

FAKE INVADERS

TOMASONI FRANCESCO 1080980
MACCARI LUCA 1081951



OBIETTIVO

Lo scopo del progetto è creare una versione modificata del celebre gioco SPACE INVADERS mediante l'utilizzo di Java

Fake Invaders ha le stesse dinamiche del gioco madre, bisogna annientare le navicelle aliene prima che raggiungano la terra

DIFFICOLTÀ INCONTRATE

- ★ Utilizzo di **GitHub**, capire le sue dinamiche, l'uso di branch singoli e eventuali conflitti di merge che si sono creati nel corso della scrittura del codice.
- ★ Difficoltà nel rispettare i requisiti imposti
- ★ Difficoltà nell'apprendimento delle dinamiche di javaSwing

PARADIGMA DI PROGRAMMAZIONE

Il team ha adottato **StarUML** per la modellazione, utilizzando **UML** come linguaggio di rappresentazione.

Per lo sviluppo del software, la scelta è ricaduta su **Eclipse** e **VSCode** come IDE, mentre l'applicazione è stata implementata in linguaggio **Java** con paradigma OOP e gestione dell'interfaccia utente mediante **Java Swing**.

Per la valutazione del codice, è stato impiegato **CodeRM** e **Structure 101**.

Per l'organizzazione del lavoro e dei meeting, **Discord** è stato il principale strumento utilizzato.



SOFTWARE CONFIGURATION MANAGEMENT

Il lavoro del team è conservato in una repository su GitHub condivisa tra i membri.

Ciascun componente del team ha creato un proprio branch.

Durante il lavoro, ogni membro ha estratto la versione aggiornata dal branch principale utilizzando le **pull request**.

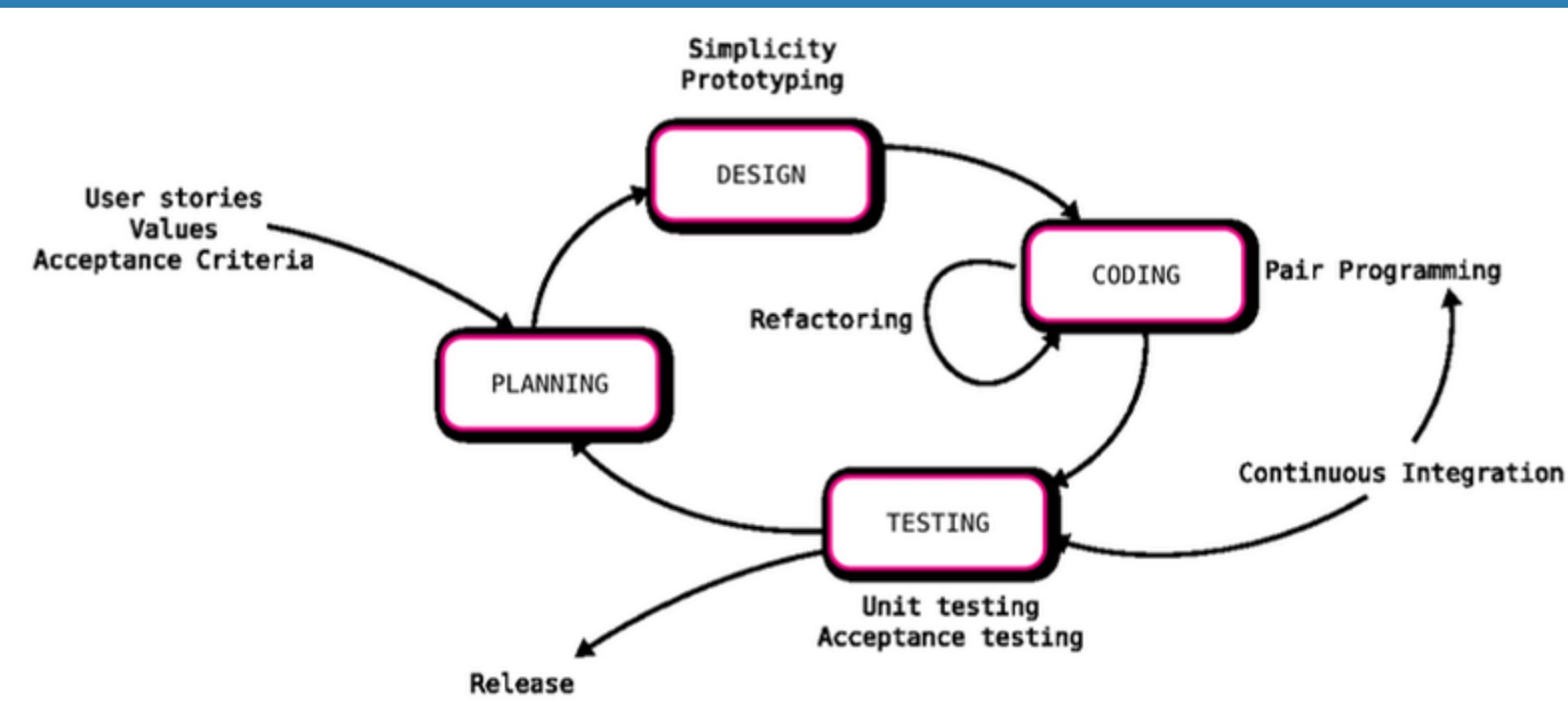
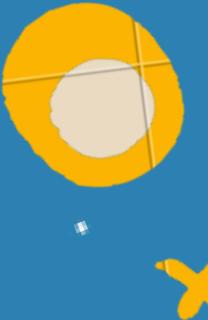
Al completamento del lavoro, nella prima fase di sviluppo, il codice veniva caricato direttamente nel branch main.

Con il passare del tempo e l'aumentare delle skill, il codice è stato poi caricato prima nei branch individuali, per poi allineare il **main**





SOFTWARE LIFE CYCLE



Il team ha scelto un approccio ti tipo agile, in particolare **l'Extreme Programming (XP)**, perchè era quello che si adattava meglio alla situazione.
“*coppie di programmatore, uno programma l'altro gli guarda le spalle*”

REQUISITI

I requisiti sono stati individuati e priorizzati secondo 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.
- Effetti sonori.



Could have:

- Modalità multigiocatore
- Traduzione del gioco in altre lingue.



Won't Have:

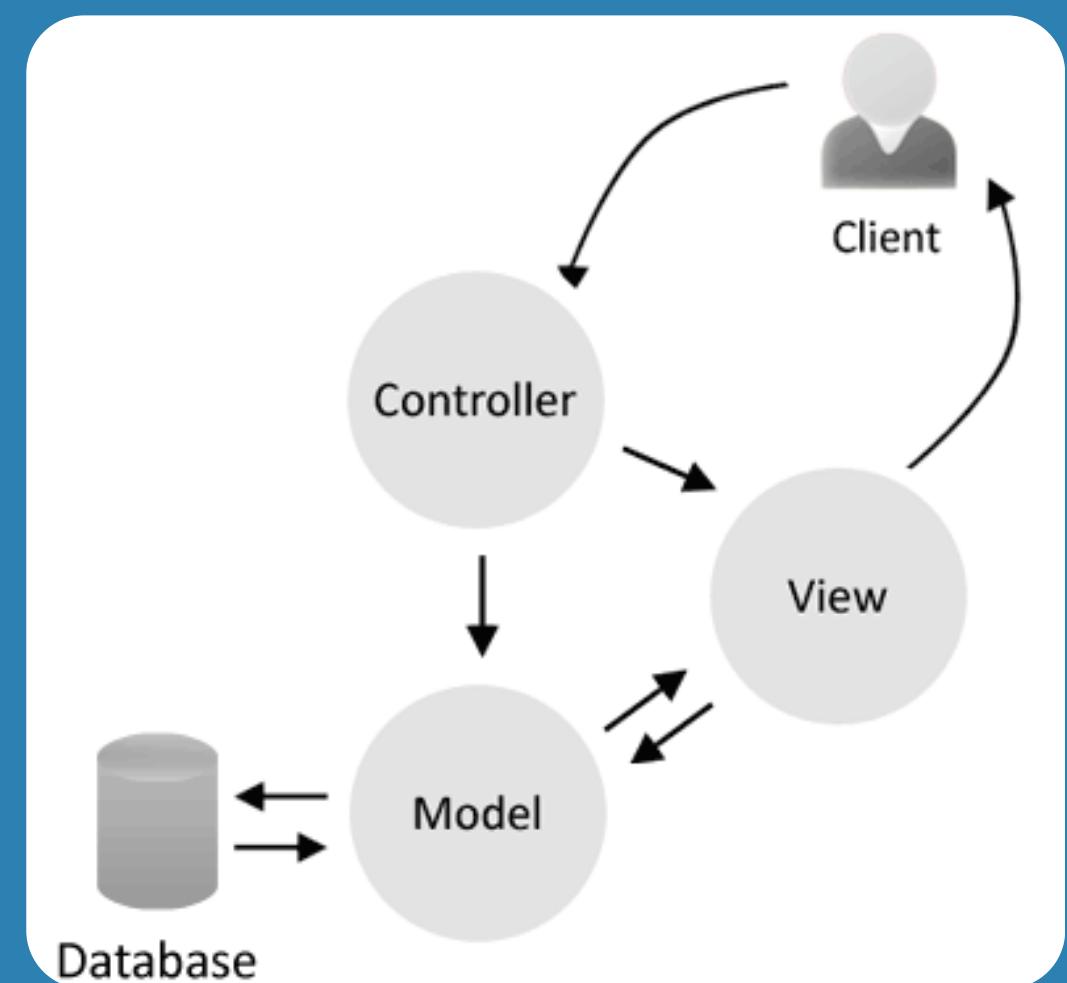
- Impostazioni per modificare la grafica.



ARCHITETTURA

Il software è stato realizzato sfruttando il pattern **MCV**, che consente di separare parte grafica, logica e la gestione dei dati del programma

- Model: Rappresenta i dati e la logica di business dell'applicazione.
- View: Si occupa della presentazione dei dati all'utente.
- Controller: Gestisce l'input dell'utente, manipola il Model e aggiorna la Vista di conseguenza.



DESIGN PATTERN

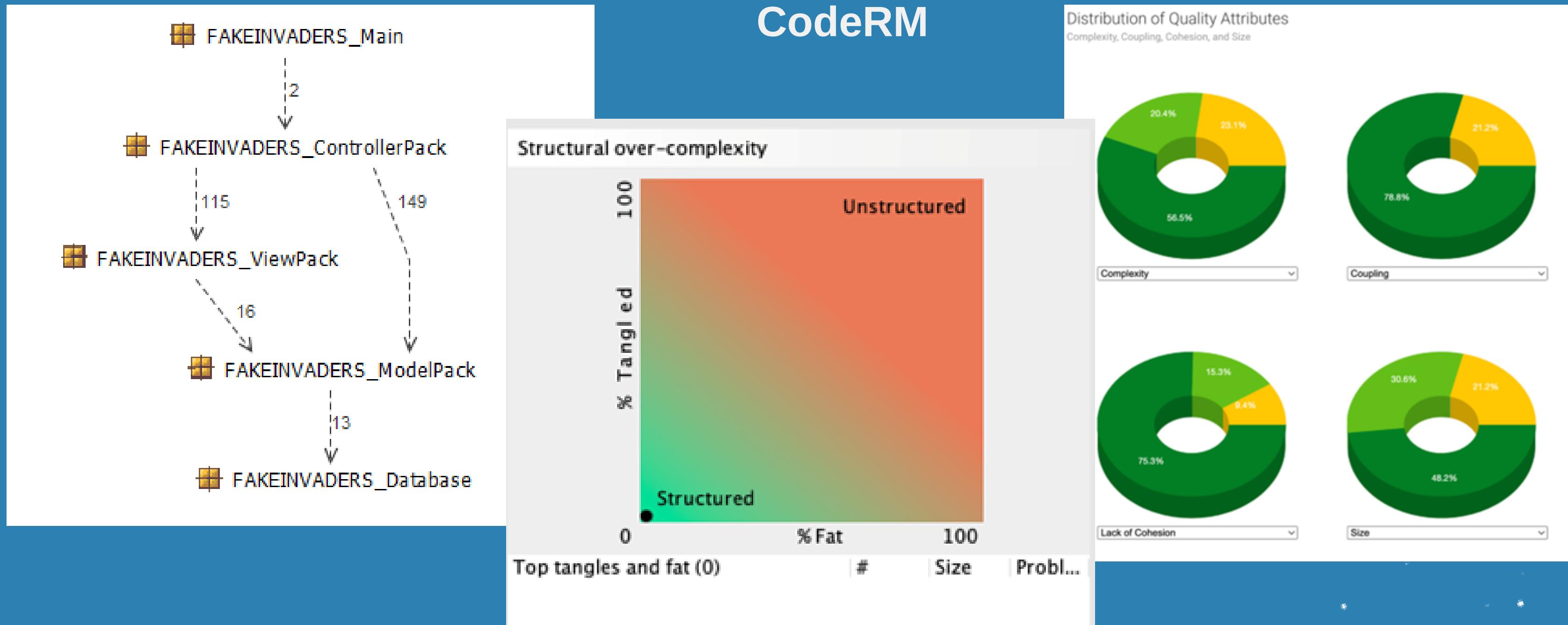


È stato adottato il **singleton pattern** nella creazione del progetto del database, garantendo che la classe possa essere presente tramite un'unica istanza privata, un costruttore privato e, un metodo pubblico: `getInstance()`.

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

METRICHE DI QUALITÀ

Per misurare la qualità del software sono stati utilizzati il software **Structure101** ed il tool



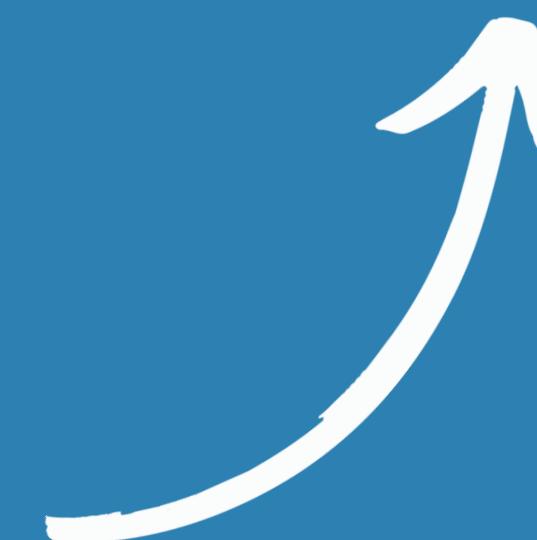
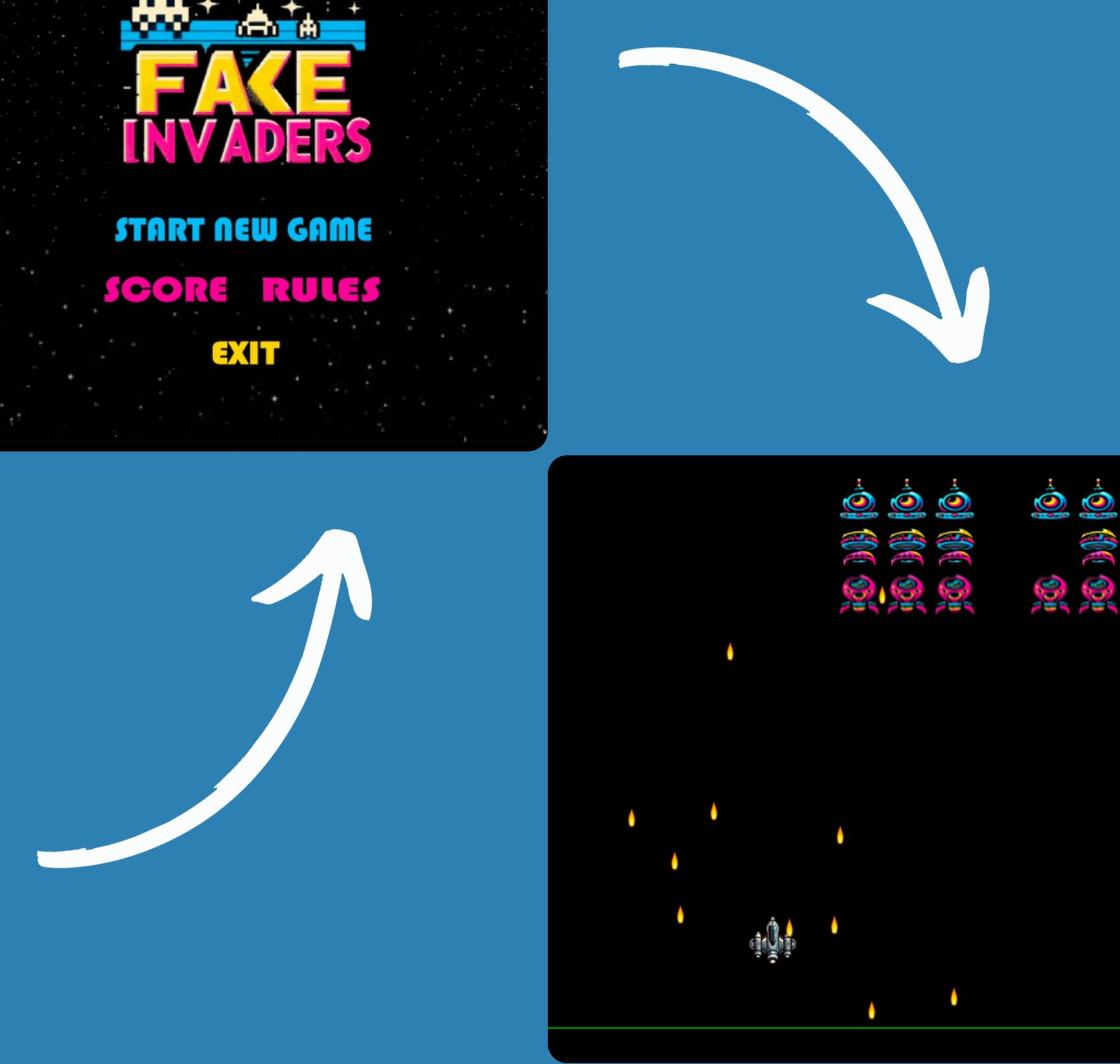
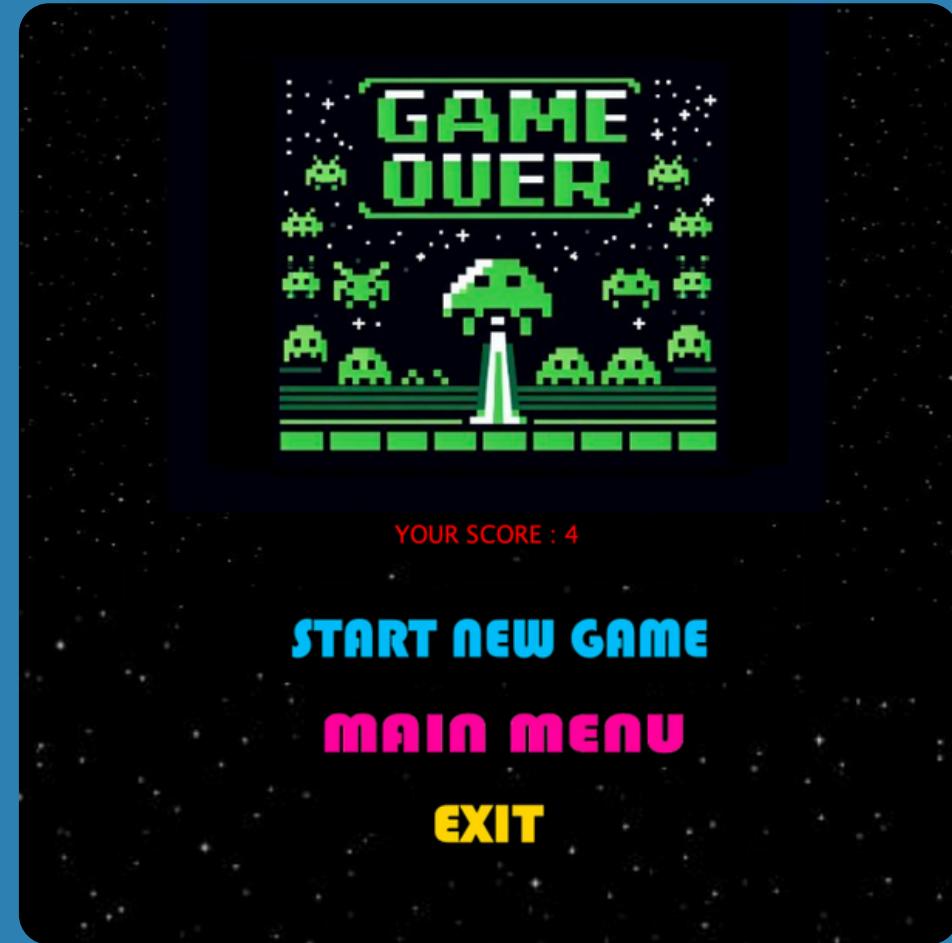
IMPLEMENTAZIONE

Rispetto ai requistiti designati con il modello MoSCoW, siamo riusciti a implementare:

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



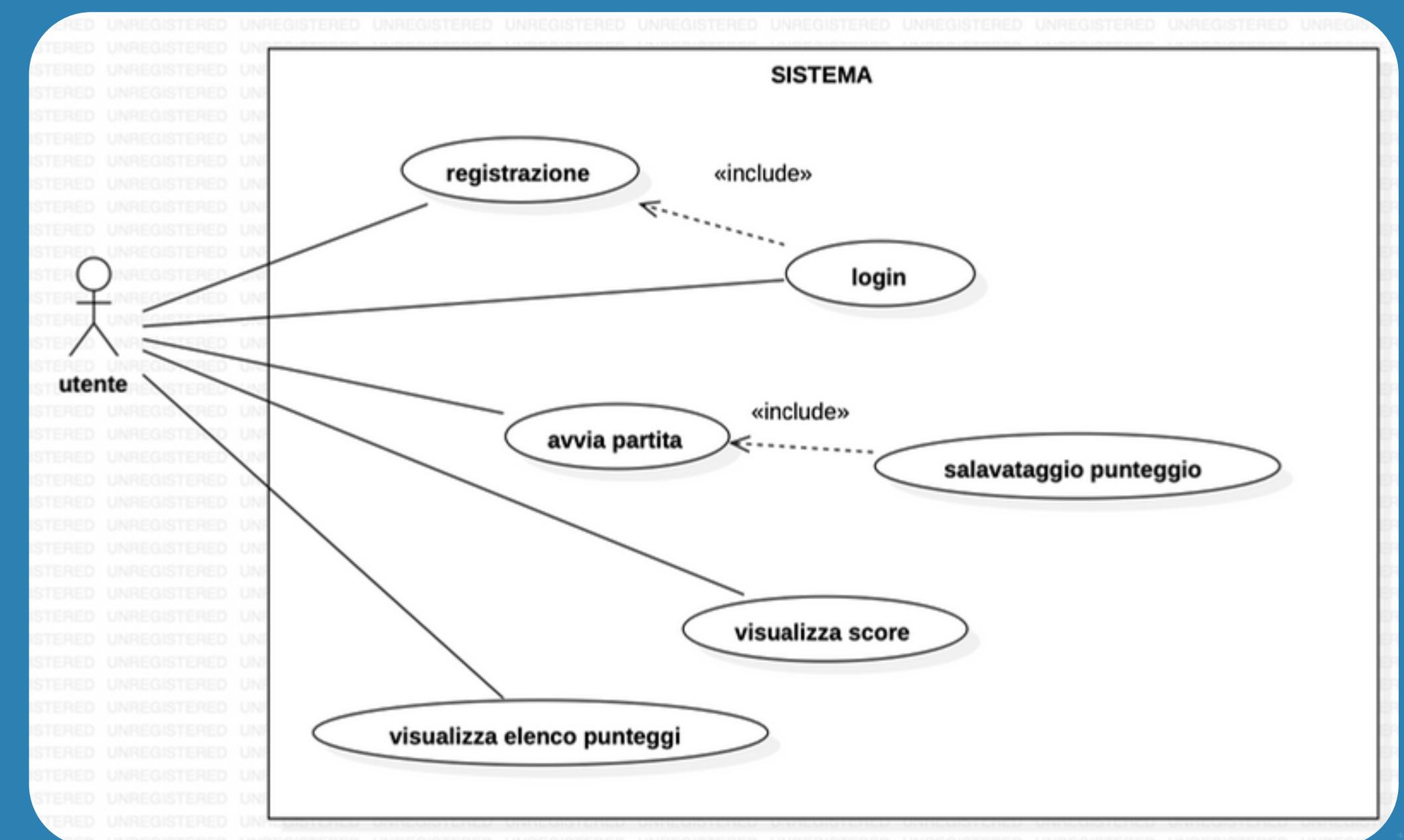
DEMO



MODELLAZIONE

Attraverso l'utilizzo di StarUML sono stati modellati i seguenti diagrammi:

- ❖ use case diagram
- ❖ class diagram
- ❖ state machine diagram
- ❖ sequence diagram
- ❖ activity diagram
- ❖ component diagram



TESTING

Per quanto riguarda la fase di testing del codice sono stati utilizzati due paradigmi, test in **junit**, che hanno permesso di trovare eventuali errori nel codice e quindi la loro risoluzione test manuali per assicurare che il software fosse sempre funzionante e che eventuali modifiche non introducessero problemi





**GRAZIE PER
L'ATTENZIONE**