# Progetto di ingegneria del software (rev. 4.2)

### Anno accademico 2023-2024

Membri del team

Tomasoni Francesco (1080980)

Maccari Luca (1080980)

Indice

Il progetto\_\_

* Project plan\_\_

1.1 Introduzione

1.2 Modello di processo

1.3 Organizzazione del progetto

1.4 Standards, linee guida e procedure

1.5 Attività di gestione

1.6 Rischi

1.7 Membri

1.8 Metodi e tecniche

1.9 Garanzie di qualità

1.10 Package di lavoro

1.11 Risorse

1.12 Budget e pianificazione

1.13 Cambiamenti

1.14 Consegna

/\*

- \_\_2. Software lifecycle\_\_

- \_\_3. Configuration management\_\_

- 3.1 Struttura del progetto

- 3.2 Issues

- \_\_4. People management\_\_

- \_\_5. Software quality\_\_

- 5.1 Operatività del software

- 5.2 Revisione del software

- 5.3 Transizione verso un nuovo ambiente

- \_\_6. Requirement engineering\_\_

- \_\_7. Modelling\_\_

- 7.1 Grafica

- 7.2 Funzionamento del back-end

- \_\_8. Software architecture\_\_

- \_\_9. Software testing e manutenibilità\_\_

\*/

**Il progetto**

Realizzazione in Java di una versione “a modo mio” ispirandosi al noto gioco Space Invader

Il giocatore ha lo scopo di difendere la terra da un’invasione aliena, distruggendo le navicelle aliene che lentamente si avvicinano

Guida rapida al gioco

**3. Tipi di Invader**

* **Alieni Standard:** Si muovono lateralmente e scendono gradualmente verso il basso.
* **Alieni Speciali:** Alcuni invasori sono più veloci o cambiano direzione più rapidamente.
* **Capo Alieno (Boss):** In alcuni livelli, potresti affrontare un capo alieno più grande e difficile da sconfiggere.

**4. Barriera Protettiva**

Tra te e gli invasori ci sono delle barriere protettive. Esse si deteriorano con il tempo e vengono colpite dai proiettili, quindi stai attento al loro stato.

**5. Strategie di Gioco**

* **Tattica dell'Avanzamento:** Sposta la tua navicella da lato a lato per evitare i proiettili nemici e cerca di avanzare gradualmente verso gli invasori.
* **Priorità di Attacco:** Fai attenzione agli invasori più vicini al basso, poiché raggiungeranno il terreno più rapidamente.
* **Pausa e Osserva:** Pratica la pausa e osserva il modello di movimento degli invasori per pianificare i tuoi spari.

**6. Punteggio**

Ogni invasore abbattuto ti guadagna punti, e alcuni potrebbero lasciare bonus speciali. Cerca di ottenere il punteggio più alto possibile!

**7. Livelli Successivi**

Man mano che avanzi nei livelli, la difficoltà aumenta con la velocità degli invasori e i loro schemi di movimento.

**8. Modalità Multiplayer**

Alcune versioni del gioco includono modalità multiplayer. Divertiti a competere contro i tuoi amici per vedere chi può ottenere il punteggio più alto.

Project plan

Si sviluppa il gioco pensando ad una community generica di videogiocatori. Nella fase iniziale del lavoro i requisiti sono decisi all'interno del team di sviluppo, mentre nel momento in cui l'applicazione sarà resa pubblica, gli utenti avranno una parte attiva nel suggerire modifiche e miglioramenti nonché di segnalare eventuali bug attraverso le piattaforme social e/o issue di GitHub raggiungibili anche dall'interno del gioco.

L'azienda che si occupa della produzione è una società di sviluppo software costituita da Luca Maccari e Tomasoni Francesco ed è stata costituita con il fine di progettare e sviluppare il software.

L'obiettivo del lavoro è dimostrare di saper applicare ad un caso di studio, i concetti, le metodologie e le tecniche appreso durante il corso. Il progetto deve contenere tutti gli argomenti trattati all’interno dei libri di testo.

1.1 Introduzione

Dopo un'attenta analisi dei requisiti ideati dal team ponendosi nell'ottica di un possibile giocatore, sono stati stabiliti i passaggi da seguire al fine di realizzare una prima versione del sistema richiesto.

A tutte le fasi fi progettazione del gioco prenderanno parte entrambi i membri; infatti, non vi saranno suddivisioni in ruoli, entrambi parteciperemo in egual modo alle diverse fasi del progetto.

Durante la prima riunione operativa si è deciso di sviluppare un'applicazione desktop che implementi le seguenti funzionalità:

//TODO PENSARE ALLE FUNZIONALITA DEL SISTEMA

- Sviluppo di un gioco multigiocatore online

- Il gioco dovrà possedere una grafica piacevole, avere una chat testuale e possibilmente vocale

- Il gioco dovrà supportare più lingue

- Il gioco dovrà essere disponibile per Windows, Mac e Linux

All'interno della riunione si è stabilito anche che per tutto il periodo di progettazione e sviluppo si sarebbero tenuti degli incontri settimanali in modo da favorire la comunicazione e collaborazione tre i membri del team, nonché per discutere sugli obiettivi e avere un feedback dell'intero gruppo di lavoro riguardo alle modifiche apportate.

Si è inoltre posta come deadline per il primo prototipo il 30/12/2023 mentre per la versione alpha il 10/02/2024.

In seguito alla pubblicazione del gioco in versione alpha sarà data la possibilità agli utenti di giocare gratuitamente per un periodo di tempo limitato e di mandare feedback riguardo al gameplay ed eventuali implementazioni e segnalarci eventuali bug.

1.2 Modello di processo

Per il processo di sviluppo del gioco si è deciso di sfruttare un approccio di tipo \*agile\*, infatti, si è ritenuto che questo fosse più adatto per un progetto di questo tipo in cui la documentazione non è essenziale per l'utente finale, ossia i giocatori.

Il processo di sviluppo sarà strutturato nel seguente modo:

* Fase di sviluppo della versione alpha

Durante questa fase verrà sviluppata una prima versione del gioco con tutte le funzionalità di base necessarie al gameplay.

* Fase di manutenzione e aggiornamento

Durante questa seconda fase ci si occuperà di correggere eventuali bug e implementare nuove funzionalità del gioco.

Gli aggiornamenti potranno essere di due tipi: aggiornamenti minori che correggono bug e/o implementano nuove funzionalità minori, e aggiornamenti maggiori che implementano nuove funzionalità principali.

Come piattaforma di Version Control System si è scelto di sfruttare GitHub, in quanto è la piattaforma più utilizzata per lo sviluppo software ed è nota anche dagli utenti, inoltre permette di sfruttare le funzionalità di issue tracking, branch e wiki.

1.3 Organizzazione del progetto

Le persone coinvolte nella progettazione del gioco inizialmente sono quelle che compongono il team. In seguito alla pubblicazione, i giocatori potranno prendere parte allo sviluppo suggerendo nuove funzionalità o segnalando bug, o, se lo desiderano, potranno contribuire direttamente allo sviluppo facendo un \*fork\* del repository, implementando le loro modifiche e aprendo una \*pull request\*. Affinché le modifiche entrino a far parte del gioco dovranno essere revisionate e approvate dal team di sviluppo.

1.4 Standards, linee guida e procedure

//TODO

Il gioco è composto da due applicativi separati:

- Server: Gestisce l'andamento di una o più partite, non possiede un'interfaccia grafica ed è compatibile con i più diffusi sistemi operativi (anche ARM)

- Client: Gestisce l'interfaccia grafica e l'interazione con l'utente

Sia server che client sono scritti in Java e per la comunicazione utilizzano la libreria open source [MCLib-Network](https://github.com/MultiCoreNetwork/MCLib) sviluppata da Lorenzo Magni e basata a sua volta sulla libreria [Netty](https://netty.io) per la comunicazione in rete tramite pacchetti TCP.<

Per la gestione dei file di configurazione e delle traduzioni si sfruttano file json mediante la libreria [Gson](https://github.com/google/gson) sviluppata da Google.<br>

Per la grafica si è deciso di utilizzare la libreria [JCEF](https://github.com/chromiumembedded/java-cef) che permette di integrare all'interno di una finestra Swing un browser \*Chromium\*. Si è deciso di utilizzare questo tipo di approccio in modo da semplificare lo sviluppo dell'interfaccia grafica mediante l'uso dei linguaggi web quali HTML, CSS e JavaScript e permettere una facile modifica (anche da parte degli utenti) delle interfacce del gioco. La scelta dei linguaggi web permette inoltre di avere un rescaling automatico delle interfacce in base alle dimensioni della finestra.<br>

Sempre per la grafica sono state utilizzate le librerie CSS/JS [Bootstrap](https://getbootstrap.com) e [jQuery](https://jquery.com) e [Font Awesome](https://fontawesome.com/) gestite localmente.

### 1.5 Attività di gestione

//TODO

Si è deciso di organizzare lo sviluppo secondo la filosofia dell'\*extreme programming\* in quanto permette di migliorare la qualità del codice e approcciarsi al cambiamento dei requisiti in modo più responsivo.

Sulla base di questo si è deciso di dedicare del tempo alla lettura del codice scritto dagli altri membri del team come metodo di validazione, inoltre si è deciso di applicare l'approccio del \*pair programming\* per la scrittura del codice, con il quale si hanno coppie di sviluppatori che lavorano insieme su un singolo frammento di codice e si dividono il lavoro di scrittura e revisione in tempo reale.

Gli incontri settimanali sono di breve durata, all'incirca 60 minuti, durante i quali possono essere effettuate delle sessioni di \*brain storming\* per trovare nuove idee o cercare soluzioni a problemi riscontrati

Al termine del meeting, ci si accorda sui compiti da portare avanti durante le settimane successive.

// Utilizzando extreme programming non abbiamo organizzato un calendario di incontri preventivo, ma ci siamo organizzati settimanalmente in base al tempo a disposizione del team.

1.6 Rischi

I principali rischi che si possono incontrare durante lo sviluppo di questo gioco con le modalità sopra descritte sono principalmente 2:

1. Il team potrebbe non essere in grado di raggiungere gli obiettivi prefissati entro la deadline. Il prolungamento della fase di programmazione corrisponde ad un aumento dei costi di sviluppo che nella fase iniziale non sono coperti.

2. Il gioco potrebbe non riscontrare l'interesse sperato da parte degli utenti o di potenziali investitori. In tal caso il tempo speso per lo sviluppo non potrà essere pagato.

1.7 Membri

La modalità di lavoro \*agile\* fa si che non ci sia una netta distinzione tra i ruoli nel gruppo di lavoro.

Viste le regole date dall’XP abbiamo evitato la suddivisione in ruoli del team (manager…), in modo da avere la stessa importanza.

Grazie ai differenti skill personali degli sviluppatori coinvolti entrambi i membri del team hanno dato un contributo attivo al progetto sfruttando al meglio le proprie competenze.

1.8 Metodi e tecniche

//TODO

/\*

* **StarUML** per quanto riguarda il design, nello specifico: Use Case Diagram, Class Diagram, Sequence Diagram e Activity Diagram e State Machine Diagram.
* Per la coordinazione del lavoro e dei file abbiamo utilizzato **GitHub**
* **Eclipse IDE** per la realizzazione e sviluppo del codice Java.

\*/

//loro hanno messo diagramma use case

1.9 Garanzie di qualità

Si punta a sviluppare un software che rispetti i parametri di qualità indicati dal modello di McCall:

- Correttezza

- Affidabilità

- Robustezza

- Integrità

- Usabilità

Per garantire la sicurezza degli utenti sono state usate librerie moderne e molto diffuse, aggiornate frequentemente e poco inclini ad avere vulnerabilità.

Per verificare la corretta funzione del programma, nella fase di testing abbiamo testato ogni possibile scenario di gioco immaginabile. Inoltre per controllare la qualità abbiamo condiviso il gioco a terzi per verificare la giocabilità e i possibili errori (bug) del software.

1.10 Package di lavoro

//TODO

Il progetto del software è basato su maven e diviso in 3 moduli: \*client\*, \*server\* e \*common\*. Questo permette di avere una struttura ben organizzata, di poter avere un codice comune su client e server necessario per la comunicazione e di non avere codice inutilizzato sui lati (client o server).<br>

Il client contiene tutta la parte di codice necessaria per la visualizzazione del modello (view) e l'input/output da e verso l'utente, il server invece ha la funzione di controller e gestisce il modello (controller/model).

1.11 Risorse

Le risorse designate allo sviluppo del gioco sono un project manager e due sviluppatori. Il team ha deciso di utilizzare l'IDE [IntelliJ Idea](https://www.jetbrains.com/idea/) per lo sviluppo del software, la suite Office e gli strumenti di GitHub per la scrittura della documentazione e della presentazione e altri strumenti di photoediting per la creazione e modifica delle immagini del gioco e di grafici e diagrammi.

Ogni membro del team ha a disposizione un computer personale, una connessione ad internet e un luogo di ritrovo fisico dove potersi incontrare. Per l'organizzazione del lavoro a distanza e dei meeting è stato anche sfruttato [Discord](https://discord.com/)

1.12 Budget e pianificazione

Come precedentemente specificato, il lavoro non è attualmente retribuito ma i possibili ricavi futuri verranno redistribuiti tra i membri del team in base al lavoro svolto.

Le ore di lavoro sono così suddivise:

1.13 Cambiamenti

Oltre alle features base incluse nella versione alpha, il team ha fissato come obiettivo l'aggiunta di funzionalità aggiuntive e nuove modalità di gioco per fornire un gioco più completo e variegato.

//TODO

- Traduzione del software in altre lingue oltre all'inglese e all'italiano

1.14 Consegna

Una volta pronta la versione alpha, prima di essere pubblicato, verrà inviata una segnalazione tramite issue a tutti i membri del team in modo tale da poter effettuare un'ultima revisione prima della pubblicazione.

2. Software lifecylce

Per il processo di sviluppo, il team ha scelto un approccio di tipo \*agile\* poiché meglio si adatta alla metodologia di lavoro desiderata:

- Si considera importante il gruppo, le abilità dei suoi membri e le loro interazioni. I lavori vengono assegnati in base alle capacità personali riducendo il tempo necessario allo studio di una competenza non conosciuta. Nel caso in cui un componente della squadra non sia in grado di portare a termine un task, il team si riunisce per trovare una soluzione.

- Nel team non c'è una struttura di tipo gerarchico, tutti i membri hanno la stessa importanza e possono esprimere le loro opinioni. Ciò rende il gruppo più coeso.

- Si considera più importante un prodotto software funzionante piuttosto che una documentazione estremamente elaborata. Ciò consente di dedicare più tempo allo sviluppo e ottenere un prodotto pubblicabile in minore tempo con un risparmio di risorse. Nel caso sia necessario, la documentazione potrà essere espansa o modificata in seguito.

- Il team è organizzato secondo la filosofia dell'\*extreme programming\* e durante lo sviluppo si è sfruttato spesso il \*pair programming\* come metodo di verifica. I programmatori in coppia hanno lavorato sullo stesso frammento di codice in modo tale da avere una verifica in tempo reale sul lavoro svolto. Questa pratica ha inoltre favorito lo scambio di idee e la ricerca di soluzioni migliori.

- Durante lo sviluppo della versione alpha si è anche sfruttata la tecnica del \*timeboxing\* per la suddivisione dello sviluppo a intervalli temporali entro i quali determinate funzionalità devono essere implementate.

- Si è scelto di dare molta importanza alla collaborazione con gli utenti, infatti dopo aver messo appunto le funzionalità base dell'applicazione, si è deciso di dare la possibilità agli utenti di collaborare mediante suggerimenti e feedback o anche attivamente allo sviluppo degli aggiornamenti. Si è tuttavia deciso di non permettere agli utenti di prendere decisioni organizzative e tutte le modifiche e proposte devono essere approvate dal team.

- Il team è favorevole al cambiamento e si impegna a dedicare il giusto tempo alla pianificazione del futuro del software. In caso di grande richiesta da parte degli utenti finali (o di finanziatori) si è ben disposti a modificare il software in accordo con le loro esigenze. Unico limite a ciò è la denaturazione dei principi di base del gioco.

Dal punto di vista della progettazione del software si è deciso di attenersi ai seguenti modelli:

- \*Model-Driven Architecture\*: prima di procedere alla scrittura del codice sono stati costruiti dei modelli che rappresentassero le features desiderate e la loro implementazione

- \*Data-Driven Architecture\*: dove possibile le impostazioni e le funzionalità del gioco sono state implementate per essere facilmente modificabili tramite file json quindi facilmente modificabili anche senza dover aggiornare o ricompilare il software (Maggiori implementazioni saranno disponibili in versioni successive alla alpha)

## 3. Configuration management

Tutto il lavoro svolto, che si tratti di documentazione o di codice, viene regolarmente salvato nel repository di GitHub in condivisione con tutti i membri del team.

### 3.1 Struttura del progetto

Il repository è strutturato nel seguente modo:

- \_\_branches\_\_:

- \*main\*: Contiene le versioni stabili del codice e la documentazione

- \*dev\*: Contiene le versioni in via di sviluppo del codice

- \_\_cartelle\_\_:

- \*code\*: Contiene il codice sorgente del progetto. È diviso in 3 moduli:

+ \*common\*: Modulo contenente le classi comuni a client e server

+ \*client\*: Modulo contenente le classi del client e le risorse grafiche

+ \*server\*: Modulo contenente le classi del server headless

- \*docs\*: Contiene la documentazione del progetto

- \*.github/workflows\*: Utilizzato per la configurazione dei workflow di GitHub Actions, testing, compilazione e distribuzione automatica

Durante lo sviluppo sono stati sfruttati anche altri branch, per la sperimentazione di tecnologie differenti o per la scrittura dei testing JUnit in parallelo allo sviluppo del codice. Questo è stato fatto per ridurre i conflitti che sarebbero potuti sorgere in un singolo branch. Questi branch una volta completata la loro funzione sono stati uniti ad altri branch o eliminati.

### 3.2 Issues

Nei vari meeting settimanali vengono creati i macro-temi che dovranno essere sviluppati nel corso della settimana seguente. Le varie attività sono create come issue, aggiungendo una breve descrizione del lavoro da svolgere, assegnandoli alla persona che si occuperà di quel lavoro e i relativi tag. Durante la settimana i membri del team sono liberi di creare nuove issue in base alle necessità.<br>

Le issue sono suddivise in 3 categorie principali:

- \*task\*: attività da svolgere

- \*bug\*: segnalazione di un bug

- \*enhancement\*: proposta di miglioramento

A seconda del loro avanzamento, i task possono trovarsi in diversi stati. Per tenere traccia di ciò è stata utilizzata una Board sulla piattaforma Monday.<br>

Gli stati sono i seguenti:

- \*Not started\*: L'attività non è ancora stata iniziata

- \*Working on it\*: L'attività è in fase di sviluppo

- \*To test\*: L'attività è stata completata e deve essere testata

- \*Testing\*: L'attività è in fase di testing che comprende la scrittura di test JUnit o la verifica manuale

- \*To fix\*: L'attività è stata testata ma non funziona correttamente

- \*Done\*: L'attività è stata completata e funziona correttamente

## 4. People management

L'azienda ha deciso di basarsi sun una struttura organizzativa non gerarchica. Grazie a ciò ha una maggiore flessibilità e può adattarsi più facilmente alle esigenze del progetto. Inoltre, tale struttura non ha una suddivisione prefissata del lavoro, è il team stesso a decidere come dividere e assegnare i vari task e, in caso di difficoltà, più membri possono affiancarsi per risolvere la problematica in gruppo. Il vantaggio di utilizzare l'adhocrazia permette anche di avere un clima più sereno e informale, utile per il benessere dei membri del team.<br>

Anche se non esite una distinzione tra i ruoli scritta, si è venuta a creare in maniera automatica la seguente suddivisione:

<table>

<tr>

<th></th>

<th>Documentazione</th>

<th>Progettazione</th>

<th>Sviluppo back-end</th>

<th>Sviluppo front-end</th>

<th>Test</th>

</tr>

<tr>

<th>Lorenzo Magni</th>

<td></td>

<td>✅</td>

<td>✅</td>

<td>✅</td>

<td>✅</td>

</tr>

<tr>

<th>Marianna Romelli</th>

<td>✅</td>

<td>✅</td>

<td></td>

<td></td>

<td>✅</td>

</tr>

<tr>

<th>Saif Bouchemal</th>

<td></td>

<td></td>

<td></td>

<td>✅</td>

<td>✅</td>

</tr>

</table>

## 5. Software quality

Come precedentemente specificato, il team si è prefissato di sviluppare un'applicazione che rispetti i parametri e gli attributi di qualità definiti da McCall-Richards-Walters nel documento redatto nel 1977. Di seguito sono elencati suddivisi in categorie come indicato dagli autori.

### 5.1 Operatività del software

- \_\_Correttezza\_\_: L'applicazione soddisfa i requisiti e le specifiche stabilite a inizio lavoro. Le funzionalità non ancora implementate lo saranno in futuri aggiornamenti del software.

- \_\_Affidabilità\_\_: L'applicazione è stata testata mediante strumenti e metodologie diverse e presenta il minor numero di bug possibile.

- \_\_Efficienza\_\_: Le risorse utilizzate dal client sono limitate, non vi è la necessità di un hardware potente o una grande quantità di RAM e il peso dell'applicazione è limitato a pochi MB. Il protocollo di comunicazione con il server è altrettanto leggero di conseguenza non richiede una connessione particolarmente veloce e una grande quantità di dati. Questo è stato ottenuto anche grazie allo spostamento del carico sul server.

- \_\_Integrità\_\_: Il software è sicuro in quanto non vengono sfruttati servizi esterni, non vengono raccolti dati personali e la chat è criptata end-to-end con una chiave dinamica.

- \_\_Usabilità\_\_: Il prodotto è semplice da utilizzare, infatti non sono richieste particolari abilità per poterne usufruire. L'uso dell'applicazione è facilitato su tutte le piattaforme grazie all'utilizzo di librerie specifiche per ogni sistema operativo in modo da sfruttare una grafica semplice e uniforme su tutte le piattaforme. Unico requisito (prima della pubblicazione su piattaforme di distribuzione) è l'installazione manuale di Java 17 o versioni successive.

I requisiti base del software sono i seguenti:

- Avere un pc con sistema operativo Windows, Linux o MacOS

- Avere una connessione a internet

- Avere una scheda video (anche integrata)

- Avere una conoscenza minima del gioco di ruolo a cui ci si sta per approcciare

### 5.2 Revisione del software

- \_\_Manutenibilità\_\_: La fase di individuazione degli errori è semplificata dalla scelta di separare i lati server client e dall'introduzione di un sistema di logging attivabile avviando il programma in modalità di debug. Il processo è inoltre facilitato dal fatto che il codice è strutturato secondo il paradigma \*model-view-controller (MVC)\* grazie al quale è possibile focalizzarsi su una singola parte del codice in base alla provenienza del bug.

- \_\_Testabilità\_\_: Tutte le funzionalità incluse sono testabili in qualunque momento tramite test manuali (mediante l'uso dei logger) e automatici.

- \_\_Flessibilità\_\_: Come per la manutenibilità, la scelta della struttura del progetto semplifica la fase di modifica o implementazione di nuove funzionalità rendendolo molto flessibile.

### 5.3 Transizione verso un nuovo ambiente

- \_\_Portabilità\_\_: Il gioco è nativamente eseguibile sui più diffusi sistemi operativi desktop (Windows, Linux, MacOS) dotati di una connessione internet e un hardware relativamente recente. Il server è in grado di funzionare su qualsiasi tipo di dispositivo senza particolari limiti. La scelta di basare la grafica su linguaggi web è anche vantaggiosa nel caso in cui si volesse portare il gioco su altre piattaforme quali smartphone e tablet, console o in versione web senza la necessità di effettuare troppe modifiche.

- \_\_Riusabilità\_\_: Sono state sfruttate e scritte librerie ad hoc per la scrittura dell'applicazione. Tali librerie possono essere riutilizzate in altri ambiti in quanto sono state sviluppate in modo indipendente dal contesto del gioco.

## 6. Requirement engineering

La fase iniziale dello sviluppo è stata dedicata alla definizione dei requisiti del software richiesti da parte del committente. Dal momento che la produzione del gioco non è stata commissionata bensì è stata un'idea spontanea, i requisiti sono stati definiti in base a quello che era ritenuto importante per il team e alla conoscenza del gioco di ruolo dal quale il software ha origine.<br>

Al contrario, in seguito alla prima pubblicazione, i requisiti potranno essere ridefiniti o modificati anche sulla base delle critiche e dei suggerimenti dei giocatori. È già stata preventivata la possibilità di effettuare eventi in diretta sulle piattaforme social durante i quali i giocatori avranno modo di interagire direttamente con il team di sviluppo e discutere insieme a loro del gioco e di eventuali modifiche da apportare. Un evento di questo tipo permetterebbe di svolgere in un ambiente informale le fasi di \*elicitation\* e \*negotiation\* necessarie per la definizione di nuovi requisiti (\*requirements specification\*) che verrebbe poi conclusa in una riunione del team.<br>

Oltre a quanto già detto, il project manager ha definito i requisiti sulla base del modello MoSCoW come segue:

- \_\_Must Have\_\_:

- Sviluppo di una versione base del gioco funzionante

- Inserimento della traduzione italiana e inglese

- Multigiocatore online

- Interfaccia grafica semplice e intuitiva

- Chat testuale

- Grafica diversificata per le fasi giorno e notte

- Stabilità e sicurezza del software

- \_\_Should have\_\_:

- Traduzione in altre lingue

- Guida al gioco interna all'applicazione

- Chat testuale crittografata e moderata

- Autenticazione degli utenti per fornirgli un nickname fisso e statistiche sui risultati delle partite

- Consentire la modifica delle impostazioni della partita

- Timeout delle lobby dopo un periodo di inattività

- Impostazioni di gioco

- Effetti sonori

- Aggiornamento automatico

- Chat vocale

- \_\_Won't have\_\_:

- Impostazioni per modificare la grafica

- Doppia autenticazione tramite codice OTP inviato al numero di telefono

- Lobby pubbliche

## 7. Modelling

### 7.1 Grafica

Durante la fase di studio dell'interfaccia grafica, il punto di maggiore interesse era di ottenere un risultato che potesse essere gradevole e allo stesso tempo semplice da usare per tutti gli utenti. Si è optato per la creazione di diverse scene alternate in base alle necessità.

<p align="center">

<img src="resources/state\_diagram.png">

</p>

Durante il primo avvio del client mancheranno le librerie native necessarie per la visualizzazione della grafica, in questo caso si aprirà una finestra nella quale è mostrato l'avanzamento del download al termine del quale l'interfaccia grafica standard verrà mostrata. L'utente riceverà il benvenuto con una breve schermata di caricamento, durante la quale il gioco carica tutti i dati necessari per il funzionamento e verificherà la presenza di aggiornamenti. L'utente si troverà quindi all'interno del menù principale dal quale potrà accedere alle varie funzionalità del gioco o uscire. Se l'utente decide di creare una nuova partita o di unirsi ad una già esistente verrà inviato nella schermata della lobby nella quale potrà vedere l'elenco degli altri giocatori e le regole della partita. Una volta avviata tutti gli utenti della partita saranno inviati alla schermata con il gioco al cui termine verrà mostrato il risultato e avranno modo di tornare al menù principale o alla lobby per ricominciarne una nuova. Fatta eccezione per le impostazioni a cui si può accedere anche durante il gioco e che quindi, quando chiuse, riportano alla schermata precedente, da tutte le scene è possibile sempre ritornare al menù principale.

<table>

<tr>

<td><img src="resources/main\_menu.png"></td>

<td><img src="resources/lobby.png"></td>

</tr>

<tr>

<td><img src="resources/caricamento.png"></td>

<td><img src="resources/game\_day.png"></td>

</tr>

</table>

Dopo aver stabilito come sarebbe dovuta essere la grafica, è stato possibile stabilire il comportamento che avrebbe dovuto avere in base alle azioni degli utenti. Di seguito è riportato l'\*activity diagram\* di un utente che chiede di unirsi ad una lobby.

<p align="center">

<img src="resources/activity\_diagram.png">

</p>

All'interno del diagramma sono riportati i principali controlli che vengono svolti (in parte dal client, in parte dal server) per permettere di passare dalla schermata del menù a quella del gioco. Tutti gli errori vengono mostrati all'utente sotto forma di \*dismissable modal\* ovvero una finestra che si apre in primo piano oscurando lo sfondo e che è chiudibile dall'utente.

### 7.2 Funzionamento del back-end

Una volta noti i requisiti e scelta la grafica, si è passati alla fase di progettazione del codice del software. Il progetto è stato abbozzato inizialmente con dei \*class diagram\* specifici per il client e per il server che mostrassero le classi da implementare e le loro relazioni. Durante la fase di sviluppo questi diagrammi sono stati modificati più volte per adeguarli alle esigenze sorte, alle modifiche effettuate ai requisiti e per aggirare dei vincoli posti dal linguaggio di programmazione scelto.<br>

I diagrammi seguenti mostrano la struttura di classi che è stata implementata al termine dello sviluppo della versione alpha.<br><br>

\_\_Server\_\_

<p align="center">

<img src="resources/class\_diagram\_server.png">

</p>

\_\_Client\_\_<br>

Struttura classi principale del client

<p align="center">

<img src="resources/class\_diagram\_client\_1.png">

</p>

Classi responsabili all'invio di dati e comandi tramite json all'interfaccia grafica (Java -> JavaScript)

<p align="center">

<img src="resources/class\_diagram\_client\_2.png">

</p>

Classi responsabili alla ricezione di dati e comandi tramite json dall'interfaccia grafica (JavaScript -> Java)

<p align="center">

<img src="resources/class\_diagram\_client\_3.png">

</p>

Come si evince dai diagrammi, pur avendo gran parte del carico computazionale, il server ha una struttura molto più semplice rispetto al client. La gestione della grafica ha richiesto un numero di oggetti non indifferente per gestire comodamente la comunicazione della grafica. Inoltre gran parte delle classi necessarie alla comunicazione tra client e server è stato racchiuso nel modulo \*common\* usato come libreria in entrambi i lati.<br><br>

La comunicazione tra client e server è effettuata tramite l'invio di pacchetti tcp/ip. Gli oggetti contenuti nei pacchetti vengono convertiti in un array di bit al quale viene anteposto un id che identifica il tipo di pacchetto e la lunghezza del pacchetto, in modo che possa essere ricostruito se diviso da tcp. Giunti a destinazione i pacchetti sono riconosciuti e riconvertiti da array di bit a oggetti Java.<br>

Di seguito è riportato un \*sequence diagram\* che mostra il comportamento del software e della comunicazione client-server dello svolgimento di una partita da quando viene avviata fino al suo termine.

<p align="center">

<img src="resources/sequence\_diagram.png">

</p>

Nella fase iniziale il click dell'utente su un pulsante fa si che il server avvii la partita aggiornando il client delle modifiche effettuate al model e aggiornando la grafica in accordo. Durante lo svolgimento del gioco il server aggiorna i client informandoli delle modifiche al modello o dello stato di avanzamento del gioco, questa operazione viene effettuata in un thread separato di conseguenza, il server non deve attendere la risposta dei client per continuare a gestire la partita (risposta che si limita ad un semplice ack tcp). Caso differente si ha quando si giunge in una fase in cui è necessaria l'azione di uno o più utenti per proseguire. In quel caso il server si mette in attesa della risposta dei giocatori che arriva mediante un pacchetto al \*ServerPacketListener\*.

## 8. Software architecture e design

Il software è stato progettato sfruttando il pattern \*MVC\* che permette di separare le responsabilità di gestione dei dati, della grafica e della logica del programma. Il pattern è stato applicato sia al client che al server con la seguente divisione dei compiti:

- Il client gestisce la grafica (view) e le comunicazioni con il server e gli input dell'utente.

- Il server gestisce la logica del gioco (controller) e mantiene le informazioni riguardanti le partite in corso (model), nonché eventuali iterazioni con database per la gestione di login.

Sulla base dei concetti di astrazione e modularità, il software è stato progettato in modo da avere una bassa dipendenza tra i vari moduli. Per fare ciò si è scelto di evitare l'uso di variabili globali condivise tra moduli diversi e di non utilizzare all'interno dei moduli codice di altri. Inoltre per far fede al principio dell'\*external coupling\* si è affidato al client la gestione dell'interfaccia onde possibile, spostando l'incarico al server soltanto quando necessario.<br>

La vista tramite connettori e componenti del sistema creatosi è quindi quella riportata di seguito.

<p align="center">

<img src="resources/connections.png">

</p>

Come si può vedere dal precedente grafico e dai class diagram è stata creata un'architettura modulare con una bassa dipendenza tra i vari moduli. Le interazioni tra di essi sono gestite da delle interfacce. Il codice è strutturato in modo da migliorarne la leggibilità e soprattutto la manutenibilità. È tuttavia necessario specificare che la struttura del server è attualmente monolitica, infatti esso gestisce tutti i servizi di interazione con gli utenti su un unico processo. In futuro si pianifica di dividere questi compiti in un'architettura basata su micro-servizi.

## 9. Testing

Durante lo sviluppo il corretto funzionamento del codice è stato testato costantemente con test pratici di esecuzione del programma e delle sue funzionalità. Questo processo è stato aiutato particolarmente dalla scelta di inserire all'interno del codice un gran numero di messaggi di debug (attivabili mediante il parametro di avvio `-d`) che mostrano in molti momenti lo stato del model, e le azioni che vengono svolte da giocatori e dal server.<br>

Nel caso in cui durante lo sviluppo si fosse notata la presenza di un bug, si è proceduto alla sua correzione, onde possibile, e si è poi ripetuto il test per verificare che il bug fosse stato corretto. In alcuni casi, insieme alla correzione del bug è stato necessario fare anche delle modifiche più sostanziali al programma con un conseguente \*refactoring\* del codice per adattarlo alle nuove funzionalità.<br>

Una volta scritto lo scheletro del codice e ottenuta una versione più o meno definitiva/funzionante del gioco sono stati implementati dei test automatici mediante JUnit per poter verificare il corretto funzionamento del codice anche in presenza di casi limite. La scrittura di questo tipo di test è, tuttavia, stata resa più complessa dalla necessità di simulare l'interazione tra client e server. Questo non è stato possibile per il rispetto della deadline e quindi è stato programmato per il futuro.<br>

Per fare ciò si farà uso del framework \*Mockito\* che permette di simulare l'interazione tra oggetti Java. In particolare si sarà possibile simulare il comportamento del client e del server tramite e delle loro interazioni e testare quindi il comportamento dei moduli legati al networking.