

Documentazione richiesta per progetto

INGEGNERIA DEI REQUISITI IEEE830

1.1 SCOPO DEL DOCUMENTO

Questo documento rappresenta la struttura proposta dallo standard IEEE830 per la definizione dei requisiti. Questi rappresenta uno dei criteri fondamentali per valutare l'adeguatezza dell'implementazione ottenuta e un punto di partenza per le seguenti fasi dello sviluppo dell'applicazione.

1.2 SCOPO DEL PROGETTO

Brick Breaker è un applicazione per piattaforme Java che permette di giocare al gioco Breakout: uno strato di mattoncini posizionato nella parte alta dello schermo e l'obiettivo è distruggerli tutti facendo rimbalzare ripetutamente una pallina su un paddle.

Il sistema è sviluppato in Java mediante l'Ide Eclipse che tramite l'importazione del framework Java Swing e la libreria Java AWT permette lo sviluppo dell'interfaccia grafica.

1.3 DEFINIZIONI, ACRONIMI E ABBREVIAZIONI

1.4 RIFERIMENTI

[https://it.wikipedia.org/wiki/Breakout_\(videogioco\)](https://it.wikipedia.org/wiki/Breakout_(videogioco))

1.5 PANORAMICA

Dopo questa introduzione, il documento è composto da due ulteriori fasi: una fase che sviluppa una descrizione globale dei requirements ed una parte che classifica tutti i differenti requisiti individuali utilizzando il modello Moscow, indicando la priorità tra le parentesi:

- la fisica di gioco, definibile come il cuore fondante dell'applicazione, definisce i movimenti che la pallina può eseguire (must have)
- l'interfaccia grafica è gestita tramite una serie di Jpanel di Java, che si alternano a secondo dell'azione svolta dall'utente, i quali sono contenuti all'interno di un Jcontainer (must have)
- l'interfaccia dei pulsanti fisici, necessaria per poter verificare la pressione di un determinato tasto (should have)
- la gestione dei punteggi, necessaria per capire quanti punti si sono raccolti durante una partita (could have)
- la lettura e la scrittura dei file, utile per tenere un registro locale con i risultati delle partite (could have)
- i potenziamenti, utili per diversificare lo stile di gioco di una partita (could have)
- la modifica delle impostazioni di gioco, utile per la QOL (could have)

1.6 DESCRIZIONE GENERALE

L'applicazione si sviluppa su un unico punto di vista, quello dell'utente, il quale si può osservare del Diagramma dei Casi d'Uso, rappresentato come analisi preliminare per capire le principali funzionalità da implementare.

L'utente una volta avviata l'applicazione potrà decidere se avviare la partita con le impostazioni base oppure potrà cambiare alcuni parametri di gioco, come il numero di vite. Di conseguenza, alla fine della partita, gli sarà possibile registrare il risultato immettendo il proprio nome utente il quale sarà registrato insieme al risultato.

1.7 PROSPETTIVA DEL PRODOTTO

Per la fase di elicitation sono state usate le strategie dell'interview e del questionario.

Le fasi di verification and validation, negoziazione si sono verificate dopo una prima fase di prototipizzazione.

1.8 FUNZIONI DEL PRODOTTO

A seguire sono riportate le funzionalità centrali:

- possibilità di eseguire partite di livelli differenti
- possibilità di registrare il risultato
- possibilità di modificare le impostazioni di gioco

1.9 CARATTERISTICHE DELL'UTENTE

L'utilizzo di questa applicazione è indicato a tutte quegli utenti che vogliono intrattenersi giocando ad un grande classico dei videogiochi. L'interfaccia utente è ispirata ai classici giochi incorporati nelle distribuzioni Windows.

1.10 VINCOLI

Il sistema di gioco è limitato alla semplice pressione dei tasti direzionali che andranno a muovere sull'asse orizzontale il paddle per colpire la pallina.

La classifica verrà salvata in un semplice file di testo dato che è un gioco e non ha bisogno di una sicurezza superiore in questo campo.

1.11 ASSUNZIONI E DIPENDENZE

Nel progetto viene utilizzato il framework Java Swing e la libreria Java AWT.

1.12 REQUISITI NON FUNZIONALI

Come requisiti non funzionali sono stati presi in considerazione i seguenti:

- prestazioni
- manutenibilità
- usabilità
- efficienza
- sicurezza

- affidabilità
- disponibilità

1.13 REQUISITI SPECIFICI

L'interfaccia utente è gestita tramite una serie di Jpanel di Java, che si alternano a secondo dell'azione svolta dall'utente, i quali sono contenuti all'interno di un Jcontainer. Sarà necessario quindi poter avviare l'applicazione tramite un ambiente di sviluppo per il linguaggio Java.

La priorità dei requisiti è stata realizzata applicando il modello Moscow, indicando la priorità tra le parentesi:

- la fisica di gioco, definibile come il cuore fondante dell'applicazione, definisce i movimenti che la pallina può eseguire (must have)
- l'interfaccia grafica è gestita tramite una serie di Jpanel di Java, che si alternano a secondo dell'azione svolta dall'utente, i quali sono contenuti all'interno di un Jcontainer (must have)
- l'interfaccia dei pulsanti fisici, necessaria per poter verificare la pressione di un determinato tasto (should have)
- la gestione dei punteggi, necessaria per capire quanti punti si sono raccolti durante una partita (could have)
- la lettura e la scrittura dei file, utile per tenere un registro locale con i risultati delle partite (could have)
- i potenziamenti, utili per diversificare lo stile di gioco di una partita (could have)

Il codice lo si può dividere in tre tipologie: una classe Main, Container, che serve ad avviare le interfacce e il motore del gioco; delle classi sviluppate tramite Swing per le interfacce; delle classi secondarie utilizzate dalla classe motore di gioco, Gamecore, e dall'interfaccia di gioco.

ARCHITETTURA

In successione all'analisi dei requisiti vi è l'analisi dell'architettura e del design. La rappresentazione più conforme all'architettura dell'applicativo vengono descritte tramite degli schemi UML. Questi possono essere caratterizzati come delle Viste indirizzate a diversi Stakeholder e definite tramite diversi Viewpoint, come stabilito da IEEE1471. A continuazione si farà riferimento a diversi diagrammi UML, questi sono resi disponibili nel corrispettivo documento o nel file MDJ.

I casi d'uso sono utili per questo tipo di applicazione. In questo caso rappresenta una possibile partita. e stato rappresentato tramite Use Case Diagram su UML.

PHYSICAL VIEW

La caratteristica fondamentale dell'applicazione è sicuramente la sua portabilità, ovvero la capacità di adattarsi a qualsiasi tipo di ambiente fisico e virtuale. Per quanto riguarda la parte fisica, infatti, non sono necessari specifici computer dotati di hardware potente, basta un sistema computazionale in grado di eseguire l'ambiente java.

DEVELOPMENT VIEW

Per quanto riguarda la struttura base del software e stato deciso di suddividere le classi in tre tipologie: una classe Main, Container, che serve ad avviare le interfacce e il motore del gioco; delle classi sviluppate tramite Swing per le interfacce; delle classi secondarie utilizzate dalla classe motore di gioco, Gamecore, e dall'interfaccia di gioco.

PROCESS VIEW

In questa sezione è stato analizzato e studiato tutti i possibili processi presenti nel nostro software. Da questo studio siamo riusciti a sviluppare un sistema in grado di raggiungere performance di un certo livello con una tolleranza agli errori abbastanza ridotta. Tutto ciò è stato favorito la realizzazione di due diagrammi UML (Sequence Diagram e Activity Diagram) che descrivono tutte le possibili comunicazioni tra i vari processi che compongono l'applicazione.

Se si segue un modello architetturale basato sul classico model-view-controller, in cui il controller è l'utente, il componente centrale del MVC, il modello, cattura il comportamento dell'applicazione in termini di dominio del problema indipendentemente dall'interfaccia utente, quindi nel nostro caso sarebbe la fisica di gioco e una vista può essere una qualsiasi rappresentazione in output di informazioni, come l'interfaccia utente e di gioco .

TEST DEL SOFTWARE

4.1 IEEE928











Questo tipo di documento si occupa di specificare e documentare come avviene la fase di testing del progetto. Questo documento rappresenta anche il test plan. Siccome stiamo facendo un tipo di sviluppo di software AGILE i test non vengono fatti alla fine ma vengono svolti assieme allo sviluppo delle classi. Viene svolto un tipo di sviluppo di tipo TDD in cui si scrivono le classi dei test in contemporanea (se non prima) delle classi che servono per l'applicazione. Le fasi del testing sono le seguenti:

- Preparation of tests
- Running the tests
- Completion of testing

4.2 PREPARAZIONE DEI TEST

I test vengono fatti in contemporanea allo sviluppo delle classi, talvolta per alcune classi sono stati fatti prima i test pensando poi alle classi che sarebbero state implementate successivamente. Gli unit test per le classi sono scritti anche loro in linguaggio Java ed eseguiti tramite Junit. Per favorire l'integrazione continua i test saranno eseguiti ogni qualvolta si effettuerà un git. I test tenderanno ad essere dei unit test, dato che andiamo a testare le funzionalità delle singole classi

In seguito vi è riportata la copertura dei test.

src		68,6 %	1.881	860	2.741
> helpframe.java		87,2 %	170	25	195
> Container.java		81,2 %	104	24	128
> mainmenu.java		85,0 %	130	23	153
> gamepanel.java		98,8 %	255	3	258
> Paddle.java		94,9 %	56	3	59
> Ball.java		100,0 %	52	0	52
> Brick.java		100,0 %	25	0	25
> classifica.java		100,0 %	28	0	28
> MyButton.java		100,0 %	47	0	47

La copertura dei test non risulta completa perchè la parte grafica dell'applicativo è stata testata manualmente.

QUALITÀ DEL SOFTWARE

5.0.1 CMM

Per quanto riguarda il CMM (Capability Maturity Model) esistono 5 livelli di maturità del software, essi sono:

- Iniziale
- Ripetibile
- Definito
- Gestito quantitativamente
- Ottimizzato

In questo progetto si punta ad utilizzare il terzo livello 3 (definito) in cui ogni attività viene documentata e

standardizzata per l'intera organizzazione per quanto riguarda il processo per il design, development, testing e via dicendo.

5.0.2 ISO 9216

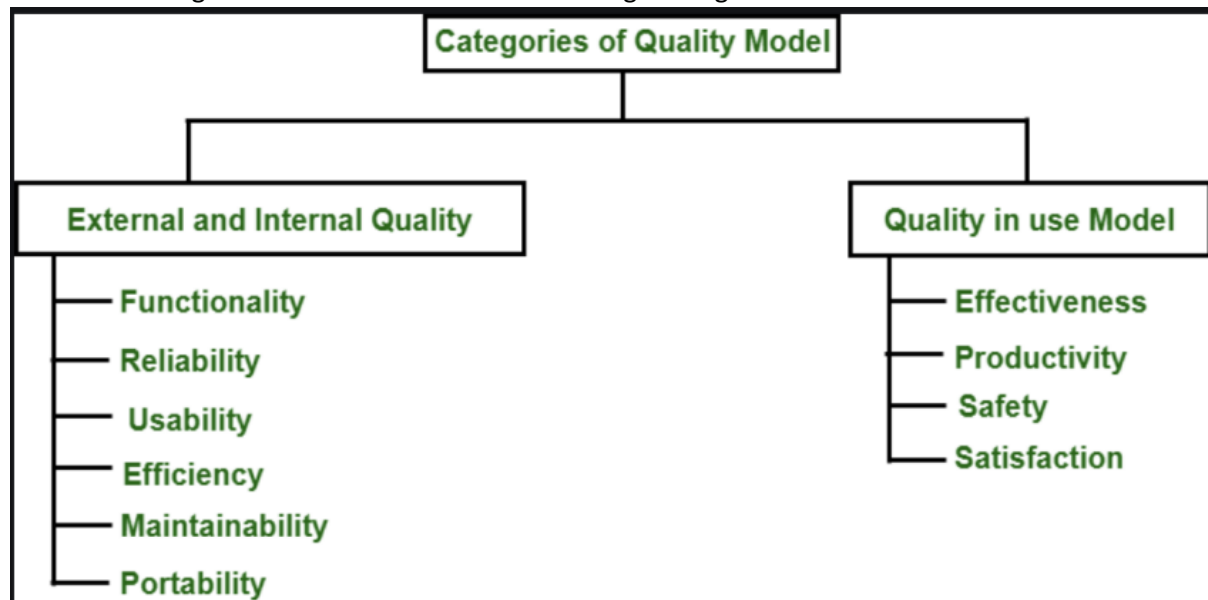
Per quanto riguarda la product conformance si fa riferimento al seguente link:

<https://www.geeksforgeeks.org/iso-iec-9126-in-software-engineering/>

Esso si basa sui 4 seguenti principi:

- Part 1: "Quality model"
- Part 2: "External metrics"
- Part 3: "Internal metrics"
- Part 4: "Quality in use metrics"

Per chiarire meglio il concetto si fa riferimento al seguente grafico:



5.1 QUALITÀ INTERNE ED ESTERNE

Durante la fase di sviluppo del software il focus principale rimane sempre il mantenimento della qualità. Per fare tutto questo bisogna tenere conto delle seguenti 6 caratteristiche:

- funzionalità
- affidabilità
- usabilità
- efficienza
- manutenibilità
- portabilità

5.1.1 FUNZIONALITÀ

Il codice si occupa di mantenere il giusto grado di sicurezza dei dati e deve essere adeguato e facile da utilizzare per gli utenti.

5.1.2 AFFIDABILITÀ

Il codice si occupa della gestione dei guasti e punta ad aver il minor numero possibile di errori.

5.1.3 USABILITÀ

Il prodotto finale non deve essere difficile da utilizzare e la user experience deve essere il più semplice possibile per l'utente in quanto non deve essere necessariamente esperto di coding. Ad esempio i bottoni sono studiati per essere il più semplici e intuitivi possibile. Dal punto di vista delle prestazioni deve essere il più reattivo possibile e facile da capire.

5.1.4 EFFICIENZA

Si vuole creare il programma più efficiente possibile che utilizzi il minimo delle risorse del dispositivo computazionale da cui si utilizza l'applicazione.

5.1.5 MANUTENIBILITÀ

Il codice deve essere scritto nel modo più conforme alle regole di standard di programmazione per favorire la leggibilità e la successiva manutenzione.

5.1.6 PORTABILITÀ

Il programma deve funzionare su più piattaforme possibili.

5.2 MODELLO DI QUALITÀ IN USO

Si basa sulle seguenti 4 caratteristiche che si vuole soddisfare:

- portabilità
- efficienza
- usabilità

5.3 INTRODUZIONE ALLA QUALITÀ

Questo file si occupa di descrivere i requisiti tassonomici di qualità del progetto. Per quanto riguarda la definizione di McCall. In particolare si vuole utilizzare le seguenti linee guida per la produzione, revisione e transizione del codice.

5.4 FUNZIONAMENTO DEL PRODOTTO

- correttezza: se il sistema fa quello richiesto
- affidabilità: se il sistema è abbastanza accurato
- efficienza: se il sistema utilizza l'hardware efficientemente
- integrità: se il sistema è sicuro
- usabilità: se il sistema è utilizzabile

In particolare ci si sofferma sull'usabilità, correttezza e affidabilità dato che il prodotto deve funzionare il meglio possibile.

5.5 REVISIONE DEL PRODOTTO

- manutenibilità: se il sistema in caso di guasto è riparabile

- testabilità: se il sistema è testabile
- flessibilità: se il sistema è facilmente cambiabile

Il prodotto che si sta costruendo in questo caso tende a essere molto testabile in quanto i test per verificare la correttezza vengono creati ed eseguiti poco dopo la creazione delle classi.

5.6 TRANSIZIONE DEL PRODOTTO

- portabilità: se il software è utilizzabile su altre piattaforme
- riusabilità: se il software è riutilizzabile
- interoperabilità: se il sistema è interfacciabile con altri sistemi

Il sistema risulterà molto portabile in quanto è stato realizzato in Java, che tende ad essere molto scalabile.

5.7 QUALITÀ PER IL CODICE JAVA

Per favorire uno sviluppo di qualità del codice ogni classe è stata strutturata in modo che rappresenta una delle tante schermate che si presentano al cliente. Nel caso si voglia apportare una modifica basterà lavorare su quella specifica parte senza preoccuparsi troppo del resto del codice.

Inoltre, per valutare la complessità del software sono stati presi in considerazione i seguenti indici:

- number of lines of code
- weighted methods per class
- coupling between object classes

MANUTENZIONE

Siccome non si sapeva utilizzare il framework Java Swing e la libreria java AWT si sono alternati momenti di forwarding engineering con momenti di reverse engineering in cui si è sistemata la documentazione in funzione del codice che è stato scritto (reducomentation) sviluppando così un processo di round trip engineering. Il refactoring è stato fatto alla fine per separare le classi delle interfacce dalla classe che le gestisce. Inoltre a fine progetto sono state rimosse le classi e i metodi non utilizzati (dead code).

SOFTWARE DESIGN

La progettazione del software riguarda la scomposizione di un sistema nelle sue parti fondamentali, le quali hanno una complessità minore del sistema completo, mentre le parti insieme risolvono il problema dell'utente. Le caratteristiche di design che ci interessano di più sono quelle che facilitano la manutenzione e il riutilizzo: semplicità, una netta separazione dei concetti in diversi moduli e una visibilità limitata delle informazioni.

Per informazione sulla complessità e altri criteri di qualità, fare riferimento al documento della qualità.

7.1 FUNZIONAMENTO DELLE CLASSI JAVA CONFORME CON REQUISITI FUNZIONALI E NON:

Il design delle classi Java è stato guidato dalla necessità di gestire dati (la fisica di gioco) che non sono disponibili tramite interfaccia utente, mentre altri lo sono. Dal punto di vista dell'utente ciò deve essere invisibile usando se necessario delle schermate di caricamento.

7.2 UTILIZZO DI DESIGN PATTERN IN JAVA:

- Observer: la dinamica Observer-Observable permette di definire una dipendenza uno a molti fra oggetti, in modo tale che se un oggetto cambia il suo stato interno, ciascuno degli oggetti dipendenti da esso viene notificato e aggiornato automaticamente. L'Observer nasce dall'esigenza di mantenere un alto livello di consistenza fra classi correlate, senza produrre situazioni di forte dipendenza e di accoppiamento elevato.

SOFTWARE LIFE CYCLE

In questo progetto è stato utilizzato un life cycle agile, dato che il progetto non è di dimensioni elevate. Inoltre, dati i requisiti del progetto, già definiti in anticipo, ma che si potrebbero modificare piano piano, è stato scelto il modello di sviluppo incrementale. Le funzionalità del sistema sono prodotte e consegnate al cliente in piccoli incrementi e miglioramenti, in questo modo, viene messa maggior attenzione sulle feature essenziali. Teoricamente, questo modello è simile al prototyping: si potrebbe infatti fornire un prototipo al cliente ogni qual volta un incremento ne consenta la creazione.

8.1 STATE OF BACKLOG

Il backlog iniziale si basa sui requisiti disponibili, nel rispettivo documento, nella fase di architettura. Il backlog tiene in considerazione tali requisiti, ma inoltre è definito dal contesto architettonico.

-Milestone 1

- approfondimento della fisica di gioco
- stesura della documentazione necessaria
- design, testing e implementazione della fisica

-Milestone 2

- informarsi sul funzionamento di Java Swing e Java AWT
- creazione della UI
- proseguire con la documentazione

-Milestone 3

- perfezionamento dell'interfaccia utente
- iniziare il testing dell'interfaccia utente
- proseguire con la documentazione