­



PROGETTO INGEGNERIA DEL SOFTWARE – FAKE INVADERS

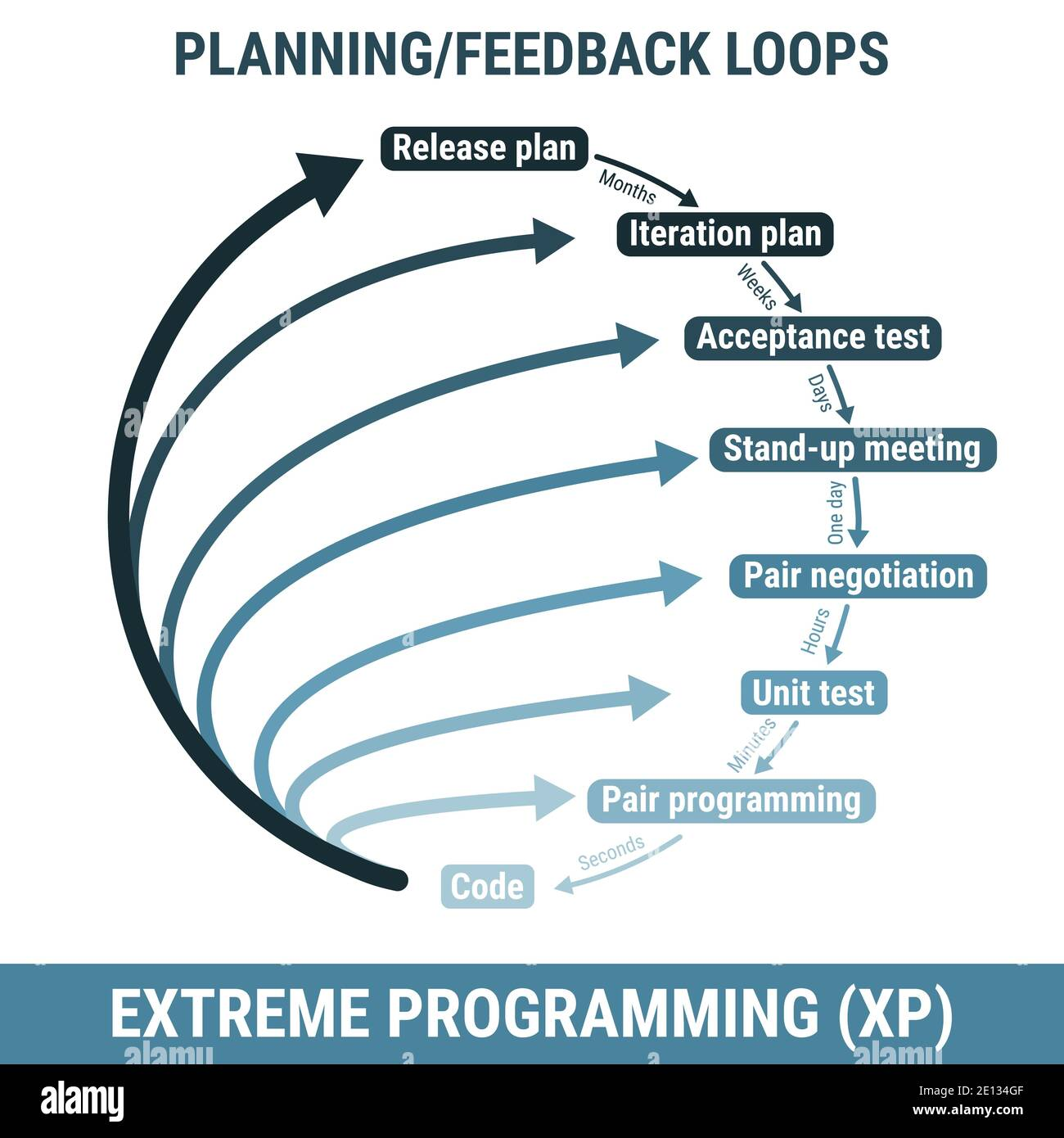
Tomasoni Francesco 1080980

Maccari Luca 1081951

**1. Software Life Cycle**

Il metodo di sviluppo che è stato adottato è il metodo agile: Extreme Programming (XP). Questa scelta è stata motivata dalla dimensione molto ridotta del nostro gruppo, poiché XP si adatta alla perfezione con una “coppia di programmatori che si guarda le spalle”.

Durante tutta la fase di progettazione del software è stato seguito il ciclo dell’XP di seguito riportato.



**2. Configuration Management**

Per la gestione della configurazione sono stati utilizzati i sistemi GitHub e GitHub Desktop. Tutto il lavoro svolto, che si tratti di documentazione o di codice, viene salvato in una repository in condivisione con tutti i membri del team.

Struttura del progetto:  
All’interno della repository sono presenti 2 cartelle:

* **Codice** in cui è presente sia il codice che le risorse necessarie a farlo funzionare
* **Docs** in cui vi è un file contenente la documentazione completa del progetto, e una cartella contenente i diagrammi UML.

Branch utilizzati:

* **Main**: branch principale.
* **Branch** **individuali**: Francesco e Luca.

Inizialmente i *commit* sono stati svolti nel branch principale Main, successivamente con l’aumento delle skils del team, si è passato al lavoro nei brach individuali con l’utilizzo di *pull request* per incorporare le modifiche nel Main.

Per tenere traccia del lavoro da svolgere e dei vari problemi, sono state utilizzate delle *issue* assegnate ai membri del team.

La repository è stata mantenuta costantemente aggiornata, includendo ogni documentazione prodotta.

**3. People Management and Team Organization**

Il team è composto da Maccari Luca e Tomasoni Francesco.

Nella realizzazione del Progetto è stata adottata una struttura organizzativa di tipo Agile, inquanto offre canali di comunicazione brevi e una migliore flessibilità e adattabilità alle esigenze del progetto, ponendo il focus anche sulle disponibilità dei singoli individui coinvolti.

In questa configurazione, non vi è una suddivisione prefissata del lavoro, è compito del team decidere autonomamente come suddividere e assegnare le diverse attività. In situazioni di difficoltà, entrambi i membri del team hanno la possibilità di collaborare, affiancandosi reciprocamente per risolvere le problematiche in modo collettivo.

Il meccanismo di coordinamento adottato è stato *l’adhocrazia*, caratterizzato da grande capacità di adattamento, offre ulteriori vantaggi, contribuendo a creare un ambiente più sereno e informale, favorevole al benessere dei membri del team.

**4. Software Quality**

Il team si è prefissato di sviluppare un'applicazione che rispetti i parametri e gli attributi di qualità definiti da *McCall*.

Di seguito sono elencati suddivisi in categorie.

### Operatività del software

* **Correttezza:** L'applicazione soddisfa i requisiti e le specifiche definite all'inizio del progetto. Eventuali funzionalità non ancora implementate verranno aggiunte mediante futuri aggiornamenti del software.
* **Affidabilità:** L'applicazione è stata sottoposta a diversi test, risultando con il minor numero possibile di errori.
* **Efficienza:** Il consumo di risorse è ottimizzato, eliminando la necessità di hardware potente o una grande quantità di RAM. Il peso dell'applicazione è limitato a pochi MB.
* **Integrità:** il software è sicuro, in quanto è necessaria un’autenticazione con username e password.
* **Usabilità:** Il prodotto è progettato per essere intuitivo e non richiede competenze particolari per essere utilizzato. L'applicazione è ottimizzata su tutte le piattaforme desktop grazie all'utilizzo di librerie compatibili con ciascun sistema operativo, garantendo una grafica uniforme.

Revisione del software

* **Manutenibilità**: Il team si è impegnato per organizzare il codice al meglio, per poterlo mantenere facilmente
* **Testabilità**: Tutte le funzionalità incluse sono testabili in qualunque momento tramite test.
* **Flessibilità**: Grazie all' organizzazione del codice, il software è flessibile, garantendo semplificazioni in eventuali ulteriori modifiche e personalizzazioni al programma.

### Transizione verso un nuovo ambiente

* **Portabilità**: Il gioco è nativamente eseguibile sui più diffusi sistemi operativi desktop (Windows, MacOS) dotati di un hardware relativamente recente e una versione di Java installata al loro interno.
* **Riusabilità**: Il codice è stato scritto in modo modulare, per poter essere riutilizzabile in futuro.

Requisiti base del software:

* Avere un pc con sistema operativo Windows o MacOS
* Avere una scheda video (anche integrata)

**5. Requirement Engineering**

La prima fase dello sviluppo è stata focalizzata sulla definizione dei requisiti necessari per la realizzazione software. Poiché la creazione del gioco non è stata commissionata, ma è stata un'idea spontanea del team, i requisiti sono stati determinati in base a ciò che si considerava essenziale, tenendo conto della conoscenza del videogioco (Space Invaders) da cui il software ha preso ispirazione, cercando di non snaturarlo ma rinnovarlo ed implementare nuove funzionalità.

I requisiti sono stati delineati seguendo il modello *MoSCoW*:

**Must Have:**

* Sviluppo di una versione base del gioco funzionante.
* Interfaccia grafica intuitiva e adatta a tutti i tipi di giocatori.
* Stabilità e sicurezza del software.

**Should Have:**

* Impostazioni di gioco.
* Modalità multigiocatore

**Could have:**

* Effetti sonori.
* Traduzione del gioco in altre lingue.

**Won't Have:**

* Impostazioni per modificare la grafica.

**6. Modelling**

la modellazione è avvenuta tramite l’utilizzo dei seguenti diagrammi UML:

* use case diagram
* class diagram
* state machine diagram
* sequence diagram
* activity diagram
* component diagram

Immagine che contiene diagramma, linea, Disegno tecnico, Piano

Descrizione generata automaticamenteDi seguito riportato il diagramma delle componenti che rappresenta l’architettura del software:

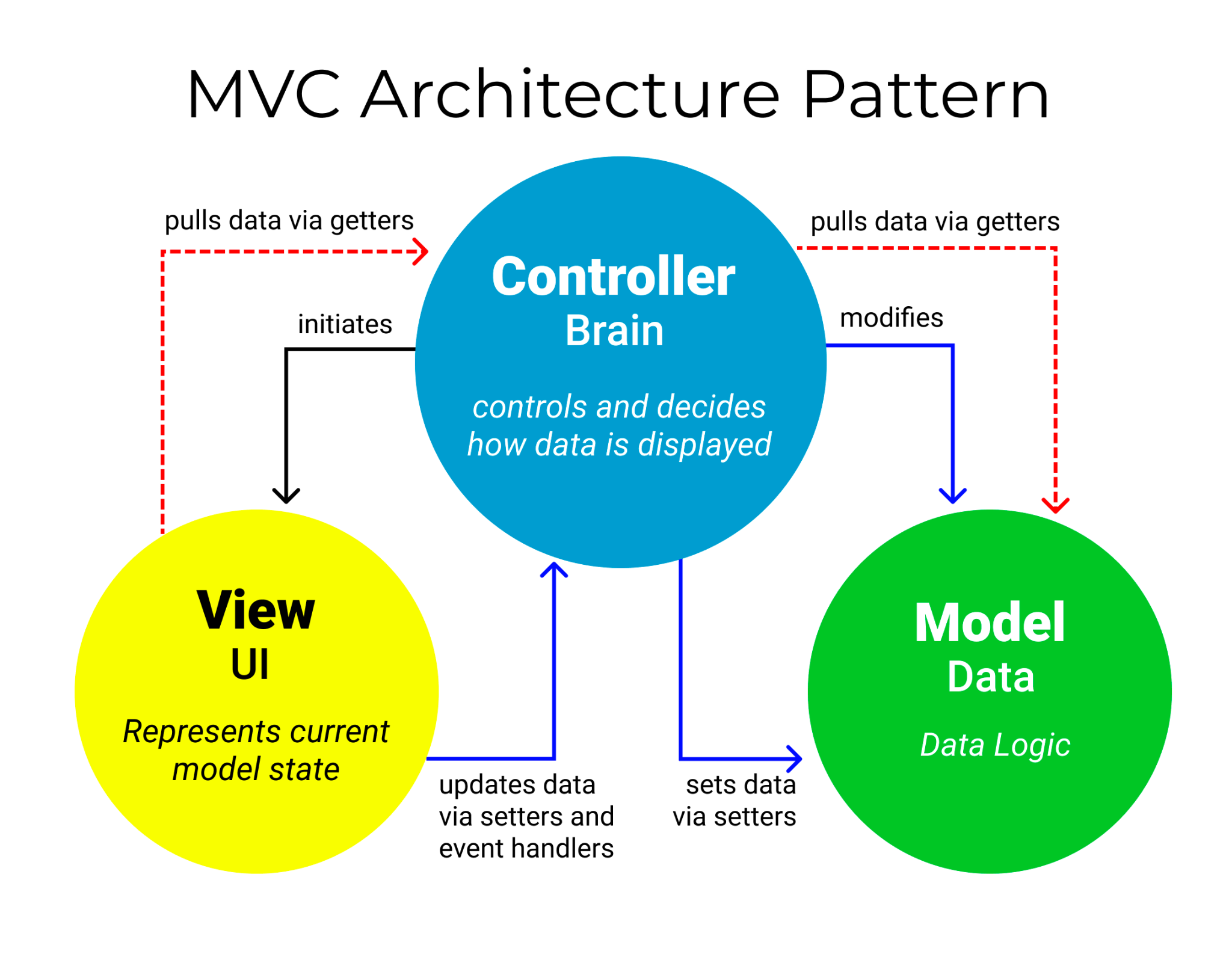
**7. Software Architecture**

Il software è stato progettato mediante l’ausilio del pattern *Model-View-controller* (*MVC)*, dividendo il codice in package distinti: **ModelPack**, **ViewPack**, **ControllerPack**.

**Model**: Nel package *ModelPack* risiedono le classi che definiscono la logica di business e gestiscono i dati. Il Model interagisce con il database (database package) per recuperare, aggiornare o salvare informazioni.

**View**: Il package *Viewpack* contiene le classi responsabili della GUI che si interfaccia con l’utente. e interagiscono con il Controller per la gestione delle azioni dell’utente. La separazione della logica di presentazione in questo package favorisce la modularità e la riusabilità del codice.

**Controller**: Il package *ControllerPack* contiene le classi Controller, che fungono da intermediari tra il Model e la View. I Controller gestiscono gli input utente, coordinando le azioni richieste e aggiornando il Model e la View di conseguenza.



**8. Software Design**

**8.1 Diagrammi**

La descrizione del design è svolta mediante l’utilizzo di diagrammi UML, di seguito si possono trovare:

Diagramma delle sequenze che mostra il dialogo tra Player, Menu e Database

Immagine che contiene testo, diagramma, Piano, Disegno tecnico

Descrizione generata automaticamente

Diagramma delle attivitàImmagine che contiene diagramma, linea, Piano, Disegno tecnico

Descrizione generata automaticamente

**8.2 Design Pattern**

Si è optato per adottare il ***singleton*** *pattern* al fine di creare il progetto del database, garantendo che soltanto una singola istanza della classe possa essere presente. Questo risultato è conseguito tramite l'utilizzo di un'unica istanza privata, un costruttore privato e, un metodo pubblico: *getIstance()* che permette di richiamare l'istanza quando necessario.

Inoltre, si è adottato il pattern ***DAO (Data Access Object*)** per gestire l'accesso ai dati del database. Questo pattern separa la logica di accesso ai dati dalla logica di business, garantendo una maggiore organizzazione e facilitando la manutenibilità del codice.

Sono state create le classi *Player\_DAO* e *Last\_game\_DAO* che utilizzano questo pattern e che contengono i metodi per l’accesso e manipolazione dei dati nel database.

Con il DAO, si astraggono i dettagli specifici dell'accesso ai dati dietro un'interfaccia chiara, consentendo di modificare la fonte dei dati senza influire sulla logica principale dell'applicazione. Inoltre, il DAO favorisce il riutilizzo del codice, la testabilità e offre flessibilità nell'implementare diverse strategie di accesso ai dati.

Inizio modulo

Fine modulo

**8.3 Metriche**

**8.3.1 Structure 101**

Analizzando il codice tramite *Structure101 Studio*, si possono trarre i seguenti risultati:

- La percentuale di fat è anch‘essa pari a zero. Questo suggerisce che i singoli componenti del codice sono snelli e concentrati su singole responsabilità. Una bassa complessità dei componenti rende il codice più manutenibile e comprensibile.

- La percentuale di tangled è pari a zero. Questo indica che il codice presenta una bassa complessità in termini di interconnessione tra i componenti. In altre parole, i componenti sono chiaramente separati e le relazioni tra di essi sono ben definite.

Immagine che contiene testo, schermata, diagramma, Carattere

Descrizione generata automaticamenteImmagine che contiene testo, schermata, schermo, Carattere

Descrizione generata automaticamenteDi seguito riportati i diagrammi dell’analisi tramite *Structure101 Studio:*

**8.3.2 CodeMR**

Un ulteriore analisi è stata effettuata tramite il tool *CodeMR* da cui è scaturito il seguente diagramma:

**Lack of Cohesion:** Misura il grado di correlazione tra i metodi di una classe. Un'alta coesione tende a essere preferibile, perché l'alta coesione è associata a diversi tratti desiderabili del software, tra cui la robustezza, l'affidabilità, la riusabilità e la comprensibilità.

**Complexity:** Implica la difficoltà di comprensione e descrive le interazioni tra una serie di entità. Livelli più elevati di complessità nel software aumentano il rischio di interferire involontariamente con le interazioni e quindi aumenta la possibilità di introdurre difetti quando si apportano modifiche.

**Size**: Misura il numero di righe o di metodi presenti nel codice. Un numero molto alto può indicare che una classe o un metodo sta cercando di fare troppo lavoro e dovrebbe essere diviso. Potrebbe anche indicare che la classe potrebbe essere difficile da mantenere.

**Coupling**: Si riferisce al grado di dipendenza tra due o più componenti di un sistema software. Indica quanto due parti del codice sono connesse o dipendenti l'una dall'altra. Un basso accoppiamento è desiderabile perché significa che i componenti sono indipendenti e possono essere modificati, testati o sostituiti senza influenzare gli altri componenti del sistema.

Immagine che contiene testo, schermata, cerchio, design

Descrizione generata automaticamente

**9. Software Testing**

**9.1 JUnit Test**

Durante la fase di test del progetto, sono stati eseguiti controlli simultaneamente alla fase di sviluppo del codice. L'obiettivo era verificare che ogni componente integrato nel programma funzionasse correttamente, rispettando le specifiche funzionali definite.

Inizialmente, sono stati svolti test per verificare la corretta gestione del database, garantendo che le operazioni di lettura e scrittura avvengano come previsto e che i dati vengano salvati e recuperati correttamente.

Successivamente, sono stati condotti test specifici per valutare l'accuratezza dei metodi della classe principale dell'applicazione, la classe Board. Questi test hanno verificato se i metodi della classe funzionassero correttamente e se restituissero i risultati attesi in varie condizioni di input.

**9.2 Test manuale**

Durante lo sviluppo, seguendo i principi dell’Extreme Programming (XP), oltre ai test in JUnit per verificare le singole unità di codice, sono stati eseguiti test sull'eseguibile per garantire il corretto funzionamento dell'applicazione nel suo complesso.

Questa pratica è stata adottata per assicurare che il software fosse sempre funzionante e che eventuali modifiche non introducessero problemi, mantenendo così alta la qualità del codice conforme agli standard XP.

La combinazione di entrambi i tipi di test ha contribuito a fornire un sistema affidabile agli utenti, in linea con gli obiettivi di XP.

**10. Software Maintenance**

La manutenzione del codice è stata svolta tramite attività di **refactoring** che ha previsto:

* pulizia del codice che comprende l’eliminazione di metodi inutili e l’ottimizzazione di quelli necessari
* ridenominazione dei package in: FAKEINVADERS\_Package
* aggiunta di commenti ai test
* aumento della sicurezza per quanto riguarda l’accesso al database

In generale, l'obiettivo del **refactoring** è quello di rendere il codice più pulito, più chiaro, più efficiente e più facile da mantenere nel tempo, contribuendo così a migliorare la qualità complessiva del software.

Inizio modulo

Fine modulo