

# Relazione “SMOL”

Ettore Farinelli  
Marco Galeri ciao  
Giovanni Paradisi  
Mounir Samite

1 aprile 2023

# Indice

<b>1</b>	<b>Analisi</b>	<b>2</b>
1.1	Requisiti . . . . .	2
1.2	Analisi e modello del dominio . . . . .	3
<b>2</b>	<b>Design</b>	<b>4</b>
2.1	Architettura . . . . .	4
2.2	Design dettagliato . . . . .	5
<b>3</b>	<b>Sviluppo</b>	<b>6</b>
3.1	Testing automatizzato . . . . .	6
3.2	Metodologia di lavoro . . . . .	7
3.3	Note di sviluppo . . . . .	8
3.3.1	Esempio . . . . .	11
<b>4</b>	<b>Commenti finali</b>	<b>13</b>
4.1	Autovalutazione e lavori futuri . . . . .	13
4.2	Difficoltà incontrate e commenti per i docenti . . . . .	13
<b>A</b>	<b>Guida utente</b>	<b>15</b>

# Capitolo 1

## Analisi

### 1.1 Requisiti

Il software, commissionato dal gestore del centro di ricerca “Aperture Laboratories Inc.”<sup>1</sup>, mira alla costruzione di una intelligenza artificiale di nome GLaDOS (Genetic Lifeform and Disk Operating System). Per intelligenza artificiale si intende un software in grado di assumere decisioni complesse in maniera semi autonoma sugli argomenti di sua competenza, a partire dai vincoli e dagli obiettivi datigli dall’utente.

#### Requisiti funzionali

- La suddetta intelligenza artificiale dovrà occuparsi di coordinare le attività all’interno delle camere di test di Aperture, guidando l’utente attraverso un certo numero di sfide di difficoltà crescente. Una camera di test è un ambiente realizzato da Aperture Laboratories Inc. al fine di mettere alla prova le proprie tecnologie di manipolazione dell’ambiente. All’interno della camera di test, un soggetto qualificato è incaricato di sfruttare gli strumenti messi a disposizione da Aperture per risolvere alcuni rompicapi. I rompicapi sono di tipo fisico (ad esempio, manipolazione di oggetti, pressione di pulsanti, azionamento di leve), e si ritengono conclusi una volta che il soggetto riesce a trovare l’uscita dalla camera di test.
- Il piano preciso ed il numero delle sfide sarà variabile, e GLaDOS dovrà essere in grado di adattarsi dinamicamente e di fornire indicazioni di guida.

---

<sup>1</sup><http://aperturescience.com/>

- La personalità di GLaDOS dovrà essere modificabile.
- GLaDOS dovrà essere in grado di comunicare col reparto cucina di Aperture, per ordinare torte da donare agli utenti che completassero l'ultima camera di test con successo.

### **Requisiti non funzionali**

- GLaDOS dovrà essere estremamente efficiente nell'uso delle risorse. Le specifiche tecniche parlano della possibilità di funzionare su dispositivi alimentati da una batteria a patata.

## **1.2 Analisi e modello del dominio**

GLaDOS dovrà essere in grado di accedere ad un'insieme di camere di test. Tale insieme di camere prende il nome di percorso. Ciascuna camera è composta di challenge successivi. GLaDOS è responsabile di associare a ciascun challenge un insieme di consigli (suggestions) destinati all'utente (subject), dipendenti da possibili eventi. GLaDOS dovrà poter comunicare coi locali cucina per approntare le torte. Le torte potranno essere dolci, oppure semplici promesse di dolci che verranno disattese.

Gli elementi costitutivi il problema sono sintetizzati in Figura 1.1.

La difficoltà primaria sarà quella di riuscire a correlare lo stato corrente dell'utente e gli eventi in modo tale da generare i corretti suggerimenti. Questo richiederà di mettere in campo appropriate strategie di intelligenza artificiale.

Data la complessità di elaborare consigli via AI senza intervento umano, la prima versione del software fornita prevederà una serie di consigli forniti dall'utente.

Il requisito non funzionale riguardante il consumo energetico richiederà studi specifici sulle performance di GLaDOS che non potranno essere effettuati all'interno del monte ore previsto: tale feature sarà oggetto di futuri lavori.

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 GLaDOS segue il pattern architetturale MVC. Più nello specifico, a livello architetturale, si è scelto di utilizzare MVC in forma “ECB”, ossia “entity-control-boundary”<sup>1</sup>. GLaDOS implementa l'interfaccia AI, ed è il controller del sistema. Essendo una intelligenza artificiale, è una classe attiva. GLaDOS accetta la registrazione di Input ed Output, che fanno parte della “view” di MVC, e sono il “boundary” di ECB. Gli Input rappresentano delle nuove informazioni che vengono fornite all'IA, ad esempio delle modifiche nel valore di un sensore, oppure un comando da parte dell'operatore. Questi input infatti forniscono eventi. Ottenere un evento è un'operazione bloccante: chi la esegue resta in attesa di un effettivo evento. Di fatto, quindi, GLaDOS si configura come entità *reattiva*. Ogni volta che c'è un cambio alla situazione del soggetto, GLaDOS notifica i suoi Output, informandoli su quale sia la situazione corrente. Conseguentemente, GLaDOS è un “observable” per Output.

Figura 2.1: Schema UML architetturale di GLaDOS. L'interfaccia GLaDOS è il controller del sistema, mentre **Input** ed **Output** sono le interfacce che mappano la view (o, più correttamente in questo specifico esempio, il boundary). Un'eventuale interfaccia grafica interattiva dovrà implementarle entrambe.

Con questa architettura, possono essere aggiunti un numero arbitrario di input ed output all'intelligenza artificiale. Ovviamente, mentre l'aggiunta di output è semplice e non richiede alcuna modifica all'IA, la presenza di nuovi

---

<sup>1</sup>Si fa presente che il pattern ECB effettivamente esiste in letteratura come “istanza” di MVC, e chi volesse può utilizzarlo come reificazione di MVC.

tipi di evento richiede invece in potenza aggiunte o rifiniture a GLaDOS. Questo è dovuto al fatto che nuovi Input rappresentano di fatto nuovi elementi della business logic, la cui alterazione od espansione inevitabilmente impatta il controller del progetto.

In Figura 2.1 è esemplificato il diagramma UML architetturale.

## **2.2 Design dettagliato**

**Ettore Farinelli**

roba di Ettore

**Marco Galeri**

roba di Marco

**Giovanni Paradisi**

roba di Giovanni

**Mounir Samite**

roba di Mounir

# Capitolo 3

## Sviluppo

### 3.1 Testing automatizzato

Il testing automatizzato è un requisito di qualunque progetto software che si rispetti, e consente di verificare che non vi siano regressioni nelle funzionalità a fronte di aggiornamenti. Per quanto riguarda questo progetto è considerato sufficiente un test minimale, a patto che sia completamente automatico. Test che richiedono l'intervento da parte dell'utente sono considerati *negativamente* nel computo del punteggio finale.

#### Elementi positivi

- Si descrivono molto brevemente i componenti che si è deciso di sottoporre a test automatizzato.
- Si utilizzano suite specifiche (e.g. JUnit) per il testing automatico.

#### Elementi negativi

- Non si realizza alcun test automatico.
- La non presenza di testing viene aggravata dall'adduzione di motivazioni non valide. Ad esempio, si scrive che l'interfaccia grafica non è testata automaticamente perché è *impossibile* farlo<sup>1</sup>.
- Si descrive un testing di tipo manuale in maniera prolissa.

---

<sup>1</sup>Testare in modo automatico le interfacce grafiche è possibile (si veda, come esempio, <https://github.com/TestFX/TestFX>), semplicemente nel corso non c'è modo e tempo di introdurvi questo livello di complessità. Il fatto che non vi sia stato insegnato come farlo non implica che sia impossibile!

- Si descrivono test effettuati manualmente che sarebbero potuti essere automatizzati, ad esempio scrivendo che si è usata l'applicazione manualmente.
- Si descrivono test non presenti nei sorgenti del progetto.
- I test, quando eseguiti, falliscono.

## 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 **sin-  
golarmente da ogni membro del gruppo**. Nella prima parte, ciascuno dovrà mostrare degli esempi di codice particolarmente ben realizzati, che dimostrino proefficienza con funzionalità avanzate del linguaggio e capacità di spingersi oltre le librerie mostrate a lezione.

- **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. L'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 di reflection
- Definizione ed uso di nuove annotazioni
- Uso del Java Platform Module System
- Uso di parti della libreria JDK non spiegate a lezione (networking, compressione, parsing XML, eccetera...)
- Uso di librerie di terze parti (incluso JavaFX): Google Guava, Apache Commons...
- 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.
- Per ogni feature avanzata, mostrata, includere:
  - Nome della feature
  - Permalink GitHub al punto nel codice in cui è stata utilizzata

In questa sezione, *dopo l'elenco*, vanno menzionati ed attribuiti con precisione eventuali pezzi di codice “riadattati” (o scopiazzati...) da Internet o da altri progetti, pratica che tolleriamo ma che non raccomandiamo. Si rammenta agli studenti che non è consentito partire da progetti esistenti e procedere per modifiche successive. Si ricorda anche che i docenti hanno in mano strumenti antiplagio piuttosto raffinati e che “capiscono” il codice e la storia delle modifiche del progetto, per cui tecniche banali come cambiare nomi (di classi, metodi, campi, parametri, o variabili locali), aggiungere o togliere commenti, oppure riordinare i membri di una classe vengono individuate senza problemi. Le regole del progetto spiegano in dettaglio l’approccio dei docenti verso atti gravi come il plagiarismo.

I pattern di design **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
- Per ciascun elemento, si fornisce un permalink
- Ogni permalink fa riferimento ad uno snippet di codice scritto dall’autore della sezione (i docenti verificheranno usando `git blame`)

- Se si è utilizzato un particolare algoritmo, se ne cita la fonte originale. Ad esempio, se si è usato Mersenne Twister per la generazione di numeri pseudo-random, si cita [MN98].
- 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 open source 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, ai blog dedicati allo sviluppo di soluzioni e applicazioni (inclusi blog dedicati ad Android e allo sviluppo di videogame), nonché ai vari GitHub, GitLab, e Bitbucket. Conseguentemente, è *molto* conveniente *citare* una fonte ed usarla invece di tentare di spacciare per proprio il lavoro di altri.

- Si elencano design pattern

### 3.3.1 Esempio

#### Utilizzo della libreria SLF4J

Utilizzata in vari punti. Un esempio è <https://github.com/AlchemistSimulator/Alchemist/blob/5c17f8b76920c78d955d478864ac1f11508ed9ad/alchemist-swingui/src/main/java/it/unibo/alchemist/boundary/swingui/effect/impl/EffectBuilder.java#L49>

#### Utilizzo di LoadingCache dalla libreria Google Guava

Permalink: <https://github.com/AlchemistSimulator/Alchemist/blob/d8a1799027d7d685569e15316a32e6394632ce71/alchemist-incarnation-protelis/src/main/java/it/unibo/alchemist/protelis/AlchemistExecutionContext.java#L141-L143>

#### Utilizzo di Stream e lambda expressions

Usate pervasivamente. Il seguente è un singolo esempio. Permalink: <https://github.com/AlchemistSimulator/Alchemist/blob/d8a1799027d7d685569e15316a32e6394632ce71/alchemist-incarnation-protelis/src/main/java/it/unibo/alchemist/model/ProtelisIncarnation.java#L98-L120>

#### Scrittura di metodo generico con parametri contravarianti

Permalink: <https://github.com/AlchemistSimulator/Alchemist/blob/d8a1799027d7d685569e15316a32e6394632ce71/alchemist-incarnation-protelis/src/main/java/it/unibo/alchemist/protelis/AlchemistExecutionContext.java#L141-L143>

#### Protezione da corse critiche usando Semaphore

Permalink: <https://github.com/AlchemistSimulator/Alchemist/blob/d8a1799027d7d685569e15316a32e6394632ce71/alchemist-incarnation-protelis/>

src/main/java/it/unibo/alchemy/model/ProtelisIncarnation.java#  
L388-L440

# 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.



# Bibliografia

- [MN98] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, January 1998.