UNIPAC-MAN

DOCUMENTATIONE

REALIZZATO DA

BELLOSI JACOPO (M. 1081058)
LONGHI LARA (M. 1079261)
POLONI LUCA (M. 1078817)

Indice

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.

1.1 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 a tema universitario.

1.1.1 Obiettivo

L' obiettivo principale del gioco UniPac-Man è mangiare tutti i CFU sparsi nel labirinto. Nel far ciò c'è la possibilità di incombere nei quattro fantasmi della procrastinazione 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.

1.1.2 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.

1.1.3 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 dal professore e youtuber Elia Bombardelli, noto fra i più giovani per fornire spiegazioni semplici e chiare a problemi di matematica sul suo canale YouTube.
- Fantasmi della procrastinazione: "nemici" dello studente, rappresentano tentazioni come lo smartphone, la musica, il sonno e il caffè.

1.2 Descrizione generale

1.2.1 Prospettiva del prodotto

UniPac-Man è la rivisitazione del celebre gioco prodotto dalla Namco nel 1980, è possibile usufruire del gioco in locale tramite qualsiasi computer con l'ultima versione di Java installato, attualmente non sono presenti piattaforme online e multiplayer.

1.2.2 Funzioni del prodotto

UniPac-man avrà le seguenti funzionalità:

- *Movimento del Personaggio:* Il giocatore controlla il personaggio di Pacman attraverso un labirinto, utilizzando i tasti WASD per muoversi in su, giù, a sinistra o a destra.
- *Raccolta di CFU*: Pacman deve mangiare tutti i CFU disseminati nel labirinto per completare il livello.
- *Evitare i Fantasmi della procrastinazione:* Il gioco presenta dei fantasmi che inseguono Pacman nel tentativo di catturarlo. Il giocatore deve evitare i fattori della procrastinazione o utilizzare il power-up "*Elia Bombardelli*" per invertire la situazione e assicurarsi la vittoria.

- *Power-Up*: Pacman può consumare i power-up speciali che invertiranno temporaneamente il ruolo dei fantasmi, consentendo a Pacman di inseguirli e mangiandoli guadagnare CFU e tempo extra.
- *Visualizzazioni dei Punteggi e dello Stato di Gioco*: Il gioco fornisce informazioni sul punteggio del giocatore e il numero di vite rimaste. La transizione ai vari livelli è gestita da una schermata che indica il superamento dell'anno corrente, una volta completati i tre anni si è finalmente laureati!
- *Sistema di Vite:* Il giocatore ha un numero limitato di vite (tre) per completare il livello.
- *Menu di Avvio e Opzioni:* Il gioco include un menu di avvio con opzioni per iniziare un nuovo gioco o uscire, sarà possibile mettere il gioco in pausa in qualsiasi momento premendo il tasto P.

1.2.3 Caratteristiche dell'utente

Il gioco è facile e intuitivo, trattandosi un gioco di tipo arcade non è richiesta una grande conoscenza o padronanza di qualche skill specifica. Qualunque tipo di utente può giocarci e comprendere velocemente le funzionalità del gioco.

1.2.4 Vincoli

Per giocare a UniPac-Man bisogna aver installato JDK (versione 21 o recenti), indipendentemente dal sistema operativo utilizzato.

1.2.5 Presupposti e dipendenze

Si presume che il gioco "Pacman" sarà eseguito in un ambiente di gioco standard, senza interferenze o problemi esterni significativi.
Si assume che gli utenti abbiano una familiarità di base con l'uso di tastiere, controller o touchscreen, a seconda della piattaforma e che sia disponibile uno spazio d'archiviazione sufficiente per l'installazione del gioco.
Il corretto funzionamento del gioco dipende dalla presenza di risorse grafiche e sonore, come sprite, suoni e file musicali, nel sistema di gioco. È disponibile solamente la versione del gioco in italiano.

1.3 Requisiti specifici

1.3.1 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 i tasti WASD e mettere il gioco in pausa col tasto P. Premendo il tasto *enter* in relazione a "ESCI" si uscirà direttamente dall'applicativo. Quando si passa da un livello all'altro compare una schermata informativa, è possibile giocare direttamente al livello successivo premendo il tasto *space*. Al termine del terzo livello e al conseguimento della laurea è possibile tornare alla schermata iniziale di gioco tramite il tasto *space*.

1.3.2 Richieste funzionali

Il sistema deve garantire un tempo di risposta rapido ai comandi del giocatore, assicurando un'esperienza di gioco fluida e reattiva, gestendo in maniera ottimale anche le collisioni all'interno delle varie dinamiche di gioco. Deve fornire feedback audiovisivo immediato, per migliorare l'immersione e la comprensione degli eventi di gioco tramite animazioni e suoni personalizzati.

1.3.3 Requisiti di prestazione

Non sono richieste risorse computazionali elevate per questo applicativo, non necessita di connessione internet.

2. 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.

3. 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 tutta la squadra per avere sempre a disposizione la versione aggiornata.

4. People management

Il gruppo di lavoro è formato da solamente tre persone, questo aumenta l'efficienza della comunicazione ma garantisce un bilanciamento dei ruoli. Abbiamo adattato i ruoli e i compiti in base alle conoscenze pregresse e alla motivazione di ogni singolo componente. Essendo un'ambiente prettamente dedicato allo sviluppo software, possiamo individuare un'organizzazione a matrice dove, singolarmente o con supporto di un'altra persona abbiamo elaborato le varie funzionalità.

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

5. 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:

A) Funzionamento del prodotto:
   a. Correttezza: il prodotto funziona, è usabile e giocabile alla pari della versione ufficiale di pac-man ufficiale;
   b. 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.
   c. Efficienza: l'applicativo non è ottimizzato al suo livello assoluto, quindi, potrebbe non funzionare nel miglior modo possibile su ogni hardware;
   d. 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;
   e. Usabilità: è eseguibile se si è installato Java sulla macchina.

B) Revisione del prodotto
   a. Manutenibilità: si possono apportare modifiche in qualsiasi momento e il codice è stato progettato per essere il più scalabile possibile;
   b. Testabilità: il prodotto è sempre testabile;

c. Flessibilità: il software reagisce prontamente a cambiamenti del codice, si possono tranquillamente implementare funzionalità aggiuntive.

C) Transizione del prodotto
   a. Portabilità: l'applicabile è usabile su qualunque macchina abbia installato l'ultima versione di Java;
   b. 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)
   c. Interoperabilità: essendo un semplice eseguibile .jar non sono necessarie ulteriori installazioni né determinati requisiti a livello di OS né API.

6. Requirement engineering

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

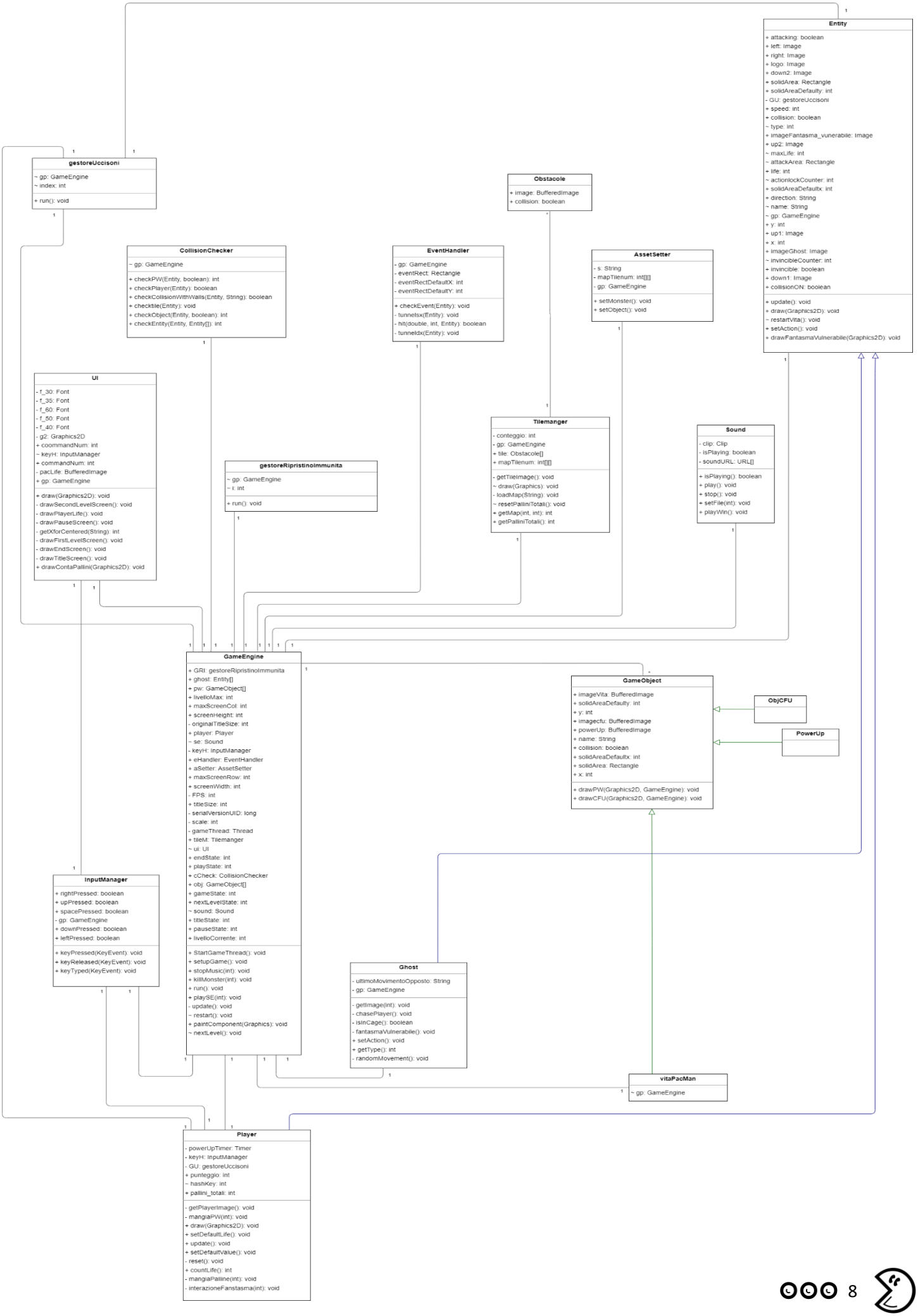
- Must have:
  - Sviluppo di un applicativo funzionante;
  - Interfaccia grafica semplice, intuitiva e familiare;
  - Movimento fluido dei fantasmi della procrastinazione;
  - Portabilità.
- Should have:
  - Diversi livelli aggiuntivi con progressiva difficoltà;
  - Suoni.
- Could have:
  - Storico delle partite;
  - Frutta sottoforma di appunti e libri di testo per aumentare il punteggio;
  - Classifica in base ai punteggi;
  - Movimento intelligente dei fantasmi secondo le dinamiche del gioco originale e non randomico.
- Won't have:
  - Funzionalità grafiche aggiuntive;
  - Multiplayer

7. Modeling

7.1 Il diagramma delle classi

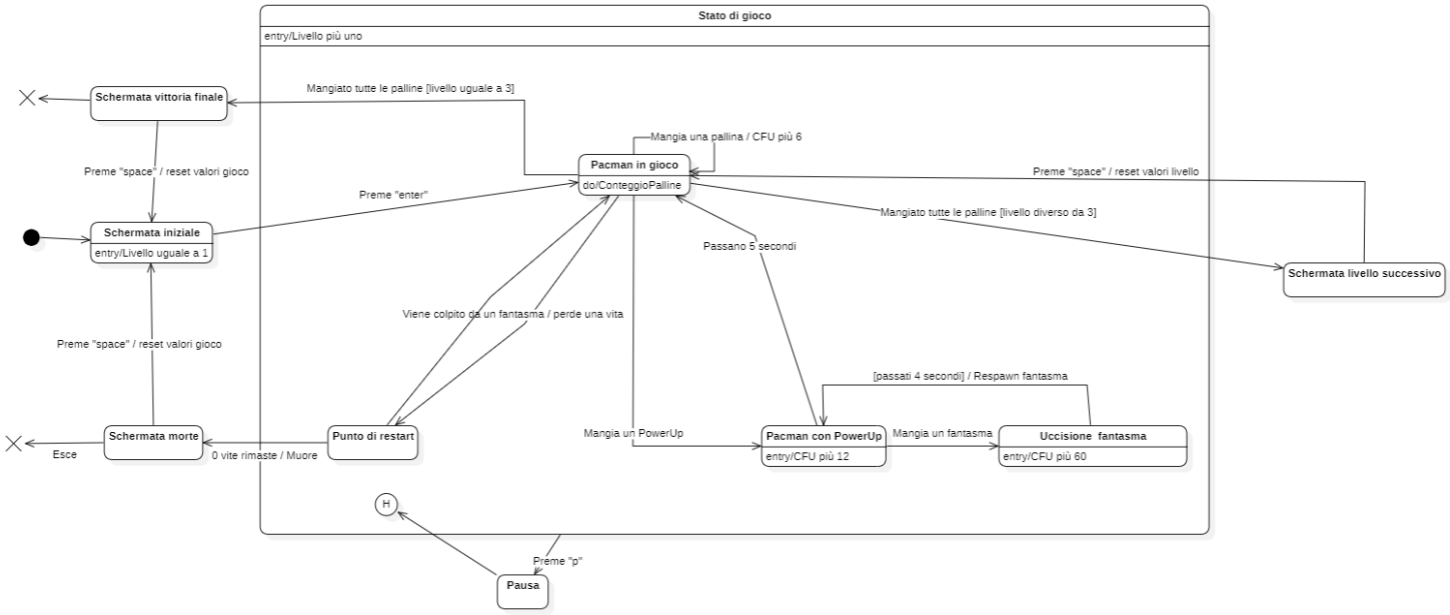
Di seguito il diagramma delle classi, abbiamo utilizzato principalmente associazioni 1-1, inoltre per evidenziare meglio le generalizzazioni abbiamo scelto di utilizzare due colori:

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


Entity
+ attacking: boolean
+ left: image
+ right: Image
+ logo: Image
+ down2: Image
+ solidArea: Rectangle
+ solidAreaDefaulty: int
- GU: gestoreUccisoni
+ speed: int
+ collision: boolean
~ type: int
+ imageFantasma_vunerabile: Image
+ up2: Image
~ maxLife: int
~ attackArea: Rectangle
+ life: int
~ actionlockCounter: int
+ solidAreaDefaultx: int
+ direction: String
~ name: String
~ gp: GameEngine
+ y: int
+ up1: Image
+ x: int
+ imageGhost: Image
~ invincibleCounter: int
+ invincible: boolean
+ down1: Image
+ collisionON: boolean
+ update(): void
+ draw(Graphics2D): void
~ restartVita(): void
+ setAction(): void
+ drawFantasmaVulnerabile(Graphics2D): void
gestoreUccisoni
~ gp: GameEngine
~ index: int
+ run(): void
Obstacole
+ image: BufferedImage
+ collision: boolean
CollisionChecker
~ gp: GameEngine
+ checkPW(Entity, boolean): int
+ checkPlayer(Entity): boolean
+ checkCollisionWithWalls(Entity, String): boolean
+ checktile(Entity): void
+ checkObject(Entity, boolean): int
+ checkEntity(Entity, Entity[]): int
EventHandler
- gp: GameEngine
- eventRect: Rectangle
- eventRectDefaultX: int
- eventRectDefaultY: int
+ checkEvent(Entity): void
- tunnelsx(Entity): void
- hit(double, int, Entity): boolean
- tunneldx(Entity): void
AssetSetter
- s: String
- mapTilenum: int[][]
- gp: GameEngine
+ setMonster(): void
+ setObject(): void
UI
- f_30: Font
- f_35: Font
- f_60: Font
- f_50: Font
- f_40: Font
- g2: Graphics2D
+ coommandNum: int
~ keyH: InputManager
+ commandNum: int
- pacLife: BufferedImage
+ gp: GameEngine
+ draw(Graphics2D): void
- drawSecondLevelScreen(): void
- drawPlayerLife(): void
- drawPauseScreen(): void
- getXforCentered(String): int
- drawFirstLevelScreen(): void
- drawEndScreen(): void
- drawTitleScreen(): void
+ drawContaPallini(Graphics2D): void
Tilemanger
- conteggio: int
- gp: GameEngine
+ tile: Obstacole[]
+ mapTilenum: int[][]
- getTileImage(): void
~ draw(Graphics): void
- loadMap(String): void
~ resetPalliniTotali(): void
+ getMap(int, int): int
+ getPalliniTotali(): int
Sound
- clip: Clip
- isPlaying: boolean
- soundURL: URL[]
+ isPlaying(): boolean
+ play(): void
+ stop(): void
+ setFile(int): void
+ playWin(): void
gestoreRipristinoImmunita
~ gp: GameEngine
~ i: int
+ run(): void
GameEngine
+ GRI: gestoreRipristinoImmunita
+ ghost: Entity[]
+ pw: GameObject[]
+ livelloMax: int
+ maxScreenCol: int
+ screenHeight: int
- originalTitleSize: int
+ player: Player
~ se: Sound
- keyH: InputManager
+ eHandler: EventHandler
+ aSetter: AssetSetter
+ maxScreenRow: int
+ screenWidth: int
- FPS: int
+ titleSize: int
- serialVersionUID: long
- scale: int
- gameThread: Thread
+ tileM: Tilemanger
~ ui: UI
+ endState: int
+ playState: int
+ cCheck: CollisionChecker
+ obj: GameObject[]
+ gameState: int
+ nextLevelState: int
~ sound: Sound
+ titleState: int
+ pauseState: int
+ livelloCorrente: int
+ StartGameThread(): void
+ setupGame(): void
+ stopMusic(int): void
+ killMonster(int): void
+ run(): void
+ playSE(int): void
- update(): void
~ restart(): void
+ paintComponent(Graphics): void
~ nextLevel(): void
GameObject
+ imageVita: BufferedImage
+ solidAreaDefaulty: int
+ y: int
+ imagecfu: BufferedImage
+ powerUp: BufferedImage
+ name: String
+ collision: boolean
+ solidAreaDefaultx: int
+ solidArea: Rectangle
+ x: int
+ drawPW(Graphics2D, GameEngine): void
+ drawCFU(Graphics2D, GameEngine): void
ObjCFU
PowerUp
InputManager
+ rightPressed: boolean
+ upPressed: boolean
+ spacePressed: boolean
- gp: GameEngine
+ downPressed: boolean
+ leftPressed: boolean
+ keyPressed(KeyEvent): void
+ keyReleased(KeyEvent): void
+ keyTyped(KeyEvent): void
Ghost
- ultimoMovimentoOpposto: String
- gp: GameEngine
- getImage(int): void
- chasePlayer(): void
- isInCage(): boolean
- fantasmaVulnerabile(): void
+ setAction(): void
+ getType(): int
- randomMovement(): void
vitaPacMan
~ gp: GameEngine
Player
- powerUpTimer: Timer
- keyH: InputManager
- GU: gestoreUccisoni
+ punteggio: int
~ hashKey: int
+ pallini_totali: int
- getPlayerImage(): void
- mangiaPW(int): void
+ draw(Graphics2D): void
+ setDefaultLife(): void
+ update(): void
+ setDefaultValue(): void
- reset(): void
+ countLife(): int
- mangiaPalline(int): void
- interazioneFantasma(int): void

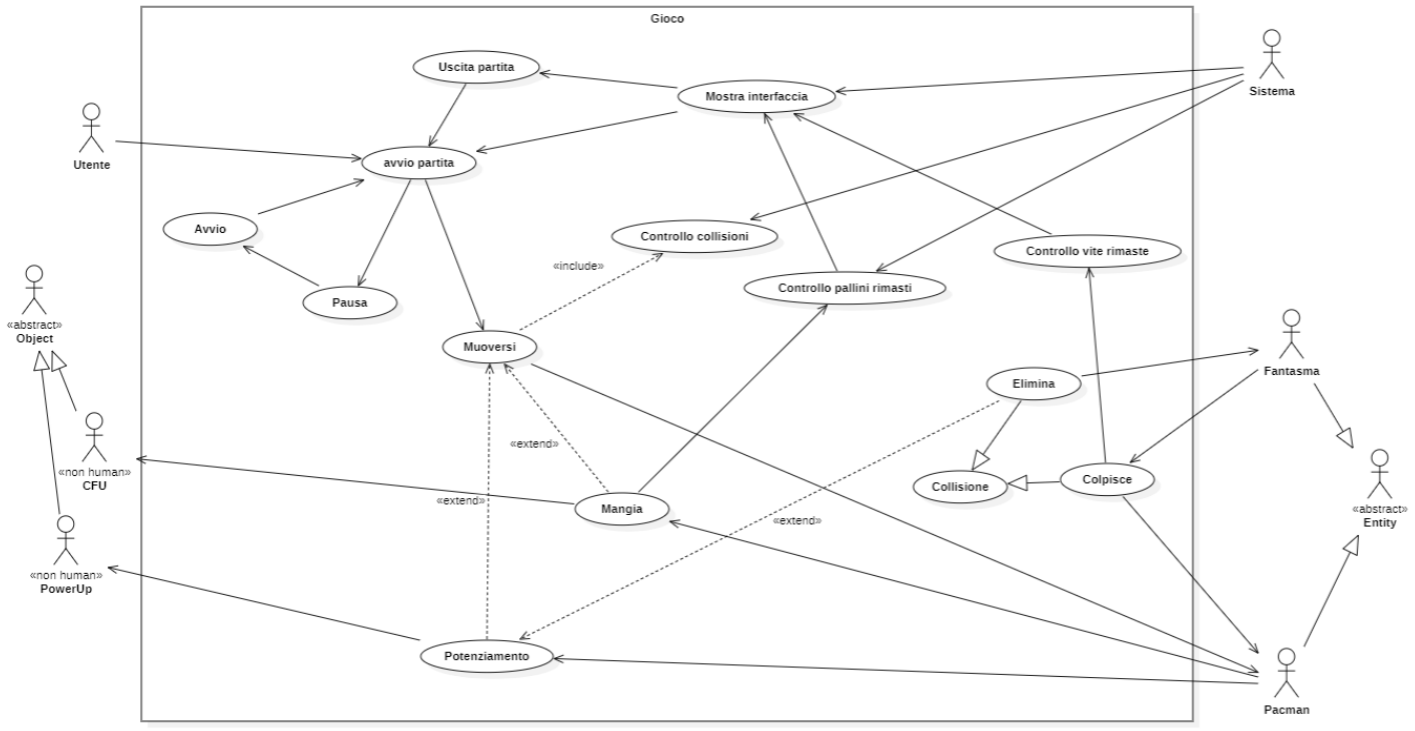
7.2 Il diagramma della macchina a stati

Di seguito lo state chart machine che presenta lo studio dal punto di vista dell'interfaccia utente e di ciò che viene attivato all'interno del gioco quando si verificano i vari eventi.



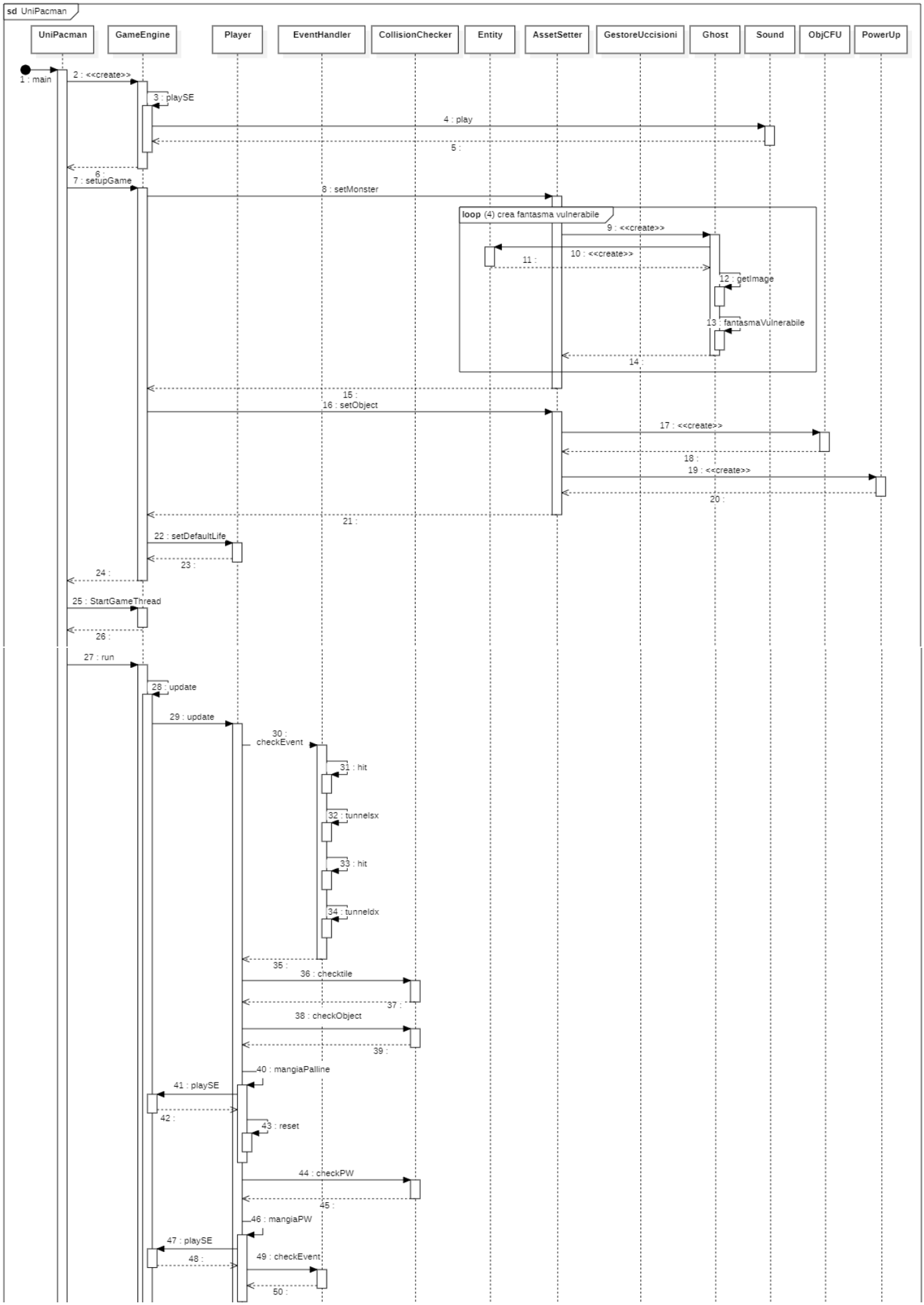
7.3 Il diagramma dei casi d'uso

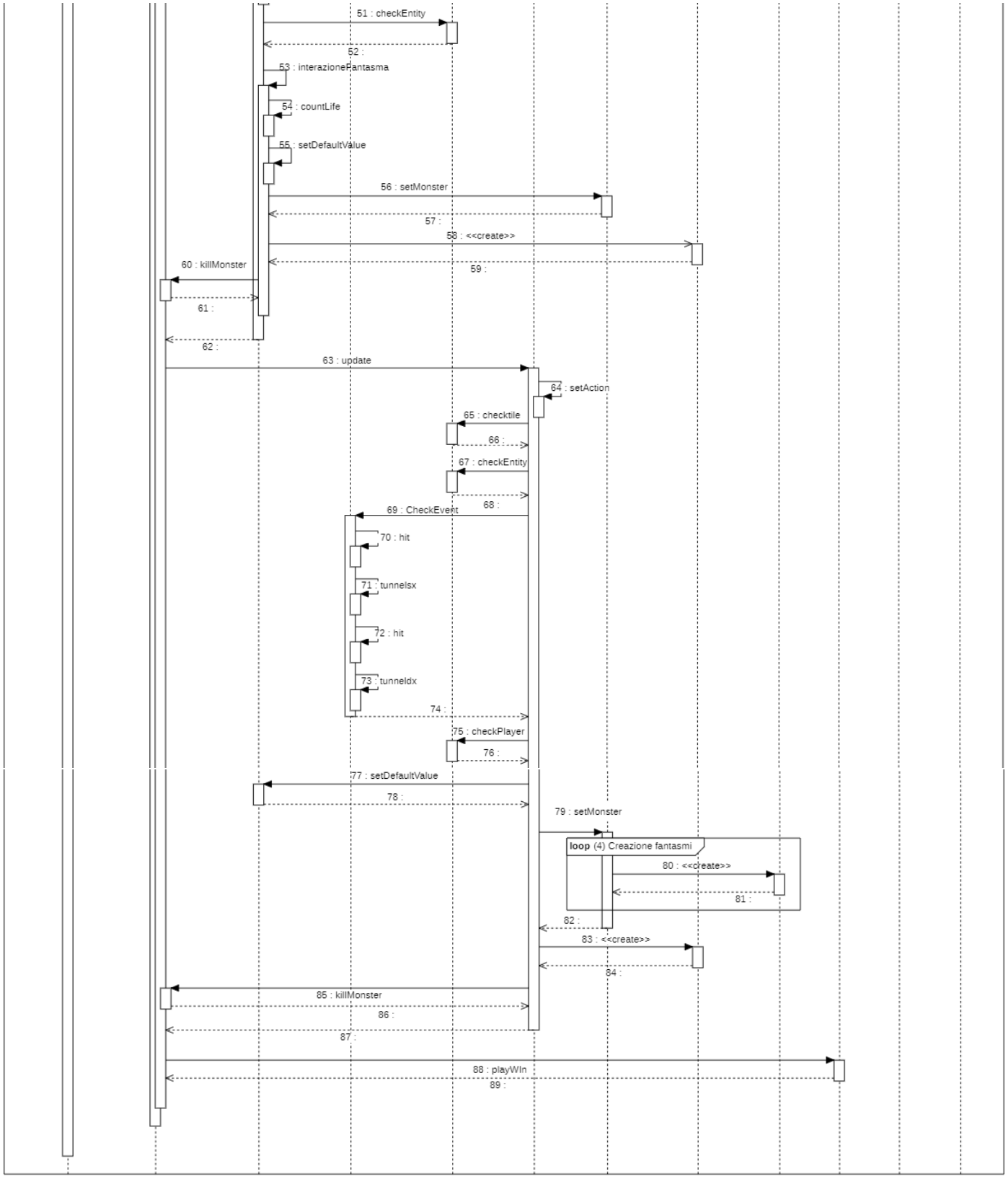
Il digramma dei casi d'uso considera tutte le interazioni tra le varie entità.



7.4 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.


sd UniPacman
UniPacman
GameEngine
Player
EventHandler
CollisionChecker
Entity
AssetSetter
GestoreUccisioni
Ghost
Sound
ObjCFU
PowerUp
1 : main
2 : <<create>>
3 : playSE
4 : play
5 :
6 :
7 : setupGame
8 : setMonster
loop (4) crea fantasma vulnerabile
9 : <<create>>
10 : <<create>>
11 :
12 : getImage
13 : fantasmaVulnerabile
14 :
15 :
16 : setObject
17 : <<create>>
18 :
19 : <<create>>
20 :
21 :
22 : setDefaultLife
23 :
24 :
25 : StartGameThread
26 :
27 : run
28 : update
29 : update
30 :
checkEvent
31 : hit
32 : tunnelsx
33 : hit
34 : tunneldx
35 :
36 : checktile
37 :
38 : checkObject
39 :
40 : mangiaPalline
41 : playSE
42 :
43 : reset
44 : checkPW
45 :
46 : mangiaPW
47 : playSE
48 :
49 : checkEvent
50 :



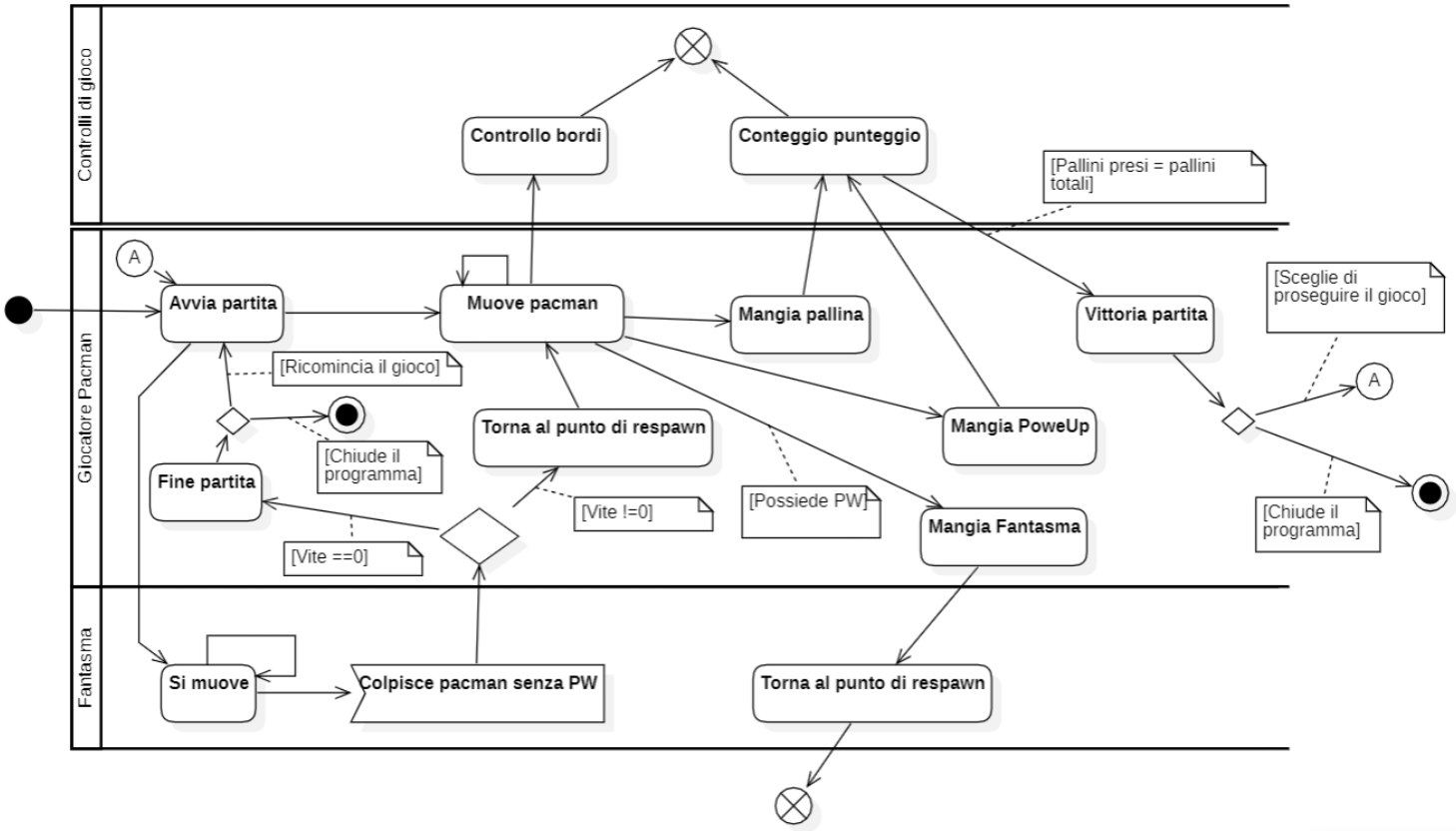
7.5 Il diagramma delle attività

Abbiamo individuato tre partizioni:

- Giocatore Pacman, dove troviamo tutte le attività comandate dall'utente;
- Controlli di Gioco, con le attività che il programma esegue in autonomia per un migliore funzionamento del gioco;
- Fantasma, in cui individuiamo le attività che svolgono i fantasmi che non interagiscono in autonomia con l'utente e l'ambiente di gioco.

L'attività "Avvia partita" rappresenta sia l'avvio del gioco che la scelta di proseguire al livello successivo da qui la presenza del:

- Ramo collegato al nodo iniziale;
- Collegamento dall'attività di fine partita, infatti il giocatore può scegliere di ricominciare il gioco;
- Connettore dal nodo decisionale che scaturisce da "Vittoria partita", in quanto alla vittoria di ogni livello si può avviare il livello successivo o in caso di vittoria definitiva può riavviare il primo livello.



8. *Software architecture*

Il software da noi progettato è basato sullo stile architetturale MVC:

- Model (Modello): Il Model rappresenta la logica dell'applicazione. Gestisce la manipolazione dei dati, le regole e la comunicazione con il sistema. La separazione del Model permette una gestione efficace della logica di gioco senza vincoli diretti sulla presentazione. La classe che gestisce il Model è GameEngine.java.

- View (Vista): La View gestisce la presentazione e l'interfaccia utente. È responsabile della visualizzazione dei dati provenienti dal Model e della gestione degli input dell'utente. La separazione della View consente una gestione flessibile dell'aspetto visivo del gioco senza inficiarne la logica.

- Controller (Controllore): Il Controller gestisce l'input dell'utente, interagisce con il Model e aggiorna di conseguenza la View. Promuove la separazione delle responsabilità, consentendo una gestione chiara degli eventi generati dagli input dell'utente e delle azioni di sistema. Utilizzare una separazione del Controller facilita la manutenzione e l'estensione del codice, migliorando anche la testabilità.

L'Architettura MVC genera diversi vantaggi per il nostro gioco:

- Separazione Chiara delle Responsabilità: La netta separazione tra Model, View e Controller semplifica la comprensione e la manutenzione del codice. Ogni componente, infatti, possiede un ruolo definito, contribuendo a una struttura ordinata e modulare.
- Facilità di Estensione: L'architettura MVC facilita l'aggiunta di nuove funzionalità senza dover modificare profondamente le componenti esistenti. La modularità permette di estendere il Model, la View o il Controller in modo indipendente.
- Testabilità Migliorata: La separazione delle componenti semplifica la scrittura di test unitari per il Model e il Controller. I test possono essere eseguiti in modo isolato per garantire la robustezza del sistema.

L'adozione dell'architettura MVC nel nostro progetto è stata una decisione strategica per migliorare la struttura del codice, facilitare la manutenzione e consentire una crescita organica del gioco. La chiara divisione delle responsabilità rappresenta un approccio solido ma garantendo un sistema flessibile e robusto.

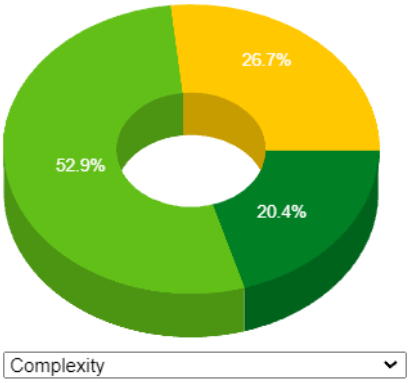
9. Software design

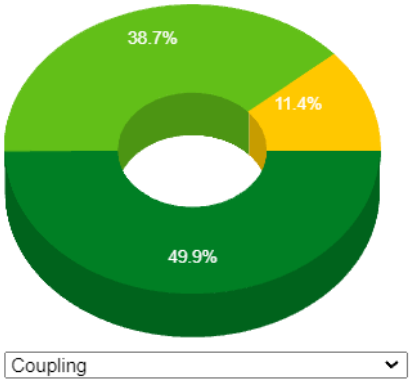
All'interno del progetto abbiamo deciso di utilizzare i seguenti pattern:

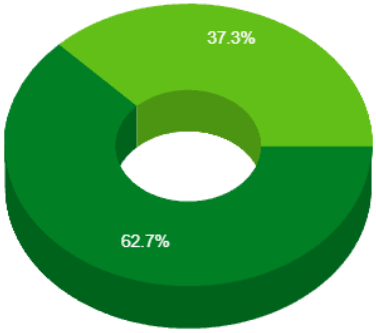
- Singleton Pattern: Pattern principale, utilizzato per la classe Player per garantire la presenza di un solo PacMan all'interno del gioco. È accessibile da tutte le classi per poter condividere informazioni come la posizione, il numero di vite rimaste e il punteggio.
- Factory Pattern: L'utilizzo di un Factory Pattern è dettato dalla scelta di introdurre una classe SetSetter in quanto durante la lettura della mappa in fase di caricamento, questa classe va a creare tutti gli oggetti presenti all'interno del gioco come i bordi, i pallini, i power-up e i fantasmi.
- Abstract Pattern: GameEngine contiene tutti i dati principali del nostro applicativo, dai dati di gioco, alle posizioni, al passaggio dei livelli e li condivide a tutte le altre classi per rimanere coerenti con la logica di gioco del vero PacMan.

Analizzando il codice prodotto abbiamo individuato una coesione comunicativa in quanto tutti gli elementi principali condividono le stesse informazioni.

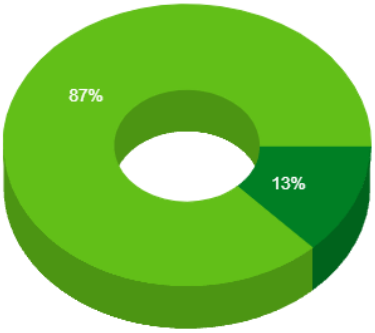
Attraverso il tool per Eclipse CodeMR abbiamo analizzato la complessità, coesione e accoppiamento delle classi ottenendo i seguenti risultati:







Lack of Cohesion



Size

In seguito, riportata anche una lista di tutte le classi presenti:

List of all classes (#19)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	GameEngine					139	medium-high	medium-high	low	low-medium
2	UI					203	low-medium	low-medium	low-medium	low-medium
3	Player					152	low-medium	low-medium	low-medium	low-medium
4	Entity					65	low-medium	low-medium	low	low-medium
5	AssetSetter					51	low	low-medium	low	low-medium
6	CollisionChecker					186	medium-high	low	low	low-medium
7	Ghost					107	low-medium	low	low	low-medium
8	InputManager					58	low-medium	low	low	low-medium
9	gestoreUccisoni					21	low-medium	low	low	low
10	gestoreRipristino...					12	low-medium	low	low	low
11	vitaPacMan					10	low-medium	low	low	low
12	ObjCFU					8	low-medium	low	low	low
13	PowerUp					8	low-medium	low	low	low
14	Tilemanger					99	low	low	low-medium	low-medium
15	EventHandler					37	low	low	low	low
16	Sound					28	low	low	low	low
17	UniPacman					16	low	low	low	low
18	GameObject					15	low	low	low	low
19	Obstacole					3	low	low	low	low

10. Software testing

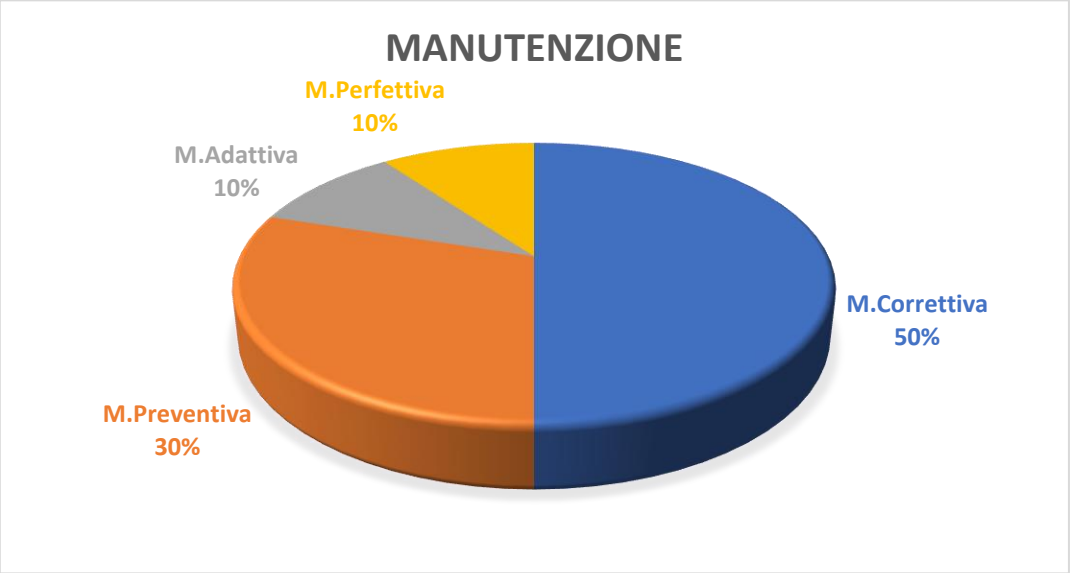
La fase di testing del codice sin dalle prime versioni del programma è stata svolta tramite il debug di Eclipse, anche grazie all'utilizzo di breakpoint, si è potuto osservare l'esecuzione della funzionalità appena implementata. Questo è stato reso possibile dal momento in cui abbiamo un'architettura MVC dove l'output del codice viene fornito direttamente a schermo perciò ogni singolo input è prontamente visualizzabile. Nel caso in cui si fosse riportato un effetto indesiderato nel testing, tramite un processo empirico di verifica e correzione abbiamo ottenuto il risultato desiderato. Per testare ad esempio il passaggio fra i livelli abbiamo imposto che per passare al livello successivo non ci volessero la totalità dei pallini mangiati bensì un numero nettamente inferiore, ad esempio quattro. Questo ha velocizzato la costruzione della mappa. Altro esempio sono stati le modifiche istantanee della rinascita dei fantasmi della procrastinazione oppure della loro vulnerabilità anche senza prendere un PowerUp. Tutte queste funzionalità sono state volutamente momentanee, infatti nella versione finale del codice non vengono riportate.

11.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 con un approccio di manutenzione prevalentemente correttiva e preventiva per garantire un codice sempre funzionante, privo di errori e 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.