

**Progetto di ingegneria del software, anno accademico 2022-2023**

**Anno accademico 2022-2023**

**Membri del team**

* Lorenzo Magni, matricola 1073257
* Saif Bouchemal, matricola 1074800
* Marianna Romelli, matricola 1072382

**Sommario**

**Il nostro progetto**

Project plan

Introduzione

Modello di processo

Organizzazione del progetto

Standards, linee guida e procedure

Attività di gestione

Rischi

Membri

Metodi e tecniche

Garanzie di qualità

Package di lavoro

Risorse

Budject e pianificazione

Cambiamenti

Consegna

Software Life Cycle

Configuration Management

Struttura del progetto

Issue

Branch

People Management

Software quality

Parametri riguardanti l’operatività del software

Parametri riguardanti la revisione del software

Parametri riguardanti la transizione verso un nuovo ambiente

Requirement Engineering

Funzionali e non funzionali

Modelling

State machine diagram

Use case diagram

Class diagram

Sequences diagram

Activity diagram

Software Architecture

Software Design

Software Testing e manutenibilità

Applicazione Web

**Il nostro progetto**

Abbiamo creato un gioco online disponibile su desktop chiamato Lupus in tabula.

Lo scopo del gioco per i villaggeri è quello di ucidere i lupi, mentre quello dei lupi è quello di mangiare i villaggeri.

**I personaggi**

Narratore, Cittadini, Lupi Mannari, Veggente, medium (da 10 giocatori in su), guardia del corpo (da 12 giocatori in su), indemoniato (da 15 giocatori in su), massoni (da 15 giocatori in su), gufo (da 15 giocatori in su), criceto mannaro (da 15 giocatori in su), mitomane (da 15 giocatori in su)

**I ruoli**

Narratore: chiama il giorno e la notte e durante la notte chiama i ruoli e dirige

Cittadini: durante il giorno decidono chi linciare e di notte dormono

Lupi mannari: di solito sono 2 se in 15+ diventano 3, di notte decidono chi uccidere

Veggente: di giorno è un cittadino normale ma di notte può scegliere qualcuno e vedere se è un lupo

Medium: di notte può vedere se un morto è o no un lupo

Guardia del corpo: dalla seconda notte può indicare qualcuno e proteggerlo dai lupi

Indemoniato : è un cittadino comune ma patteggia per i lupi

Massoni: SOLO la prima notte si riconosceranno e si difenderanno a vicenda

Gufo: ogni notte indicherà una persona e la manda allo spareggio se si gioca in 20 o più sarà direttamente uccisa

Criceto mannaro: di notte non può essere ucciso dai lupi ma se la veggente ne scopre il ruolo lo uccide

Mitomane: dalla seconda notte può indicare qualcuno e se becca un lupo o un veggente ne diventa una copia

**Lo svolgimento**

Il gioco prevede due fasi che si alternano, il giorno e la notte. Durante il giorno i cittadini discutono su chi dovranno linciare, di notte il narratore chiamerà tutti i giocatori che avranno un tempo prestabilito per svolgere i propri ruoli.

I Cittadini vincono se eliminano i lupi, i lupi vincono se sono in ugual numero con i cittadini, l’indemoniato vince se vincono i lupi, il Criceto Mannaro vince se arriva vivo alla fine.

**Il gioco**

Il gioco si divide in due fasi: la notte e il giorno. Di notte i lupi mannari sbranano un giocatore. Di giorno si discute e si lincia qualcuno, sperando che sia un lupo mannaro. Tra gli umani si cela un veggente che di notte è in grado di sondare la mente di un giocatore, e capire se si tratta di un lupo mannaro o meno.

*La notte*

Il moderatore dichiara l’inizio della notte (“è notte, chiudete tutti gli occhi”). Tutti i giocatori chiudono gli occhi.  
Per coprire eventuali altri rumori, di notte tutti i giocatori battono ritmicamente una mano sul tavolo. Il moderatore chiama ad alta voce il **veggente** (per es. “*il veggente apre gli occhi e sceglie una persona*”).  
Il **veggente apre gli occhi** e indica con un gesto un giocatore a sua scelta. Il moderatore gli risponde con un cenno del capo affermativo **se il giocatore indicato è un lupo mannaro**, negativo altrimenti. Il moderatore quindi chiama la fine della fase del veggente (“*il veggente chiude gli occhi*”), il quale **richiude gli occhi**. **Nota:** questa fase va giocata anche se il veggente è un fantasma, per non dare indizi agli altri; in questo caso il moderatore chiama il veggente e, poco dopo, la fine della fase senza che il veggente gli risponda. Il moderatore quindi chiama i **lupi mannari** (“*i lupi mannari aprono gli occhi e scelgono una persona da sbranare*”). I **lupi mannari aprono gli occhi** e si riconoscono. Dopodiché si mettono rapidamente e silenziosamente d’accordo su **quale giocatore sbranare**, indicandolo con un cenno.  
Il moderatore prende atto della decisione e chiama la fine della fase dei lupi mannari (“*i lupi mannari chiudono gli occhi*”), i quali **richiudono gli occhi**. A questo punto la notte è terminata.

*Il giorno*

Il moderatore dichiara ora l’inizio del giorno (“*è giorno, aprite tutti gli occhi*”). Tutti i giocatori **aprono gli occhi**. Il moderatore dice al giocatore scelto dai Lupi Mannari che è stato sbranato! Questo giocatore da ora in poi non gioca più come personaggio ma **come Fantasma**! Egli deve **astenersi da commenti** e **non può più parlare** per il resto della partita. Egli riceve una carta fantasma, e il “Benvenuto!”. (Ricordate che non può rivelare il proprio personaggio!).  
I giocatori devono ora scegliere qualcuno da **linciare**; possono parlare fra loro per identificare chi siano i lupi mannari attraverso ipotesi, deduzioni e contraddizioni. I lupi mannari, mescolati tra i superstiti, tenderanno a convogliare i sospetti verso altri villici o verso il giocatore che reputano il veggente. Si può mentire liberamente, ma non si può **mai mostrare la propria carta** agli altri!

***Il linciaggio***

Dopo un massimo di tre minuti di discussione, il moderatore chiede a ogni giocatore, partendo da chi è a sinistra di quello che ha il “Benvenuto!” e procedendo in senso orario, chi secondo lui vada linciato. Tutti i giocatori, **inclusi i fantasmi**, prendono in mano il gettone voto, e al proprio turno votano mettendo il gettone davanti al giocatore che vogliono linciare (in questa fase, il lato su cui viene giocato l’indizio non ha importanza). I due giocatori con più voti (cioè con più gettoni davanti) **sono indiziati**. Eventuali casi di parità vengono risolti scegliendo chi è più vicino al giocatore col “Benvenuto!”, in senso orario.

Ogni giocatore riprende il proprio gettone voto. Il moderatore mette davanti ai due indiziati un indizio di licantropia a testa: uno riceve l’impronta, l’altro il ciuffo di peli.

I due giocatori indiziati possono difendersi con un ultimo breve discorso. Terminate le arringhe, i giocatori **non indiziati** e **ancora vivi** (esclusi quindi gli indiziati e i fantasmi) votano di nuovo il giocatore tra gli indiziati che verrà linciato. Questa votazione avviene in segreto e in contemporanea.

I giocatori posano sul tavolo il gettone voto col **lato che corrisponde all’indizio di licantropia del giocatore che vogliono linciare verso l’alto**, e lo coprono con la mano. Quando tutti hanno scelto, i voti vengono rivelati allo stesso tempo, al comando del moderatore. **Chi ha preso più voti è linciato e diventa un fantasma!** In caso di parità viene linciato il più vicino in senso orario a chi ha il “Benvenuto!”.

Il moderatore recupera gli indizi, e dà al giocatore linciato una carta fantasma. Il giocatore linciato deve **astenersi dal commentare** la decisione, e **non può più parlare** fino alla fine della partita (né rivelare il proprio personaggio!).

A questo punto il giorno è terminato: si ricomincia con una nuova notte e così via, finché una fazione non vince.

**Project Plan**

Sviluppiamo l’applicativo per una community generica di giocatori. Inizialmente i requisiti li decidiamo noi e nel momento in cui l’applicazione sarà resa pubblica, gli utenti avranno modo di fare proposte di modifiche e miglioramenti tramite varie piattaforme social che potranno essere ragiunte dall’interno dell’applicazione.

La nostra azienda è una società di sviluppo software costituita da Lorenzo Magni, Saif Bouchemal e Marianna Romelli al fine di progettare l’intera applicazione iniziale, in modo che possa poi essere il prototipo per possibili migliorie richieste dagli utenti o finanziatori.

Il fine del nostro lavoro è di mettere sul mercato un prototipo del gioco per possibili futuri investitori per ricevere un eventuale finanziamento.

Per quanto riguarda il compenso salariale, inizialmente non ci sarà e il nostro lavoro sarà finanziato successivamente dal possibile investitore. Saremo disposti a prendere in considerazione requisiti aggiuntivi richiesti dagli investitori, se necessario, al fine di migliorare o modificare l’applicazione come richiesto.

La community sarà formata dagli utenti che hanno avuto l’opportunità di usufruire del nostro gioco.

1. **Introduzione**

Dopo un’attenta analisi dei requisiti scritti da noi mettendoci nei panni di un possibile utente, di seguito saranno riportati i passi seguiti al fine di realizzare una prima versione del sistema richiesto.

Alla progettazione del sistema e allo sviluppo del primo prototipo lavorerà tutto il team composto da Lorenzo, Saif e Marianna. Successivamente saranno precisati i ruoli specifici per ognuno.

Dopo un primo incontro si decide di sviluppare fisicamente un’applicazione che implementi tale funzionalità:

- Sviluppo del gioco nella versione base funzionante

- Supporto della lingua inglese e della lingua italiana

- Multigiocatore remoto

- Interfaccia grafica

- Chat testuale

Abbiamo stabilito che la prima versione sarà disponibile per la fine di dicembre 2022, quindi abbiamo all’incirca 70 giorni.

Ci saranno degli incontri settimanali durante la fase di progettazione del software in modo da collaborare e discutere insieme, nonché avere un feedback dell’intero gruppo di lavoro.

Entro fine dicembre renderemo il prototipo disponibile online in modo che gli utenti possano iniziare ad utilizzarlo ed a mandarci alcuni requisiti che vorrebbero implementare.

Entro metà dicembre metteremo il prototipo online in modo che gli utenti possano iniziare ad utilizzarlo ed a mandarci alcuni requisiti che vorrebbero implementare.

1. **Modello di processo**

Per il processo di sviluppo del prototipo abbiamo optato per un approccio di tipo Agile, infatti, puntiamo a rilasciare una versione di base del prodotto e integrarla mano a mano con piccoli aggiornamenti che porteranno con sé le funzionalità descritte sopra, in modo da facilitare anche le fasi di testing e debugging.

Fondamentalmente il processo di sviluppo sarà guidato dalle nostre idee e successivamente verranno integrate le richieste della community o di un eventuale finanziatore.

Per tenere traccia dello sviluppo delle funzionalità richieste, optiamo per l'utilizzo delle issue su Github, che saranno assegnate settimanalmente a ogni componente del gruppo in base a quello che si decide nei meeting. Mano a mano che ognuno andrà a completare i punti che gli sono stati assegnati, le issue verranno chiuse e documenti e codice pubblicati sulla piattaforma.

1. **Organizzazione del progetto**

Le persone coinvolte nel progetto sono inizialmente solo il team di progettisti/sviluppatori e in seguito anche gli utenti della community.

L'intera documentazione riguardante il progetto sarà privata e disponibile per investitori su GitHub. In particolare, il team si incontra settimanalmente per analizzare il lavoro svolto dall'ultimo meet e pianificare le attività future, tracciando i task come issue su GitHub in modo che ogni persona del team sia al corrente di cosa stiano facendo gli altri.

1. **Standards, linee guida e procedure**

Il team ha deciso di sviluppare un'applicazione per implementare il sistema descritto. L'applicazione sarà suddivisa in due parti:

○ Client: si occupa dell'interfaccia grafica e di interagire con il server;

○ Server: gestisce l’andamento di una o più partite/giochi.

Per la comunicazione client - server abbiamo sfruttato la tecnologia basata

su socket mediante l’utilizzo delle librerie MCLib-Network (sviluppata da noi) e Netty.

Per quanto riguarda invece la grafica del client abbiamo deciso di utilizzare JCEF, ovvero l’integrazione in Swing di un browser Chromium, in modo da semplificare lo sviluppo dell’interfaccia, permetterne una rapida modifica (anche da parte degli utenti) e gestire comodamente il rescaling dell’interfaccia.

1. **Attività di gestione**

Abbiamo deciso di organizzarci secondo la filosofia della programmazione estrema, cioè (l’extreme programming). Ad esempio, la lettura del codice da parte dei colleghi, è un metodo di test molto efficace, dunque nel metodo che utilizziamo noi lo si fa: due programmatori lavorano insieme sullo schermo di un computer. Uno fa la codifica, l'altro guarda alle spalle, dà consigli, nota piccoli errori, fa domande e così via. In qualsiasi momento, possiamo cambiare i ruoli.

Gli incontri saranno settimanali e relativamente brevi, della durata di un'ora circa. Durante gli stessi potranno essere effettuate sessioni di brainstorming per condividere idee e risolvere problemi.

Il team di sviluppo si pone l'obiettivo di incontrarsi settimanalmente per aggiornarsi sullo stato dei lavori. In base alle esigenze si decide se effettuare la riunione virtualmente o se è necessario incontrarsi e svolgerla di persona.

Ad ogni incontro, i membri del gruppo espongono brevemente quali sono stati i punti portati avanti e quali sono state le problematiche incontrate sul proprio cammino. Mano a mano che si procede nell'esposizione, mediante l’utilizzo di issue, branch e commit teniamo traccia del lavoro svolto.

Alla fine del meeting ci si accorda su quali sono i punti da portare avanti; sarà il project manager ad assegnare a ciascun membro del gruppo, se stesso compreso, i compiti che dovranno essere eseguiti prima del prossimo incontro.

1. **Rischi**

La durata è di 70 giorni e l’obbiettivo è quello di arrivare al termine con tutta la documentazione e il prototipo completo. In particolare, abbiamo stabilito come scadenza il 31 dicembre per la consegna dell’applicazione con le prime funzionalità così che la community possa testarla ed il 15 gennaio per l’applicazione implementata con le modifiche effettuate successivamente dalle richieste degli utenti.

Essendo questo progetto inizialmente non commissionato abbiamo deciso di finanziarlo mediante una campagna di crowdfounding su kickstarter. Tali fondi sono necessari per sostenere i costi dello sviluppo e vi è il rischio che siano perduti in caso di mancato acquisito o finanziamento successivamente alla pubblicazione.

1. **Membri**

Utilizzando una modalità di sviluppo Agile, non c'è una vera e propria distinzione fra i ruoli nel gruppo. Inoltre, i task, decisi in comune accordo con il project manager, verranno inseriti come issue su Github e designati ad un programmatore. Nonostante questo, in una panoramica più generale possiamo definire i seguenti ruoli:

○ Project Manager e responsabile della comunicazione: Marianna Romelli

○ Progettista Software: Lorenzo Magni

○ Back-end e Front-end Developer: Lorenzo Magni e Saif Bouchemal

○ Tester: Lorenzo Magni, Saif Bouchemal e Marianna Romelli

1. **Metodi e tecniche**

Il team inizialmente si incontra per creare lo USE-CASE DIAGRAM in modo da comprendere le funzionalità necessarie per la nostra applicazione.

Svolgiamo poi il SEQUENCE DIAGRAM per alcune parti della nostra applicazione per comprendere meglio come dovrà essere implementato il lavoro. Anche per quanto riguarda il STATECHART DIAGRAM lo utilizziamo come base per il nostro lavoro di sviluppo.

1. **Garanzie di qualità**

Il team 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 utilizziamo librerie moderne e molto diffuse, aggiornate frequentemente e poco inclini ad avere vulnerabilità.

Inoltre, dividiamo il lato client da quello server, cosicché in futuro sarà possibile aggiornare separatamente le due parti.

Puntiamo a sviluppare un software che, oltre a rispettare i quality assurance, sia anche sicuro.

Il nostro software non raccoglie dati personali e la chat di gioco sarà criptata end to end.

1. **Package di lavoro**

Il protocollo di lavoro sarà costituito da tre moduli principali: il lato client, il lato server e una parte comune. In questo modo il modulo relativo all’interfaccia grafica sarà presente nel codice del client, la parte di gestione del modello sul server e il modulo comune sarà sfruttato come libreria su entrambi i lati.

1. **Risorse**

Le risorse designate al progetto sono un project-manager e due sviluppatori che lo affiancano. Il team utilizza IntelliJ per la scrittura di codice e per condividere con tutto il team il lavoro svolto si utilizza GitHub. Anche per quanto riguarda l'assegnazione dei compiti e la segnalazione di bug o problemi usiamo le issue di Github. In questo modo abbiamo un unico servizio su cui lavorare.

Al team vengono forniti 3 computer, una connessione internet, una stanza ed un ufficio per collaborare. Il team può decidere di incontrarsi di persona per discutere del progetto e per collaborare in ufficio, oppure può decidere tranquillamente di svolgere il proprio lavoro da casa collaborando attraverso Discord.

1. **Budget e pianificazione**

Al momento la suddivisione delle ore è raccolta privatamente e verrà resa pubblica in seguito.

1. **Cambiamenti**

Oltre alle feature base incluse inizialmente nel prodotto e descritte al punto uno di questo documento, l'obiettivo sarà quello di mettere le basi per ulteriori funzionalità allo scopo di fornire un servizio più completo, in accordo con la community. Queste verranno inserite nelle successive versioni del software e includeranno:

- Altre lingue oltre all’italiano e all’inglese

- Guida del gioco per una comprensione migliore

- Chat vocale

- Dare la possibilità al master del gioco di decidere i ruoli disponibili per i giocatori

1. **Consegna**

Gli incontri sono fissati ogni settimana, il giorno può variare in base agli impegni, ma abbiamo comunque stabilito una frequenza settimanale in modo da rimanere aggiornati con lo sviluppo e concordare insieme i prossimi passi da svolgere. Per quanto riguarda la prima settimana di lavoro abbiamo deciso di trovarci tre volte per concordare come procedere e come suddividere il lavoro.

**Software Life Cycle**

Per il processo di sviluppo il team ha scelto un approccio di tipo Agile poiché meglio si adatta alla nostra metodologia di lavoro:

* Consideriamo importante il team, le abilità e le interazioni tra di noi. I lavori vengono assegnati in base alle capacità che ognuno possiede. Se un componente della squadra si accorge che il compito va oltre le sue competenze, avvisa gli altri e se ne discute insieme.
* Nel team non c’è una struttura di tipo gerarchico: ci consideriamo tutti allo stesso livello. Questo ci porta ad essere più compatti e motivati.
* Consideriamo più importante un prodotto software funzionante che spendere troppo tempo sui documenti: a sviluppo iniziato, se ci accorgiamo che ci sono punti da implementare che non avevamo in precedenza considerato, li andiamo ad aggiungere poi alla documentazione.
* Abbiamo deciso di organizzarci secondo la filosofia della programmazione estrema, cioè (l’extreme programming). Due programmatori lavorano insieme sullo schermo di un computer: uno fa la codifica, l'altro guarda alle spalle, dà consigli, nota piccoli errori, fa domande e così via. In qualsiasi momento, possiamo cambiare i ruoli.
* Durante lo sviluppo del software utilizziamo la tecnica del timeboxing: suddividiamo lo sviluppo del codice in intervalli temporali entro i quali dobbiamo sviluppare determinate funzionalità, gli intervalli sono decisi di comune accordo con il resto del gruppo in base alla difficoltà e alla durata dell’attività da svolgere.
* Diamo molta importanza alla collaborazione con gli utenti, infatti dopo aver messo appunto le funzionalità base dell’applicazione mettiamo subito a disposizione il gioco in modo che gli utenti della community possano darci consigli su come migliorare le funzionalità. Gli utenti non collaborano con noi dal punto di vista organizzativo, ma ci aiutano nell’individuare alcune funzioni aggiuntive che vorrebbero fossero implementate.
* Il team è propenso al cambiamento e si impegna a dedicargli tempo in maniera adeguata. Quando gli utenti avanzano delle richieste, cerchiamo di fare in modo di accontentare tutti stando in ogni modo attenti a quello che si è sviluppato fino a quel momento.
* Nel caso in cui il nostro prodotto venga richiesto da possibili investitori saremo lieti di prendere in considerazioni le proposte. Se l’investitore vorrà poi avere delle funzionalità aggiuntive saremo lieti di prenderle in considerazione. Prima di un possibile finanziamento faremo provare l’applicazione anche al finanziatore.

Come approccio di progettazione software per lo sviluppo del sistema ci siamo attenuti alla model-driven architecture (MDA), o architettura guidata dal modello.

**Configuration menagement**

Tutto il lavoro svolto che si tratti di documentazione o di codice viene salvato in un repository su Github in condivisione con tutti i membri del team.

Struttura del progetto

All’interno del repository sono presenti 2 cartelle:

* La prima cartella chiamata code contiene all’interno tutto il codice del progetto. Le sottocartelle sono 3 più il file pom. Nella prima sottocartella abbiamo il codice appartenente al client, la seconda è in comune con client e server e la terza è appunto il server. La cartella common è stata importata come libreria in server e client.
* La seconda chiamata docs contiene tutta la documentazione a riguardo. Esso conterrà tutta la documentazione effettuata e le linee guida per lo sviluppo del gioco, nonché la spiegazione del gioco.

Oltre alle due cartelle, nel repository vi è anche il file README contenente le informazioni principali del nostro codice.

Issue

Nei vari meeting settimanali vengono creati i macro task che dovranno essere sviluppati nel corso della settimana. Le varie attività sono create come issue su GitHub, aggiungendo una breve descrizione del lavoro e la persona di riferimento che si occuperà di portarla a termine. Durante la settimana ogni membro del team è libero di creare ulteriori issue se lo ritiene necessario.

Le issue possono trovarsi in vari stati a seconda del loro avanzamento. Per tenere traccia dello stato si è utilizzata una Board su Monday suddividendo lo stato in: Not Started, Working on it, To Test, Testing, To Fix e Done:

* *Not Started*: contiene i task che sono stati creati e che devono essere sviluppati.
* Working on it: viene utilizzato per segnalare che si sta lavorando su una determinata funzionalità. Al termine del lavoro, il codice viene salvato su Github e la funzionalità viene marcata come to Test.
* *Testing*: Una volta effettuato il push su Github, il codice deve essere testato. Il nostro tester si occuperà di verificare che il codice sviluppato soddisfi i requisiti descritti all’interno del task. Se tutto funziona correttamente allora passa allo stato successivo (Done), altrimenti lo stato torna in Working on it e viene segnalato a chi ha scritto il codice il problema.

Branch

Il branch principale è il main. Tutta la documentazione si trova in versione stabile, mentre il branch dev che contiene il codice in fase di sviluppo. Esiste anche un terzo branch chiamato test all’interno del quale vengono effettuati i test con JUnit.

Il branch dev viene aggiornato con le versioni work in progress o snapshot e viene utilizzato come backup e cronologia delle versioni del codice. Una volta raggiunta una versione stabile viene fatto il merge sul branch main.

**People management**

L’azienda ha deciso di basarsi su una struttura organizzativa non gerarchica. Utilizzando questa, ha una grande capacità di adattamento e non ha una vera e propria divisione prefissata del lavoro; è infatti il team a decidere come dividere e assegnare i vari task. Inoltre, in caso di difficoltà durante lo sviluppo, due membri del team possono affiancarsi per risolvere il problema in coppia. Il vantaggio di utilizzare l'adhocrazia è che abbiamo un'organizzazione flessibile, adattabile e informale.

Il team è organizzato secondo il modello SWAT (skilled worker with advanced tools). Abbiamo quindi una squadra composta da tre persone che sviluppano utilizzando canali di comunicazione informali (issue su github e discord). Le riunioni tra i membri del team sono tutte informali ed abbiamo deciso di fissare con una frequenza settimanale una riunione per condividere idee e passi su come procedere.

A nostro parere, il vantaggio di rifarsi ad un’architettura non gerarchica e ad un modello SWAT è che ogni componente del team sviluppa software e implementa funzionalità di componenti che lo affascinano maggiormente. Vogliamo quindi non solo arrivare all'obiettivo finale di sviluppare software e svilupparlo bene, ma di costruirlo anche con piacere.

Anche se non abbiamo una vera e propria distinzione del lavoro, possiamo riassumere in maniera più generale i settori di sviluppo e le persone che ne dedicano più tempo:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Project manager | Progettista software | Back-end developer | Front-end developer | Testing |
| Lorenzo |  | X | X | X | X |
| Saif |  |  |  | X | X |
| Marianna | X |  |  |  | X |

**Software Quality**

Il team si è prefissato l’obiettivo di sviluppare un’applicazione che rispetti i parametri e gli attributi di qualità definiti da McCall-Richards-Walters nel documento da loro redatto nel 1977, di seguito elencati e suddivisi in categorie come indicato dagli autori.

Parametri riguardanti l’operatività del software

* *Correttezza* – L’applicazione da noi realizzato soddisfa i requisiti e le specifiche individuate all’inizio del lavoro. Le funzionalità da noi non sviluppate ma solo progettate saranno portate avanti dal team di sviluppo del futuro investitore che assumerà dopo che avremo consegnato il prototipo.
* *Affidabilità* - L’applicazione è affidabile in quanto revisionato a livello di team e sottoposto a una lunga fase di testing prima del rilascio.
* *Efficienza* - Le risorse utilizzate dal prodotto sono limitate in quanto si tratta di un’applicazione web e molte delle operazioni sono effettuate lato server. È quindi comunque necessaria una connessione internet e un browser per accedere ai servizi del software da noi sviluppato.
* *Integrità* - Il prodotto che sviluppiamo è sicuro:
  + *I messaggi in chat sono criptati e nessun dato degli utenti è salvato sul server*.
  + *Non sfruttiamo servizi esterni*
* *Usabilità* - Il prodotto è semplice da utilizzare, infatti non sono richieste particolari abilità per poterne usufruire. L’uso dei servizi è facilitato su tutti i tipi di piattaforma grazie all’utilizzo delle librerie che permette di avere componenti grafici semplici e facili da utilizzare.

I requisiti “base” includono:

* + Avere una connessione internet
  + Sapere come si gioca a Lupus in tabula (o dal manuale ufficiale o quello fornito nelle versioni future)
  + Avere un pc funzionante (Linux, Mac o Windows)

Parametri riguardanti la revisione del software

* *Manutenibilità* - La fase di individuazione degli errori è semplificata dalla nostra scelta di separare i lati server e client. Il processo è inoltre facilitato poiché si ha la parte grafica separata dalla parte di gestione dei dati: è semplice fare manutenzione avendo persone specializzate in ambiti differenti.
* *Testabilità* - Tutte le feature incluse nel prototipo sviluppato sono testabili (tramite test manuali e test automatici) prima del rilascio dell’applicazione al pubblico. Delle funzionalità che saranno invece da noi solo progettate si occuperanno in futuro i dipendenti della società dell’investitore: saranno loro a svilupparle e ad effettuare la fase di testing.
* *Flessibilità* - Come per la manutenibilità, anche il processo di modifica, adattamento e perfezionamento dell’applicazione è facilitato dalla scelta di separare i lati client e server.

Parametri riguardanti la transizione verso un nuovo ambiente

* *Portabilità* - È possibile utilizzare i servizi offerti dal nostro prodotto da qualsiasi dispositivo (Linux, Mac o Windows) dotato di una connessione internet. Il server opera su qualsiasi dispositivo senza limiti, mentre il client ha bisogno di scheda video (anche integrata). Sfruttiamo delle librerie native per i vari sistemi operativi che sono già integrati e disponibili nel programma.
* *Riusabilità* – sono state utilizzate delle librerie, da noi sviluppate contenenti codice per il software ma che possono essere riutilizzate anche in altri ambiti o contesti.
* *Interoperabilità* - Nel futuro la nostra applicazione potrà essere integrata con altri servizi in quanto sviluppiamo in modo separato client e server.

**Requirements Engineering**

Uno dei passi più importanti prima della scrittura del codice è quello di analizzare i requisiti.

Nel nostro caso non ci siamo basati su dei requisiti chiesti da clienti, in quanto il nostro lavoro al momento non è stato acquistato e non è stato richiesto da nessuno. Per tale motivo siamo stati noi a decidere i requisiti che a nostro parere sono importanti per la nostra applicazione. Abbiamo scelto dei requisiti che potessero essere funzionali per il nostro gioco e che potessero inizialmente essere la base per diverse altre implementazioni.

Per la seconda parte del lavoro invece ci siamo basati sui pareri degli utenti della community che hanno giocato al gioco e che hanno voluto darci alcune recensioni e alcune idee per le migliorie. Alcuni di questi utenti hanno preso parte al nostro meeting settimanale per aiutarci a comprendere meglio i requisiti e le funzionalità aggiuntive da implementare. Abbiamo dunque svolto in questo meeting con alcuni utenti la fase di elicitation, capire dunque le funzionalità volute dagli utenti, la fase di negotiation, negoziare dunque le funzionalità volute con quelle a nostro parere realizzabili e infine la fase di requirements specification, cioè specificare i requisiti finali approvati sia dal team che dagli utenti della community.

Il documento finale contenente i requisiti può subire variazioni a fronte di modifiche richieste da ulteriori utenti oppure da futuri investitori. Queste richieste verranno presentate durante la fase di validazione in cui il team mostrerà agli utenti od a futuri investitori parte del prodotto finale tramite prototipi. Qualora il team avesse fatto errate supposizioni, può concordare con utenti della community o prossimi finanziatori una soluzione.

Inoltre, il project manager ha applicato un’ulteriore suddivisione dei requisiti secondo il modello MoSCoW.

Nel documento dei requisiti concordato con il cliente ci siamo accordati su quelli qui di seguito riportati.

Funzionali

● Must Have:

* Sviluppo del gioco nella versione base funzionante
* Inserimento della lingua italiana oltre alla lingua inglese
* Networking
* Interfaccia grafica
* Inserimento della chat testuale
* Grafica del giorno e della notte
* Stabilità del software

●  Should have:

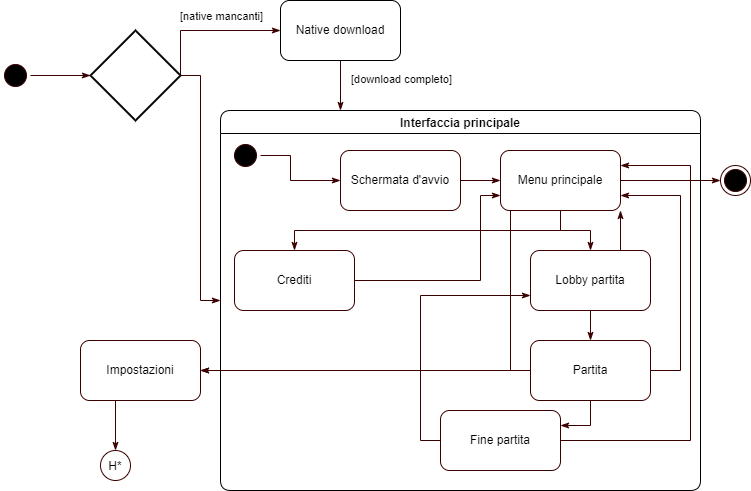
* Altre lingue oltre all’italiano e all’inglese
* Guida del gioco per una comprensione migliore
* Crittografia chat testuale
* Autenticazione utenti sicura
* Consentire al game master di scegliere i ruoli da aggiungere
* Timeout delle lobby
* Possibilità di scegliere la lingua del gioco
* Effetti sonori

●  Won’t have:

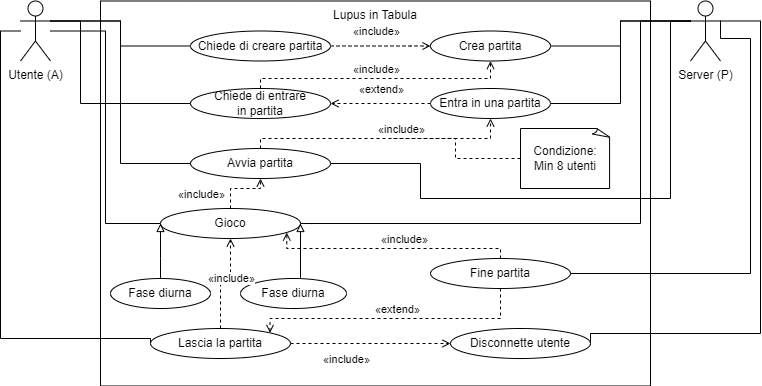
* Chat vocale
* Possibilità di modificare le impostazioni e la grafica del gioco
* Doppia autenticazione tramite codice OTP inviato al numero di telefono

**Modelling**

* State machine diagram



Abbiamo rappresentato con la macchina a stati il funzionamento della interfaccia del client. Dalla schermata di avvio a seguito del native download (operazione una tantum in cui vengono scaricate le librerie essenziali per il corretto funzionamento del programma) si arriva nella schermata di avvio e a seguire del caricamento si giunge nel menù principale, dal quale si può andare a controllare i crediti, le impostazioni, la lobby della partita o si può terminare. Arrivati alla schermata con i crediti si può tornare alla schermata del menù principale. Dalla lobby della partita si può o tornare indietro al menù principale o andare alla partita. Anche dalla partita si può andare al menù principale, alle impostazioni o alla fine della partita. Al termine della partita di può tornare alla lobby della partita oppure al menù principale.

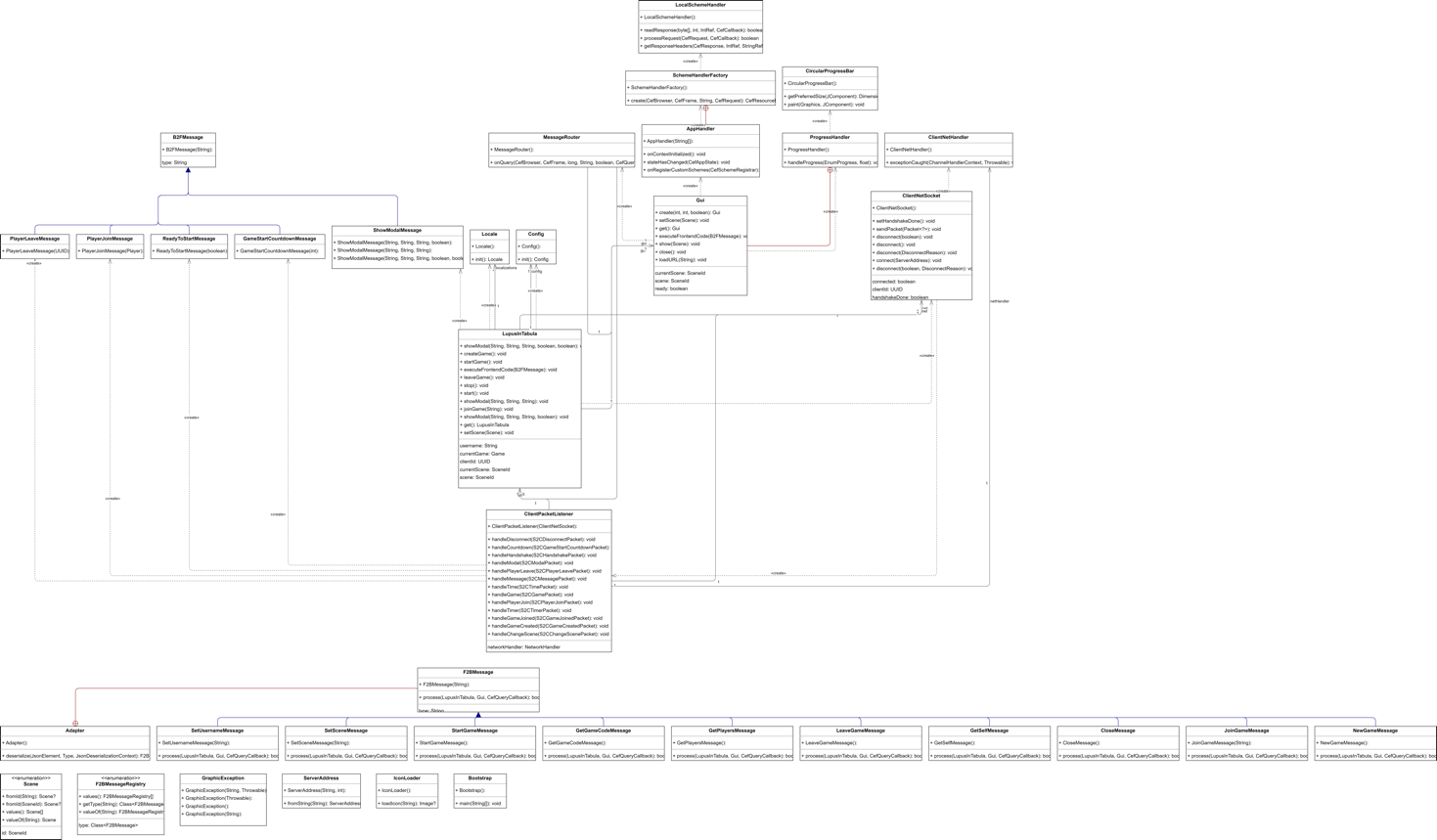
* Use case diagram

Per modellare i casi d’uso, ovvero le funzionalità degli attori nel sistema, abbiamo modellato uno use-case diagram.

L’attore principale è l’utente che effettua diverse azioni. Come prima cosa deve chiedere al server di creare la partita o chiedere di unirsi ad una già esistente. Una volta avviata la partita, cosa che richiede un minimo di 8 utenti, può iniziare a giocare. Nel gioco ci sono due fasi: giorno e notte (caratterizzate anche dal cambiamento della grafica). Al termine della fase di gioco la partita può terminare. Per lasciare la partita bisogna aver terminato il gioco oppure si può lasciare la partita anche durante lo svolgimento. Una volta lasciata la partita l’utente viene disconnesso in automatico.

Il secondo attore è il server che deve essere attivo per l’intero svolgimento delle azioni dell’utente. Esso risponde alle richieste dell’utente.

* Class diagram



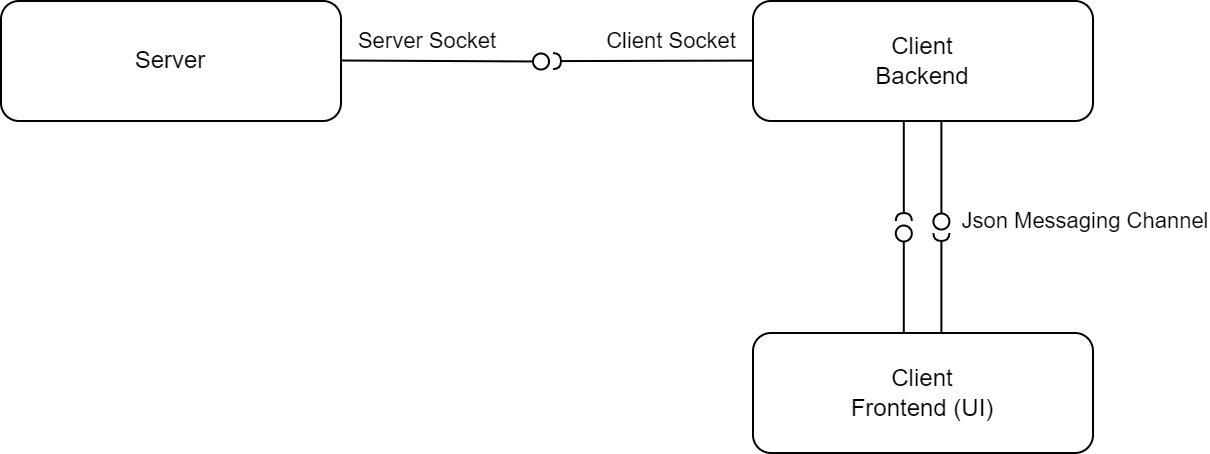
* Sequence diagram
* Activity diagram

**Software Architecture**

L’architettura del nostro sistema è un’architettura client server basata sul pattern MVC (model - view - controller). Abbiamo quindi il client che si limita a fare da view in quanto mostra lo stato del gioco all’utente e da IO in quanti gestisci gli input.

Il client mostra la view all’utente o una rappresentazione del modello sotto una veste grafica e accetta le interazioni dall’utente che vengono poi inviate al server per essere elaborate. Il server fa da controller e model, in quanto si occupa di astrarre e manipolare i dati. Ovvero si occupa di gestire la partita, coordinare il funzionamento, decretare i vincitori.

La vista con connettori e componenti del nostro sistema è riassunta nel seguente schema:



Abbiamo creato un’architettura modulare con bassa dipendenza tra i vari moduli. Le interazioni tra un modulo e l’altro sono gestite da delle interfacce adeguate. Il codice è strutturato in modo da migliorare la leggibilità e soprattutto la mantenibilità del codice. É bene sottolineare che al momento la struttura del lato server è monolitica, infatti gestisce tutti i servizi di interazione con gli utenti su un unico processo. In futuro è in programma la divisione di questi compiti in un’architettura basata su micro servizi.

**Software Design**

Per lo sviluppo dell’applicazione il team si è basato sul pattern Model-view-controller (MVC), allo scopo di suddividere il codice in parti che abbiano funzionalità distinte tra loro.

I dati, la loro gestione e le interazioni con il database (presenti in futuro) sono gestiti dai livelli Model-Controller a lato back-end, che si occupa di gestire le query e la connessione alla database.

Il livello che corrisponde alla View è l’output che presentiamo in front-end, ovvero le pagine vere e proprie che appaiono all’utente.

Il team si è inoltre preposto di scrivere codice basandosi sui concetti di astrazione e modularità, con l’obiettivo finale di avere un programma che abbia:

● basso accoppiamento, ovvero avere moduli altamente indipendenti tra di loro; ad esempio:

○  common coupling: non ci sono variabili globali condivise tra i moduli;

○  control coupling: nessun modulo va ad utilizzare codice che appartiene a un altro modulo;

○  external coupling: il modulo relativo all’interfaccia grafica sarà presente nel codice del client, la parte di gestione del modello sul server e il modulo comune sarà sfruttato come libreria su entrambi i lati. Per il resto non ci sono file condivisi.

● alta coesione, ovvero avere una alta correlazione delle funzionalità presenti dentro un singolo modulo; ad esempio:

○ procedural cohesion: per ogni procedimento ci sono delle azioni che vengono sempre eseguite una dopo l’altra. Ad esempio, quando un’utente interagisce con l’interfaccia grafica il controller dell’interfaccia si occuperà di gestire se è un azione che può gestire autonomamente (come il cambio di scena) oppure se deve fare richiesta al server.

**Software Testing e manutenibilità**

Durante lo sviluppo il corretto funzionamento del codice è stato testato costantemente con prove pratiche di esecuzione. In caso di superamento dei test siamo andati nella scrittura del codice, in caso contrario ci siamo focalizzati nella correzione di eventuali bug, effettuando nel contempo eventuali miglioramenti e refactoring. Una volta terminato gran parte del lavoro abbiamo implementato dei test mediante JUnit per verificare il corretto funzionamento del codice anche in presenza di casi limite.