uniPAC-MAN

documentatione

**realizzato da**

**bellosi jacopo ( m..1081058 )**

**longhi lara ( m. 1079261 )**

**poloni Lluca ( m. 1078817 )**

*Indice*

[1. Requirements specification (Standard IEEE 830) 3](#_Toc154826105)

[1.1 Introduzione 3](#_Toc154826106)

[1.1.1 Obiettivo 3](#_Toc154826107)

[1.1.2 Scopo 3](#_Toc154826108)

[1.1.3 Definizioni, acronimi e abbreviazioni 3](#_Toc154826109)

[1.2 Descrizione generale 3](#_Toc154826110)

[1.2.1 Prospettiva del prodotto 3](#_Toc154826111)

[1.2.2 Funzioni del prodotto 3](#_Toc154826112)

[1.2.3 Caratteristiche dell’utente 3](#_Toc154826113)

[1.2.4 Vincoli 3](#_Toc154826114)

[1.2.5 Presupposti e dipendenze 3](#_Toc154826115)

[1.3 Requisiti specifici 3](#_Toc154826116)

[1.3.1 Requisiti dell’interfaccia esterna 3](#_Toc154826117)

[1.3.2 Richieste funzionali 4](#_Toc154826118)

[1.3.3 Requisiti di prestazione 4](#_Toc154826119)

[1.3.4 Vincoli di progettazione 4](#_Toc154826120)

[2. Software lifecycle 4](#_Toc154826121)

[3. Configuration management 5](#_Toc154826122)

[4. People management 5](#_Toc154826123)

[5. Software quality 5](#_Toc154826124)

[6. Requirement engineering 6](#_Toc154826125)

[7. Modeling 7](#_Toc154826126)

[7.1 Il diagramma delle classi 7](#_Toc154826127)

[7.2 Il diagramma della macchina a stati 8](#_Toc154826128)

[7.3 Il diagramma di sequenza 8](#_Toc154826129)

[7.4 Il diagramma di comunicazione 8](#_Toc154826130)

[7.5 Il diagramma dei componenti 8](#_Toc154826131)

[8. Software architecture 8](#_Toc154826132)

[9. Software design 8](#_Toc154826133)

[10. Software testing 8](#_Toc154826134)

[11. Software maintenance 8](#_Toc154826135)

1. Requirements specification (Standard IEEE 830)

Prima di procedere con lo sviluppo del prodotto software, è necessario esplicitare i requisiti alla base del progetto. La specifica è strutturata secondo lo standard IEEE830.

* 1. *Introduzione*

Il progetto consiste nello sviluppare un’applicazione Java per Windows che consenta all’utente di giocare a una versione rivisitata del gioco Pac-Man, la rivisitazione consiste in variazioni grafiche che rappresentano il tema universitario.

* + 1. Obiettivo

L’ obiettivo principale del gioco Pac-Man è mangiare tutti i CFU sparsi nel labirinto. Nel far ciò c’è la possibilità di incombere in quattro fantasmi che cercano di catturare Pac-Man. Quando Pac-Man mangia una power-up, i fantasmi diventano vulnerabili e possono essere mangiati da Pac-Man per un breve periodo di tempo. Il gioco continua finché Pac-Man riesce a mangiare tutti i punti nel labirinto o fino a quando perde tutte le vite disponibili.

Il gioco è strutturato su tre livelli che rappresentano gli anni necessari per il raggiungimento di una laurea triennale.

### *Scopo*

Lo scopo del gioco è ottenere il punteggio più alto possibile mangiando CFU, power-up e fantasmi. Ogni CFU mangiato contribuisce al punteggio del giocatore. Oltre ai punti normali, ci sono anche punti bonus ottenuti mangiando i power-up e a seguito i fantasmi.

* + 1. Definizioni, acronimi e abbreviazioni
* CFU: Crediti formativi universitari, rappresentati dal simbolico pallino verde presente all’interno del libretto elettronico una volta superato l’esame;
* Power-up: oggetto che conferisce una particolare abilità temporanea, nel nostro specifico caso è rappresentato dallo youtuber Elia Bombardelli, conosciuto per il suo canale YT riguardante spiegazioni di argomenti di matematica.

## *Descrizione generale*

### *Prospettiva del prodotto*

### *Funzioni del prodotto*

### *Caratteristiche dell’utente*

L’applicazione è facile e intuita, quindi non necessita di preparazione pregressa, qualunque tipo di utente può giocarci e comprendere velocemente le funzionalità del gioco.

### *Vincoli*

### *Presupposti e dipendenze*

## *Requisiti specifici*

### *Requisiti dell’interfaccia esterna*

L’interfaccia utente prevede una schermata iniziale dove l’utente potrà decidere se iniziare una nuova partita o uscire dalla schermata. A seguito della pressione del tasto *enter* in relazione a “NUOVA PARTITA” apparirà la schermata di gioco dove l’utente potrà muovere il Pac-Man con AWSD.

Immagine che contiene testo, software, schermata, Software multimediale

Descrizione generata automaticamenteImmagine che contiene testo, computer, schermata, software

Descrizione generata automaticamenteImmagine che contiene testo, computer, software, schermata

Descrizione generata automaticamenteImmagine che contiene testo, computer, software, schermata

Descrizione generata automaticamente

### *Richieste funzionali*

### *Requisiti di prestazione*

Non sono richieste risorse computazionali elevate per questo applicativo, non necessità nemmeno di una connessione internet.

### *Vincoli di progettazione*

1. Software lifecycle

Il modello di processo scelto per lo sviluppo dell’applicativo è un metodo agile nello specifico eXtreme Programming. La scelta di un process model di tipo AGILE ci permette di non preoccuparci troppo dei tempi ma di adattarci ai cambiamenti e agli sviluppi che notiamo necessari in corso d’opera.

Nell’ambito specifico dell’ eXtreme Programming (XP) troviamo l’utilizzo in forte misura di diverse pratiche:

* *Whole team*, tutto il gruppo è stato coinvolto in tutte le fasi del lavoro;
* *Cliente in loco*, la stessa squadra ha agito pure da cliente testando di persona il codice;
* *Pair programming*, il codice spesso verrà scritto su una sola macchina con supporto da remoto di un altro programmatore. Questo garantisce una diminuzione del tempo di scrittura e una bontà e ottimizzazione maggiori. Quest’approccio è stato utilizzato per quasi tutta l’interezza del progetto;
* *Proprietà collettiva*, tutti hanno accesso a tutto e possono modificarlo in qualsiasi momento;
* *Sviluppo guidato dei test*, ogni nuova implementazione delle funzionalità/dinamiche di gioco è stata seguita da diversi test effettuati dagli stessi programmatori o amici.

1. Configuration management

Come strumento per la gestione della configurazione abbiamo utilizzato GitHub, il quale ci ha permesso di tenere traccia di tutti gli artefatti realizzati, e delle modifiche effettuate dagli altri componenti del gruppo. Ogni commit con annessa descrizione della modifica/progresso rappresenta i cambiamenti eseguiti nel codice, visibili da tutto il team per avere sempre a disposizione la versione aggiornata.

1. People management

Essendo un’ambiente prettamente dedicato allo sviluppo software, possiamo individuare un’organizzazione a matrice dove, singolarmente o con supporto da parte di un altro gruppo abbiamo elaborato le varie funzionalità.

|  |  |  |  |
| --- | --- | --- | --- |
| *Funzioni\Persone* | Bellosi | Longhi | Poloni |
| Costruzione Panel | x | x |  |
| Grafica |  | x | x |
| Pac-Man | x |  |  |
| Fantasmi | x |  |  |
| Arena |  |  | x |
| Tunnel |  | x |  |
| Vite | x | x |  |
| Stati di gioco |  | x | x |
| Power-up |  |  | x |
| Score Manager | x |  |  |
| Suoni | x | x | x |
| Livelli |  |  | x |

1. Software quality

Nel glossario IEEE sappiamo che la qualità è il grado in cui un sistema soddisfa le esigenze, per esplicare il nostro grado di qualità usiamo i fattori predisposti dalla tassonomia di McCall:

1. Funzionamento del prodotto:
   1. Correttezza: il prodotto funziona, è usabile e giocabile alla pari della versione ufficiale di pac-man ufficiale;
   2. Affidabilità: il prodotto sotto alcune caratteristiche presenta dei bug, che si presentano raramente e non in tutte le fasi di test; quindi, risulta difficile riconoscerli.
   3. Efficienza: l’applicativo non è ottimizzato al suo livello assoluto, quindi, potrebbe non funzionare nel miglior modo possibile su ogni hardware;
   4. Integrità: il prodotto è sicuro in quanto l’acceso al software viene effettuato solo a seguito di una condivisione dell’eseguibile generato direttamente dai programmatori;
   5. Usabilità: è eseguibile se si è installato Java sulla macchina.
2. Revisione del prodotto
   1. Manutenibilità: si possono apportare modifiche in qualsiasi momento e il codice è stato progettato per essere il più scalabile possibile;
   2. Testabilità: il prodotto è sempre testabile;
   3. Flessibilità: il software reagisce prontamente a cambiamenti del codice, si possono tranquillamente implementare funzionalità aggiuntive.
3. Transizione del prodotto
   1. Portabilità: l’applicabile è usabile su qualunque macchina abbia installato l’ultima versione di Java;
   2. Riutilizzabilità: l’intero codice può essere modificato anche da altri sviluppatori i quali possono implementare nuove funzionalità o utilizzare parti di codice come spunto per altre soluzioni simili (es. le collisioni in qualsiasi gioco possono essere implementate secondo la nostra logica)
   3. Interoperabilità: essendo un semplice eseguibile .jar non sono necessarie ulteriori installazioni né determinati requisiti a livello di OS né API.
4. Requirement engineering

Per la gestione del processo di sviluppo abbiamo analizzato i requisiti secondo il criterio MoSCoW:

* Must have:
  + Sviluppo di un applicativo funzionante;
  + Interfaccia grafica semplice, intuitiva e familiare;
  + Portabile e quindi usabile da più utenti.
* Should have:
  + Movimento intelligente dei fantasmi secondo le dinamiche del gioco originale e non randomico;
  + Diversi livelli aggiuntivi con progressiva difficoltà;
  + Suoni.
* Could have:
  + Storico delle partite;
  + Classifica in base ai punteggi.
* Won’t have:
  + Funzionalità grafiche aggiuntive;
  + Multiplayer

1. Modeling

## *7.1 Il diagramma delle classi*

Di seguito il diagramma delle classi, abbiamo utilizzato principalmente associazioni 1-1, inoltre per evidenziare meglio le generalizzazioni abbiamo scelto di utilizzare due colori:

* Il blu per le estensioni di Entity;
* Il verde per le estensioni di GameObject.

Immagine che contiene testo, schermata, Rettangolo, quadrato

Descrizione generata automaticamente

## *7.2 Il diagramma della macchina a stati*

## *7.3 Il diagramma di sequenza*

Essendo un programma molto complesso risulta dispendioso realizzare un diagramma di sequenza per tutte le classi; perciò, abbiamo deciso di realizzare un diagramma solo per le classi principali che attivano il maggior numero di classi, di seguito il diagramma di sequenza che studia la classe UniPacman e GameEngine.

Immagine che contiene testo, diagramma, Parallelo, Disegno tecnico

Descrizione generata automaticamente

Immagine che contiene testo, schizzo, Parallelo, diagramma

Descrizione generata automaticamente

Immagine che contiene testo, diagramma, Parallelo, Disegno tecnico

Descrizione generata automaticamente

Immagine che contiene testo, diagramma, Parallelo, linea

Descrizione generata automaticamente

## *7.4 Il diagramma delle attività*

## *7.5 Il diagramma dei casi d’uso*

1. Software architecture

1. Software design
2. Software testing
3. Software maintenance

È stato utilizzato il processo di *refactoring* per garantire un codice più ottimizzato, leggibile, poco complesso e a basso costo di manutenzione:

* Estrazione di Metodo: creare nuovi metodi per frammenti di codice ripetitivi.
* Rinominazione: Assegnare nomi più significativi a variabili, metodi o classi.
* Scomposizione di Metodo: Suddividere metodi complessi in passi più piccoli.
* Sostituzione dei Parametri con il Metodo: Ridurre la dipendenza dai parametri passati a un metodo.

Nel corso della scrittura del codice si sono eseguiti parallelamente i test del refactoring per garantire un codice sempre funzionante, privo di errori ma ottimizzato.

Sono stati utilizzati strumenti di refactoring come l’ambiente di sviluppo Eclipse col quale è stato possibile suddividere le classi all’interno di package coerenti al ruolo della classe stessa, ma anche rinominare metodi, svolgere l’inlining, rinomare metodi con nomi appropriati.

La scelta di svolgere il refactoring man mano che si stesse scrivendo il codice ci ha garantito una maggiore chiarezza e pulizia, andando a preferire un approccio incrementale all’ottimizzazione del software grazie a piccole e mirate modifiche facendo sempre rimanere il codice funzionante.

È stato utilizzato anche il plugin Eclipse UCDetector (Usage Counter Detector).

È uno strumento utilizzato nella programmazione Java per individuare l'uso non dichiarato di classi e metodi in un progetto, aiuta quindi a identificare le classi e i metodi che non sono più utilizzati nel codice sorgente dell’applicazione. Tramite UCDetector sono stati eliminati dal codice metodi e variabili mai utilizzate.

Fai esempi pratici di pulizia es:   
 Metodo lungo

1. Classe numerosa
2. Ossessione primitiva
3. Gruppi di dati
4. Scambia dichiarazioni
5. Classe pigra
6. Codice duplicato
7. Caratteristica invidia
8. Intimità inappropriata

Ovviamente sono stati utilizzati alcuni strumenti di manutenzione del software come Eclipse che ci ha permesso di rinominare e distribuire le classi in diversi pacchetti, in modo da aumentare la chiarezza del codice.  
Scrivi altro in base:  
Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

VEDI CODART