libreria non spiegate a lezione (networking, compressione, parsing XML, eccetera...)
  - Uso di librerie di terze parti (incluso JavaFX): Google Guava, Apache Commons...
  - Uso di build systems

Si faccia molta attenzione a non scrivere banalità, elencando qui features di tipo "core", come le eccezioni, le enumerazioni, o le inner class: nessuna di queste è considerata avanzata.

- Descrivere *molto brevemente* le librerie utilizzate nella propria parte di progetto, se non trattate a lezione (ossia, se librerie di terze parti e/o se componenti del JDK non visti, come le socket). Si ricorda che l'utilizzo di librerie è valutato *positivamente*.

- Sviluppo di algoritmi particolarmente interessanti *non forniti da alcuna libreria* (spesso può convenirvi chiedere sul forum se ci sia una libreria per fare una certa cosa, prima di gettarvi a capofitto per scriverla voi stessi).

In questa sezione, *dopo l'elenco*, è anche bene evidenziare eventuali pezzi di codice "riadattati" (o scopiazzati...) da Internet o da altri progetti, pratica che tolleriamo ma che non raccomandiamo. I pattern di design, invece **non** vanno messi qui. L'uso di pattern di design (come suggerisce il nome) è un aspetto avanzato di design, non di implementazione, e non va in questa sezione.

Elementi positivi

- Si elencano gli aspetti avanzati di linguaggio che sono stati impiegati

- Si elencano le librerie che sono state utilizzate

- Si descrivono aspetti particolarmente complicati o rilevanti relativi all'implementazione, ad esempio, in un'applicazione performance critical, un uso particolarmente avanzato di meccanismi di caching, oppure l'implementazione di uno specifico algoritmo.

- Se si è utilizzato un particolare algoritmo, se ne cita la fonte originale. Ad esempio, se si è usato Mersenne Twister per la generazione dei numeri pseudo-random, si cita [**?**].

- Si identificano parti di codice prese da altri progetti, dal web, o comunque scritte in forma originale da altre persone. In tal senso, si ricorda che agli ingegneri non è richiesto di re-inventare la ruota continuamente: se si cita debitamente la sorgente è tollerato fare uso di di snippet di codice per risolvere velocemente problemi non banali. Nel caso in cui si usino snippet di codice di qualità discutibile, oltre a menzionarne l'autore originale si invitano gli studenti ad adeguare tali parti di codice

agli standard e allo stile del progetto. Contestualmente, si fa presente che è largamente meglio fare uso di una libreria che copiarsi pezzi di codice: qualora vi sia scelta (e tipicamente c'è), si preferisca la prima via.

Elementi negativi

- Si elencano feature core del linguaggio invece di quelle segnalate. Esempi di feature core da non menzionare sono:
  - eccezioni;
  - classi innestate;
  - enumerazioni;
  - interfacce.
- Si elencano applicazioni di terze parti (peggio se per usarle occorre licenza, e lo studente ne è sprovvisto) che non c'entrano nulla con lo sviluppo, ad esempio:
  - Editor di grafica vettoriale come Inkscape o Adobe Illustrator;
  - Editor di grafica scalare come GIMP o Adobe Photoshop;
  - Editor di audio come Audacity;
  - Strumenti di design dell'interfaccia grafica come SceneBuilder: il codice è in ogni caso inteso come sviluppato da voi.
- Si descrivono aspetti di scarsa rilevanza, o si scende in dettagli inutili.
- Sono presenti parti di codice sviluppate originalmente da altri che non vengono debitamente segnalate. In tal senso, si ricorda agli studenti che i docenti hanno accesso a tutti i progetti degli anni passati, a Stack Overflow, ai principali blog di sviluppatori ed esperti Java (o sedicenti tali), ai blog dedicati allo sviluppo di soluzioni e applicazioni (inclusi blog dedicati ad Android e allo sviluppo di videogame), nonché ai social network. Conseguentemente, è *molto* conveniente *citare* una fonte ed usarla invece di tentare di spacciare per proprio il lavoro di altri.
- Si elencano design pattern

Capitolo 4

Commenti finali

In quest'ultimo capitolo si tirano le somme del lavoro svolto e si delineano eventuali sviluppi futuri.

Nessuna delle informazioni incluse in questo capitolo verrà utilizzata per formulare la valutazione finale, a meno che non sia assente o manchino delle sezioni obbligatorie. Al fine di evitare pregiudizi involontari, l'intero capitolo verrà letto dai docenti solo dopo aver formulato la valutazione.

4.1 Autovalutazione e lavori futuri

È richiesta una sezione per ciascun membro del gruppo, obbligatoriamente. Ciascuno dovrà autovalutare il proprio lavoro, elencando i punti di forza e di debolezza in quanto prodotto. Si dovrà anche cercare di descrivere *in modo quanto più obiettivo possibile* il proprio ruolo all'interno del gruppo. Si ricorda, a tal proposito, che ciascuno studente è responsabile solo della propria sezione: non è un problema se ci sono opinioni contrastanti, a patto che rispecchino effettivamente l'opinione di chi le scrive. Nel caso in cui si pensasse di portare avanti il progetto, ad esempio perché effettivamente impiegato, o perché sufficientemente ben riuscito da poter esser usato come dimostrazione di esser capaci progettisti, si descriva brevemente verso che direzione portarlo.

4.2 Difficoltà incontrate e commenti per i docenti

Questa sezione, **opzionale**, può essere utilizzata per segnalare ai docenti eventuali problemi o difficoltà incontrate nel corso o nello svolgimento del

progetto, può essere vista come una seconda possibilità di valutare il corso (dopo quella offerta dalle rilevazioni della didattica) avendo anche conoscenza delle modalità e delle difficoltà collegate all'esame, cosa impossibile da fare usando le valutazioni in aula per ovvie ragioni. È possibile che alcuni dei commenti forniti vengano utilizzati per migliorare il corso in futuro: sebbene non andrà a vostro beneficio, potreste fare un favore ai vostri futuri colleghi. Ovviamente *il contenuto della sezione non impatterà il voto finale.*

Appendice A

Guida utente

Capitolo in cui si spiega come utilizzare il software. Nel caso in cui il suo uso sia del tutto banale, tale capitolo può essere omesso. A tal riguardo, si fa presente agli studenti che i docenti non hanno mai utilizzato il software prima, per cui aspetti che sembrano del tutto banali a chi ha sviluppato l'applicazione possono non esserlo per chi la usa per la prima volta. Se, ad esempio, per cominciare una partita con un videogioco è necessario premere la barra spaziatrice, o il tasto "P", è necessario che gli studenti lo segnalino.

Elementi positivi

- Si istruisce in modo semplice l'utente sull'uso dell'applicazione, eventualmente facendo uso di schermate e descrizioni.

Elementi negativi

- Si descrivono in modo eccessivamente minuzioso tutte le caratteristiche, anche minori, del software in oggetto.
- Manca una descrizione che consenta ad un utente qualunque di utilizzare almeno le funzionalità primarie dell'applicativo.

Appendice B

Esercitazioni di laboratorio

In questo capitolo ciascuno studente elenca gli esercizi di laboratorio che ha svolto (se ne ha svolti), elencando i permalink dei post sul forum dove è avvenuta la consegna.

Esempio

B.0.1 Paolino Paperino

- Laboratorio 04: `https://virtuale.unibo.it/mod/forum/discuss.php?d=12345#p123456`
- Laboratorio 06: `https://virtuale.unibo.it/mod/forum/discuss.php?d=22222#p222222`
- Laboratorio 09: `https://virtuale.unibo.it/mod/forum/discuss.php?d=99999#p999999`

B.0.2 Paperon De Paperoni

- Laboratorio 04: `https://virtuale.unibo.it/mod/forum/discuss.php?d=12345#p123456`
- Laboratorio 05: `https://virtuale.unibo.it/mod/forum/discuss.php?d=22222#p222222`
- Laboratorio 06: `https://virtuale.unibo.it/mod/forum/discuss.php?d=99999#p999999`
- Laboratorio 07: `https://virtuale.unibo.it/mod/forum/discuss.php?d=22222#p222222`

- Laboratorio 08: `https://virtuale.unibo.it/mod/forum/discuss.php?d=99999#p999999`
- Laboratorio 09: `https://virtuale.unibo.it/mod/forum/discuss.php?d=22222#p222222`
- Laboratorio 10: `https://virtuale.unibo.it/mod/forum/discuss.php?d=99999#p999999`
- Laboratorio 11: `https://virtuale.unibo.it/mod/forum/discuss.php?d=22222#p222222`

Bibliografia