Impiccato

1 Introduzione 3

1.1 Informazioni sul progetto 3

1.2 Abstract 3

1.3 Scopo 3

2 Analisi 5

2.1 Analisi del dominio 5

2.2 Analisi e specifica dei requisiti 5

2.3 Use case 9

2.4 Pianificazione 10

2.5 Analisi dei mezzi 11

2.5.1 Software 11

2.5.2 Hardware 11

3 Progettazione 13

3.1 Design dell’architettura del sistema 13

3.2 Design dei dati e database 13

3.3 Design delle interfacce 13

3.3.1 Schermata iniziale 13

3.3.2 Schermata delle Impostazioni 14

3.3.3 Schermata di creazione della partita 14

3.3.4 Schermata di attesa 14

3.3.5 Schermata di gioco 15

3.3.6 Schermata finale 15

3.4 Design procedurale 17

4 Implementazione 19

4.1 Package client 19

4.1.1 Client 19

4.1.2 Package Server 22

4.2 Package Game 24

4.2.1 GameHoster 26

4.2.2 Game 26

4.2.3 Player 26

4.2.4 UsernameChecker 26

4.2.5 Word 26

4.3 Package graphic 27

4.3.1 MainPanel 27

4.3.2 CreateGamePanel 28

4.3.3 WaitingPanel 29

4.3.4 GamePanel 30

4.3.5 FinalPanel 32

4.3.6 ErrorPanel 33

4.3.7 SettingsPanel 33

4.3.8 Addable 34

4.4 Package protocol 34

4.4.1 ProtocolCodes 34

5 Test 35

5.1 Protocollo di test 35

5.2 Risultati test 38

5.3 Mancanze/limitazioni conosciute 40

6 Consuntivo 41

7 Conclusioni 43

7.1 Sviluppi futuri 43

7.2 Considerazioni personali 43

8 Bibliografia 43

8.1 Bibliografia per articoli di riviste: 43

8.2 Bibliografia per libri 43

8.3 Sitografia 43

9 Allegati 43

# Introduzione

## Informazioni sul progetto

Allievo: Gioele Zanetti

Docente: Guido Montalbetti

Classe: SAMT I3BC

Materia: laboratorio progetti

Periodo: 16.09.2021 – 23.12.2021

## Abstract

*Fino ad ora per giocare all’impiccato bisognava giocare su carta con matite o penne. Lo scopo del mio progetto è quello di avere un’applicazione dedicata in modo da poter giocare senza l’ausilio di carta e penna. Tramite lo sviluppo ad oggetti e alla programmazione concorrente sono riuscito a creare un prodotto funzionante che copre tutti gli aspetti del gioco dell’impiccato.*

*In questo documento vado a spiegare i problemi che ho avuto durante l’implementazione nonché la progettazione di questa applicazione.*

## Scopo

Lo scopo del progetto è quello di avere un’applicazione funzionante che permetta di giocare al gioco dell’impiccato nella stessa rete. Si potranno creare partite, unirsi a queste e giocare insieme ad altre persone.

# Analisi

## Analisi del dominio

In questo progetto mi è stato richiesto di sviluppare il gioco dell’impiccato online. Fino ad adesso bisognava giocare su carta, usando penne o matite.

Con lo sviluppo di questo progetto, sarà possibile giocare all’interno della stessa rete all’impiccato in un ambiente grafico fatto apposta.

Attualmente non esistono altri applicativi che svolgono lo stesso compito.

Questo prodotto verrà creato allo scopo di ammazzare il tempo nei momenti di noia. Non è quindi indirizzato verso un pubblico specifico.

Per giocare, basta sapere le regole del gioco e sapere come trovare l’ip della macchina.

## Analisi e specifica dei requisiti

|  |  |
| --- | --- |
| **ID: REQ-001** | |
| **Nome** | Creazione della partita |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | È presente un tasto per creare una partita e uno per unirsi ad una partita |
| **Sotto requisiti** | |
| **001** | Connessione al server |

|  |  |
| --- | --- |
| **ID: REQ-002** | |
| **Nome** | Controllo nome utente |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Non deve contenere né volgarità né essere vuoto o contenere solo spazi. Deve essere inoltre diverso dai nomi dei giocatori già presenti nella partita |

|  |  |
| --- | --- |
| **ID: REQ-003** | |
| **Nome** | Unirsi ad una partita |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | - |
| **Sotto requisiti** | |
| **001** | Nome utente adeguato |
| **002** | Hosting partite funzionante |
| **003** | Generazione token partita |

|  |  |
| --- | --- |
| **ID: REQ-004** | |
| **Nome** | Chat broadcast |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Chat in cui tutti posso vedere le lettere che gli altri giocatori scrivono |
| **Sotto requisiti** | |
| **001** | Hosting partite funzionante |

|  |  |
| --- | --- |
| **ID: REQ-005** | |
| **Nome** | Lista dei giocatori |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Ritornare una lista dei giocatori, il nome di un giocatore deve essere in verde se ha indovinato la parola corrente |
| **Sotto requisiti** | |
| **001** | Hosting partite funzionante |

|  |  |
| --- | --- |
| **ID: REQ-006** | |
| **Nome** | Parole a disposizione |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessitano almeno 500 parole della lingua italiana |

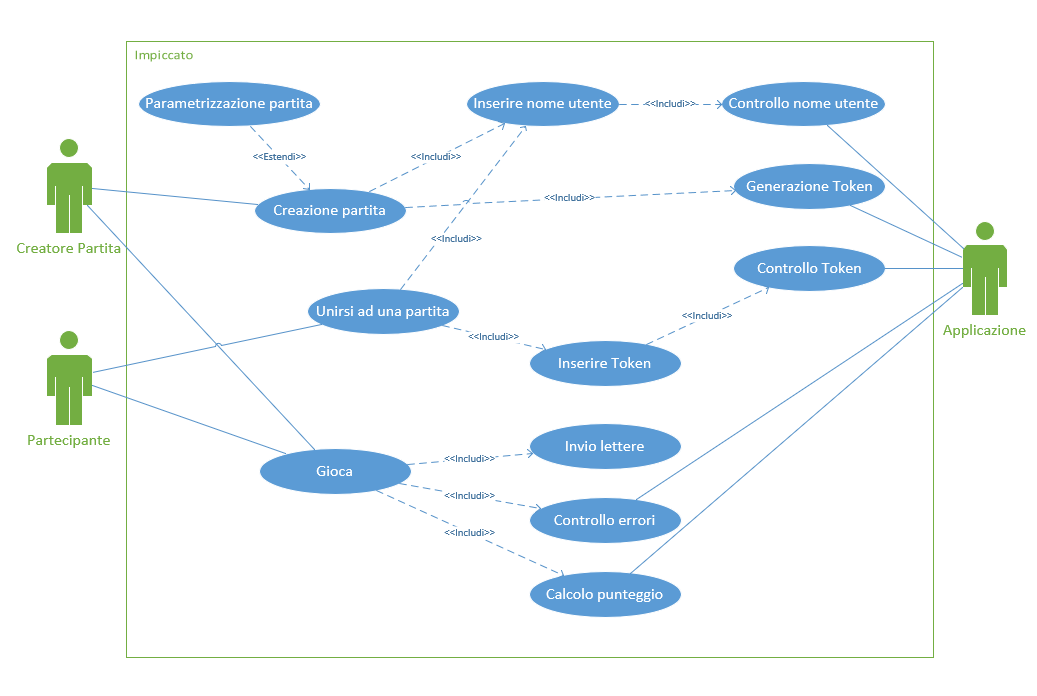
|  |  |
| --- | --- |
| **ID: REQ-007** | |
| **Nome** | Gestione delle lettere inviate al server |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Vengono gestite lettere fino al carattere U+00FF |
| **Sotto requisiti** | |
| **001** | Invio delle lettere al server funzionante |
| **ID: REQ-008** | |
| **Nome** | Gestione errori |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Un giocatore può sbagliare al massimo 10 volte. Una volta raggiuto questo limite il giocatore dovrà aspettare il turno successivo per continuare a giocare. |
| **Sotto requisiti** | |
| **001** | Hosting partite funzionante |
| **002** | Invio lettere funzionante |
| **003** | Dovrà essere possibile la ricerca di allievi |

|  |  |
| --- | --- |
| **ID: REQ-009** | |
| **Nome** | Schermata finale |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Fa vedere i giocatori con il relativo punteggio |
| **Sotto requisiti** | |
| **001** | Ultimo round terminato |
| **002** | Calcolo dei punteggi |

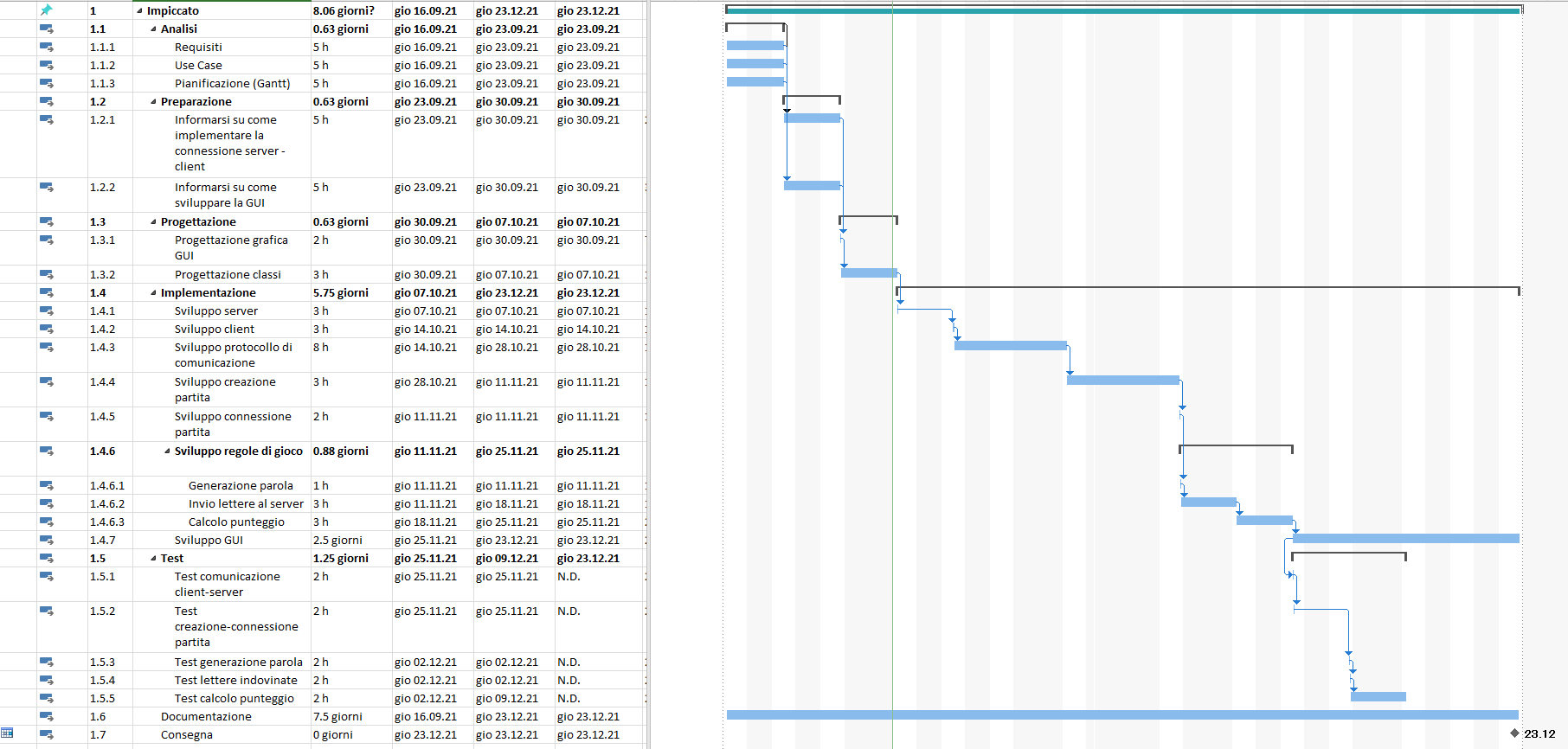
|  |  |
| --- | --- |
| **ID: REQ-010** | |
| **Nome** | Parametrizzazione partita |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si può scegliere il numero e la durata di ogni round (durata minimo 30 secondi) |
| **Sotto requisiti** | |
| **001** | Creazione di una partita |

|  |  |
| --- | --- |
| **ID: REQ-011** | |
| **Nome** | Calcolo punteggio |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Il punteggio viene calcolato in base al tempo rimasto alla fine del round |
| **Sotto requisiti** | |
| **001** | Fine di un round |

## Use case



## Pianificazione



## Analisi dei mezzi

### Software

Microsoft project, Visio, Moodle, Microsoft Word, NetBeans IDE, JAVA VM, Github,

### Hardware

PC – specifiche:

* Intel Xeon E3-1240
* 16 GB RAM
* 256 GB SSD
* Nvidia Quadro

# Progettazione

## Design dell’architettura del sistema

Ecco una breve descrizione degli oggetti che compongono il mio progetto. Un approfondimento si trova nella sezione di implementazione

App: è l’applicazione da terminale

MainFrame: è l’applicazione grafica

Playable: rende una classe giocabile

Client: rappresenta il client

ServerHandler: riceve e interpreta i messaggi del server

GameHoster: oggetto che ospita tutte le partite

Game: rappresenta la partita in sé

Player: rappresenta il player

UsernameChecker: controlla che il nome utente rispetti le specifiche

Word: si occupa di estrarre parole a caso

ProtocolCodes: si occupa di costruire i pacchetti

ClientHandler: classe per gestire i messaggi del client

Server: server che ospita tutti i client

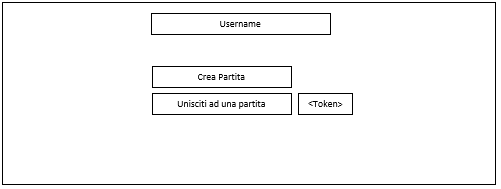
Pannelli vari: sono delle schermate di gioco grafiche

## Design dei dati e database

Questo progetto non utilizza un database.

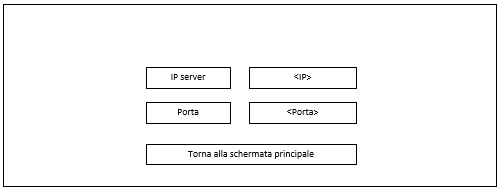
## Design delle interfacce

### Schermata iniziale



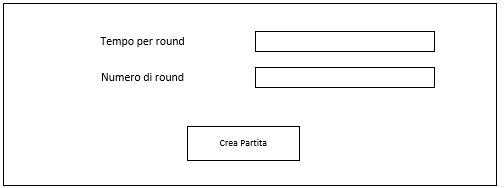
In questa schermata bisogna inserire l’username. È possibile scegliere se creare una partita o se unirsi ad una già presente (bisogna inserire il token della partita). È anche possibile accedere alla schermata delle impostazioni.

### Schermata delle Impostazioni



In questa schermata si può modificare l’IP del server e la porta in cui è in ascolto. Serve a impostare la connessione tra il client e il server. Se i dati non dovessero essere corretti, l’applicazione mostra di nuovo una schermata di errore.

### Schermata di creazione della partita



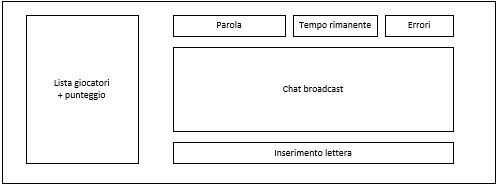
È possibile parametrizzare alcuni aspetti della partita.

### Schermata di attesa



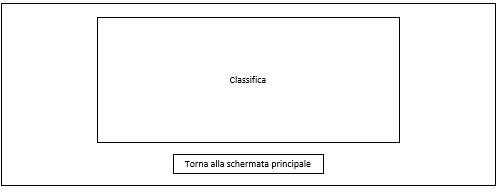
In questa schermata viene visualizzato il token della partita e i giocatori presenti. Il tasto per iniziare funziona solo se premuto dall’admin della partita.

### Schermata di gioco



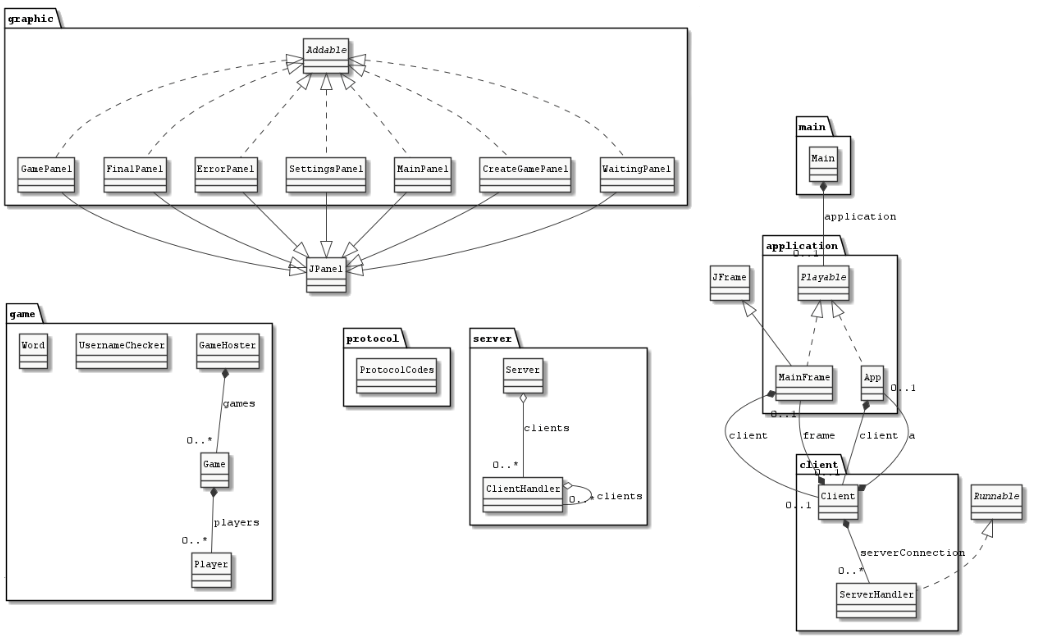
Una volta cominciata la partita, si può vedere una lista dei giocatori con il loro punteggio, la parola (ovviamente con solo le lettere indovinate inserite), il tempo prima della fine del round, la chat in cui si vedono le lettere inviate da tutti e una barra per inserire una lettera.

### Schermata finale



Nella schermata finale si vede la classifica dei giocatori della partita. È presente un tasto per tornare alla schermata iniziale.

## Design procedurale



# Implementazione

## Package client

La comunicazione tra client e server è la base del progetto. Ho pensato a 4 classi: una che simula il client, una che simula il server e le altre due gestiscono le connessioni dei client per il server e rispettivamente dal server per il client.

### Client

Per quanto riguarda il client, la classe Client è quella che lo rappresenta, mentre la classe ServerHandler serve per gestire i messaggi inviati dal server.

Gli attributi della classe Client sono: la porta del server con il relativo ip, il socket con il server, l’applicazione che farà da tramite con l’utente, l’oggetto per mandare messaggi al server e i dati utili al client: username, token della partita, parola mascherata, se la partita è cominciata e gli errori fatti.

I metodi servono a gestire le dinamiche di gioco: il metodo elaborateRequest è il metodo generale che serve a coordinare il tutto. Questo delega poi ai metodi startGame, […], createGame per gestire la richiesta dell’applicazione. ElaborateResponse invece si occupa di gestire le risposte dal server e delega ai metodi endTurn, […], gameCreatedSuccessfully il compito di attuare le operazioni necessarie per far funzionare il client.

Nella classe ServerHandler troviamo l’oggetto per ricevere dati dal server, il client a cui fa riferimento e il socket del server. Come metodi troviamo il costruttore e il metodo run, visto che implementa la classe runnable.

Questo è il codice che permette di gestire le richieste che l’utente genera quando preme bottoni o scrive comandi.

Immagine che contiene testo

Descrizione generata automaticamente

Attraverso un messaggio, il metodo sa se deve chiedere qualcosa all’utente. Ad esempio quando il messaggio è “create game”, se l’applicazione dovesse essere grafica prenderebbe i parametri passati dai textfield, mentre se invece si gioca da terminale richiede un inserimento dei dati dalla tastiera.

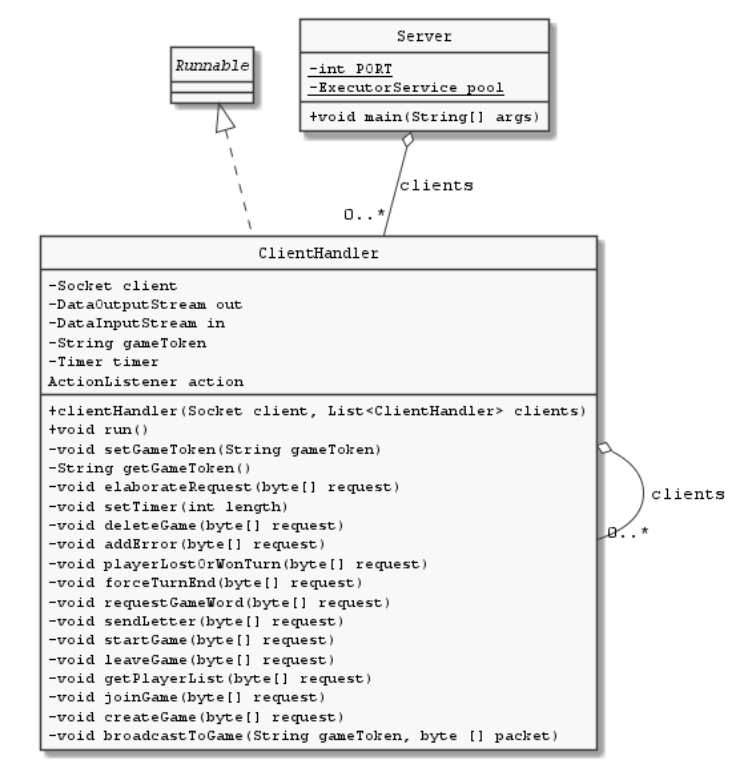
Questo è un estratto di come gestisce le risposte del server



[…]

Si può vedere che una volta riconosciuto il primo byte, il metodo sa a chi delegare il compito di interpretare la risposta.

### Package Server



Per quello che riguarda il server invece, abbiamo la porta in cui ascolta e il pool in cui mette tutti i client che si connettono. CientHandler come attributi ha il client che si connette, l’oggetto per ricevere dati e per mandarli al client. Action contiene l’azione da eseguire una volta che il timer finisce. Il metodo run non fa altro che aspettare un messaggio dal client. Gli altri metodi servono per gestire le richieste dal client: elaborateRequest è il metodo generale mentre deleteGame, […], createGame si occupano di far eseguire al server le richieste del client.

Visto che la comunicazione tra socket è uno stream continuo di dati l’app non riusciva a riconoscere quando veniva inviato qualcosa dal client o dal server. Per risolvere questo problema, mando prima del pacchetto un intero che mi specifica quanti byte devo leggere dopo: in questo modo l’app riesce a caprie quali byte sono un messaggio e quali no.

Immagine che contiene testo

Descrizione generata automaticamente

Durante la parte di implementazione della temporizzazione dei turni, ho riscontrato un problema con il timer che stavo usando. Dopo essermi documentato sufficientemente, ho trovato questa soluzione:

Immagine che contiene testo

Descrizione generata automaticamente

Questo fa in modo che il timer venga fatto partire e dopo aver aspettato ‘length’ tempo, viene eseguita l’azione contenuta in action: viene inviato un pacchetto che forza la fine del turno.

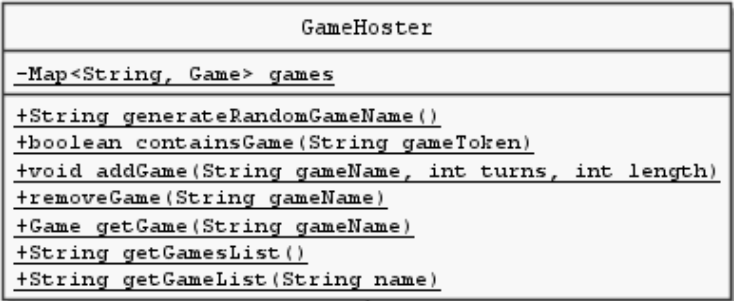
Un’altra parte importante del codice è come ho gestito le disconnessioni dei client, cosa che può capitare durante una partita. Ho deciso che se l’admin esce dalla partita tutti vengono buttati fuori e la partita finisce, mentre se l’admin rimane e qualcun altro esce, la partita continua con un giocatore in meno.

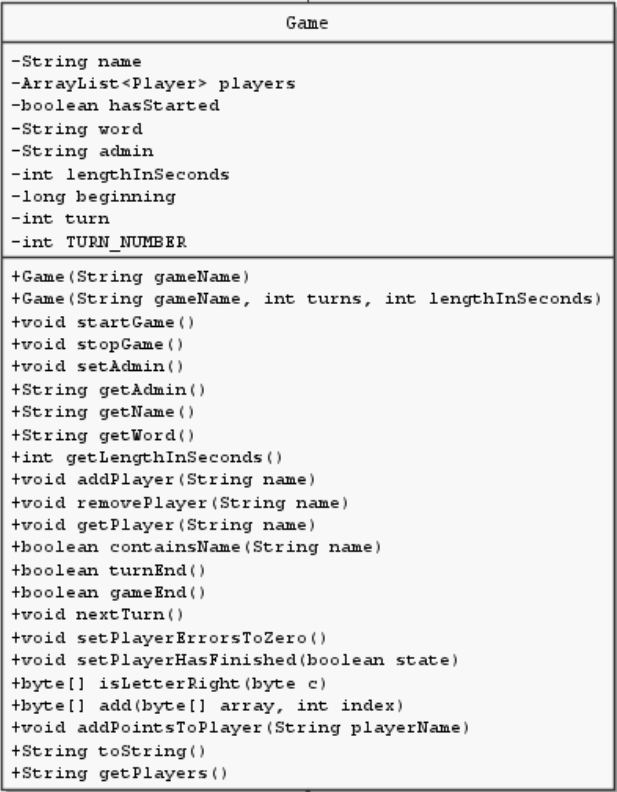
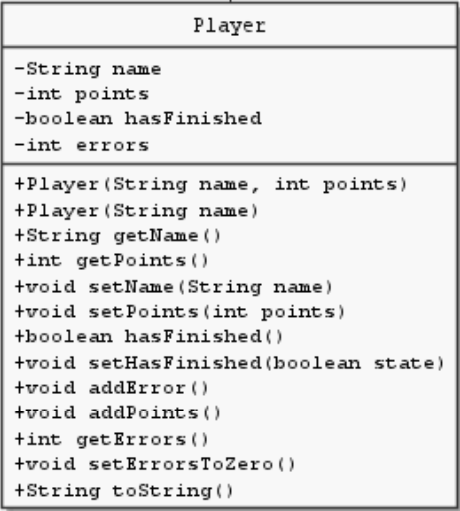
Immagine che contiene testo

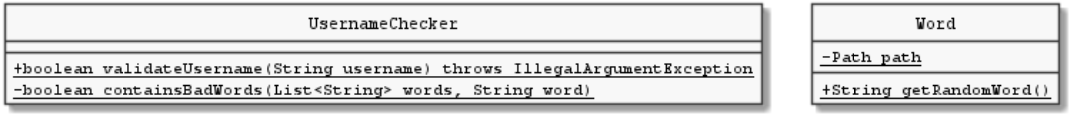
Descrizione generata automaticamente

## Package Game

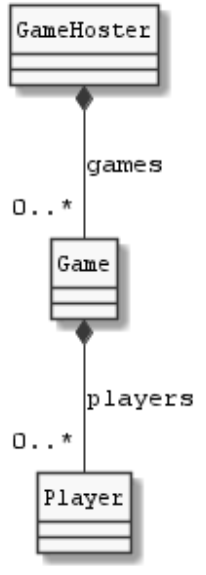
In questo package troviamo tutte le classi che servono per creare partite e poter giocare. Le classi singole sono le seguenti:







E sono relazionate in questo modo



### GameHoster

GameHoster rappresenta la classe che contiene tutte le partite e contiene dei metodi utili a creare, eliminare e ritornare le partite. Contiene un metodo per dare un nome casuale di 8 caratteri alle partite e 2 metodi per avere la lista dei partecipanti in formato stringa. Ha anche un metodo per ritornare la classifica ordinata di ogni partita.

### Game

La classe Game rappresenta la nostra partita: ha dunque degli attributi che servono per conoscere lo stato della partita: il nome, la lista dei giocatori, lo stato della partita, la parola corrente, l’admin della partita, la lunghezza in secondi, il momento in cui è cominciata, il turno corrente e il numero di turni totali. I metodi sono auto esplicativi: i costruttori creano una nuova istanza della partita, startGame fa cominciare la partita, stopGame la fa finire, setAdmin inserisce l’unsername dell’admin della partita, getAdmin lo ritorna. Il metodo getName ritorna il nome della partita e getWord ritorna la parola estratta. GetLengthInSdeconds ritorna la lunghezza della partita. AddPlayer, removePlayer e getPlayer servono ad aggiungere, rimuovere e ritornare un giocatore all’interno della partita. Il metodo containsName controlla se un certo nome utente è già presente all’interno della lista dei partecipanti. TurnEnd e gameEnd si occupano di far finire un turno o una partita, mentre nextTurn fa passare i turni. I metodi isLetterRight e add servono per far capire al client se la lettera che è stata mandata è corretta o no. Se la lettera dovesse essere corretta, viene ritornato l’indice di tutte le posizioni della lettera all’intero della parola.

### Player

La classe player rappresenta il giocatore. Tiene in considerazione il nome, i punti, gli errori e se ha finito o meno la partita / turno. Anche qui i metodi sono auto esplicativi: ci sono setter e getter per gli attributi, metodi per resettare completamente gli errori, aggiungere punti, aggiungere errori e per avere una rappresentazione sotto forma di stringa dell’oggetto.

### UsernameChecker

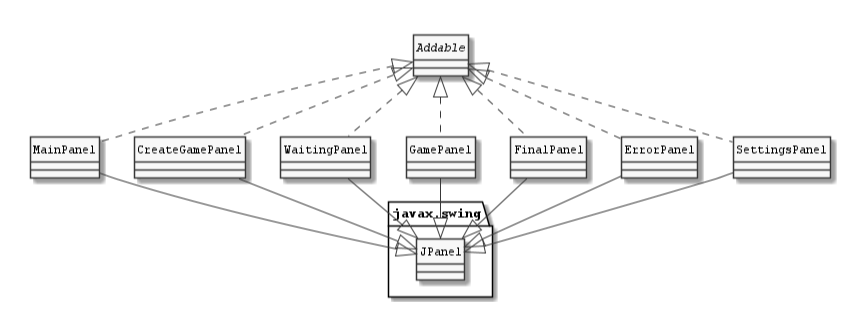
Questa classe offre dei metodi per controllare che il nome utente soddisfa i requisiti contenuti nel QdC (no nome nullo o solo spazi, no parolacce). La lista di parole non ammesse si trova all’interno di un file.

### Word

La classe word non fa altro che estrarre una parola casuale dalla lista di parole disponibili. Questa lista si trova in un file

## Package graphic

In questo package ci sono tutte le classi che servono per la grafica: troviamo quindi le varie schermate di gioco.

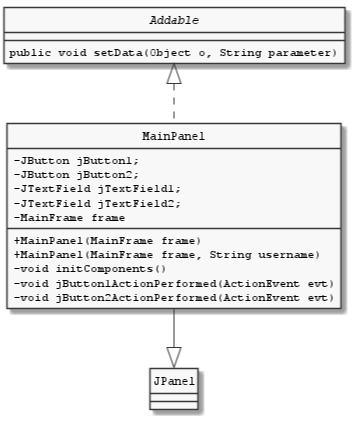


Questo è come le classi della grafica si relazionano.

Come si può vedere sono tutti dei JPanel perché li metterò dentro un JFrame per comodità. Tutti implementano un interfaccia di mia creazione di nome Addable perché in questo modo ho un modo per gestire gli eventi asincroni.

### MainPanel

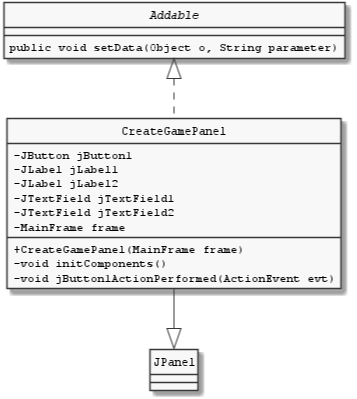
MainPanel è il pannello pricipale: ha dei bottoni che permettono di scegliere se creare una partita o se unirsi ad una già esistente. Inoltre troviamo un text field che mi permette di scegliere l’username che voglio (vincolato dalla classe UsernameChecker). L’attributo frame è il JFrame che contiene tutti i pannelli e sarà presente in tutte le classi di questo package. Permette di accedere a delle funzioni del client. La classe contiene due costruttori, i metodi di default del GUI builder, i bottoni e le azioni compiute da essi.

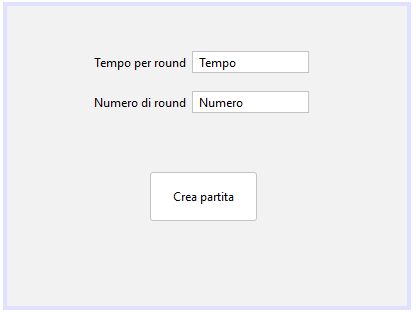




### CreateGamePanel

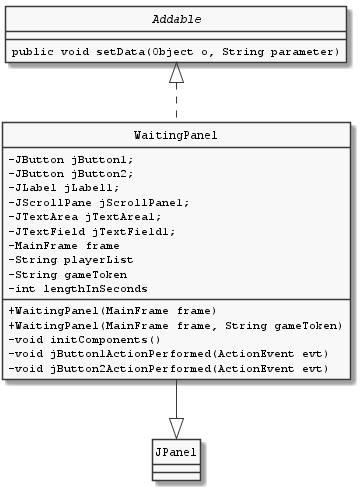
CreateGamePanel è il pannello che viene mostrato quando si clicca su “Crea partita” nella schermata principale. Ci sono dei textbox che permettono di inserire il tempo di ogni round e il numero di round. La durata di ogni round è vincolata in questo modo [30; 200], mentre il numero di round consentiti è da un minimo di 1 ad un massimo di 10 incluso.

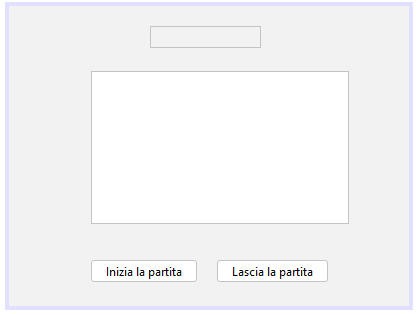




### WaitingPanel

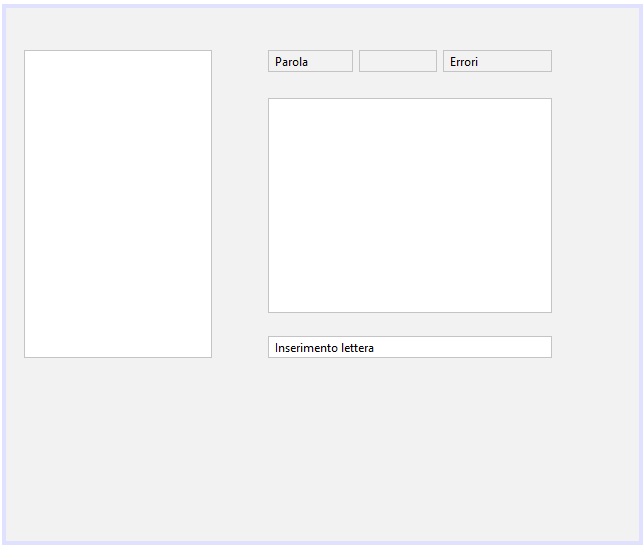
WaitingPanel è il pannello di attesa una volta che la partita è stata creata. Ha al suo interno un bottone per iniziare la partita (tutti lo possono premere, ma solo l’admin può iniziare la partita) e un bottone per lasciare la partita. C’è una textarea che permette di vedere i giocatori presenti nella partita. In alto si trova un textbox che contiene il token della partita appena creata.

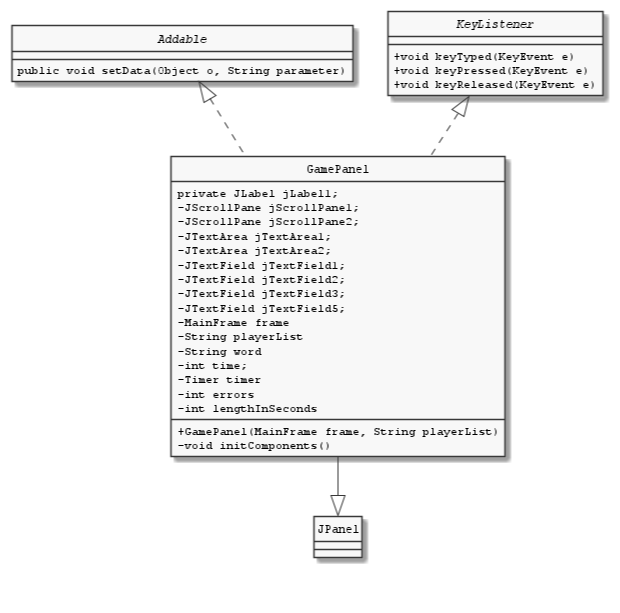




### GamePanel

GamePanel è il pannello che permette di giocare: si vede quindi la lista dei giocatori con il punteggio aggiornato ogni turno, la parola censurata, il tempo rimanente e gli errori fatti. C’è un textbox che permette di inserire le lettere e una textarea che permette di visualizzare i messaggi del server e le lettere scritte da altri giocatori. Questa classe implementa anche KeyListener perché quando viene premuto il tasto invio bisogna controllare se la lettera inserita è giusta. Gli attributi sono la lista dei giocatori, la parola (censurata), il tempo passato dall’inizio del round, il timer con l’azione da compiere e gli errori.





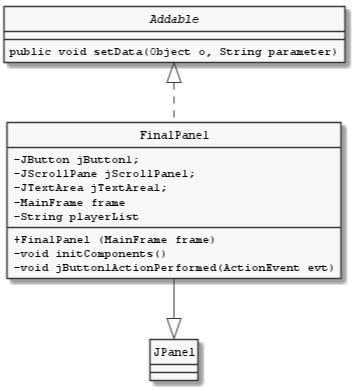
Questo è il modo in cui ho gestito l’input dalla tastiera: quando lascio il tasto invio tutto il testo contenuto in jTextField5 viene inviato al server, che si occuperà di prendere in considerazione solo il primo carattere, e in seguito viene pulito il textbox in modo che l’utente possa inserire un’altra lettera.

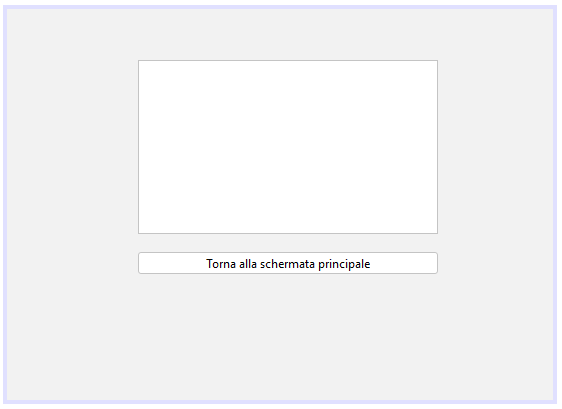
Immagine che contiene testo

Descrizione generata automaticamente

### FinalPanel

FinalPanel è il pannello che viene mostrato una volta finita la partita: mostra quindi la classifica e ha un bottone per tornare alla schermata principale.





### ErrorPanel

ErrorPanel viene mostrato quando il client non riesce a trovare il server socket di default (127.0.0.1 : 3000).

Contiene una breve spiegazione sul perché si è verificato l’errore e un bottone che permette di accedere alla schermata delle impostazioni. Non metto il diagramma delle classi perché non c’è niente di interessante in questo pannello

Immagine che contiene testo

Descrizione generata automaticamente

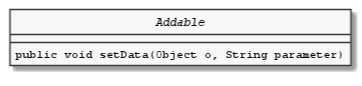
### SettingsPanel

SettingsPanel permette di modificare l’ip e la porta in cui ascolta il server. Possiede quindi due textbox per inserire l’ip e la porta e un bottone per tornare alla schermata principale. Se l’ip o la porta non dovessero essere corretti, verrebbe mostrata un’altra volta la schermata di errore. Anche in questo caso il diagramma delle classi non ha niente di interessante.



### Addable

Addable è l’interfaccia che permette a un JPanel di gestire gli eventi asincroni. Ho fatto in modo che ci fosse un solo metodo che permettesse di gestire i messaggi dal server.



Ecco un esempio di implementazione di questo metodo:

Immagine che contiene testo

Descrizione generata automaticamente

In questo esempio il pannello controlla se viene ricevuta la lista dei giocatori. Una volta ricevuta, imposta gli elementi della lista jList1.

## Package protocol

### ProtocolCodes

La classe ProtocolCodes è la classe che si occupa di costruire tutti i vari pacchetti che vengono mandati tra client e server. Ho deciso che i pacchetti vengono costruiti da array di byte perché mi sembrava una buona scelta in fatto di risorse.

Questa classe ha circa 34 costanti che rappresentano ogni azione e un metodo relativo ad ogni codice. Non metto il diagramma delle classi perché verrebbe fuori troppo lungo.

Oltre ai metodi per costruire i pacchetti, ha anche dei metodi per togliere il codice dall’inizio del pacchetto (quindi avere solo i dati), un metodo per leggere da un certo indice fino ad un altro e dei metodi per togliere la lunghezza che c’è all’inizio di ogni pacchetto (vedi spiegazione [ClientHandler](#Run_ClientHandler))

Questa è la lista dei codici disponibili (ad ogni codice corrisponde un metodo apposito):

* broadcast
* create game
* game created successfully
* join game
* game joined successfully
* game joined unsuccessfully
* username already used
* add player
* get player list
* player list returned
* leave game
* game leaved successfully
* game leaved unsuccessfully
* start game
* game started successfully
* game started unsuccessfully
* send letter
* letter indexes
* request game word
* return game word
* player won turn
* end game
* notify turn won
* player lost turn
* notify turn lost
* add error
* end turn
* delete game
* force turn end
* player joined game
* player left game
* admin left game
* get leaderboard

# Test

## Protocollo di test

Definire in modo accurato tutti i test che devono essere realizzati per garantire l’adempimento delle richieste formulate nei requisiti. I test fungono da garanzia di qualità del prodotto. Ogni test deve essere ripetibile alle stesse condizioni.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001  REQ-001 | **Nome:** | Verifica creazione della partita |
| **Descrizione:** | Verifico che dopo aver schiacciato il tasto per creare una partita la partita venga effettivamente creata sul server | | |
| **Prerequisiti:** | Il protocollo di comunicazione deve comprendere un pacchetto per creare una partita | | |
| **Procedura:** | 1. Avvio l’applicazione 2. Clicco sul tasto per creare una partita 3. Vengo trasportato nella schermata di attesa | | |
| **Risultati attesi:** | Nella console del server viene stampato “creating <Token>” | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-002  REQ-002 | **Nome:** | Verifica controllo nome utente |
| **Descrizione:** | Vengono controllati i requisiti del nome utente: no volgarità, no nomi vuoti o solo spazi, no nomi di giocatori già presenti nella partita | | |
| **Prerequisiti:** | La classe che controlla i nomi deve essere implementata | | |
| **Procedura:** | 1. Avvio l’app 2. Inserisco il nome utente con errori sopra citati | | |
| **Risultati attesi:** | L’applicativo stampa un errore che mi dice che il nome utente non può essere utilizzato per un motivo specifico | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-003  REQ-003 | **Nome:** | Verifica connessione ad una partita |
| **Descrizione:** | Verifico che posso unirmi ad una partita | | |
| **Prerequisiti:** | Il protocollo di comunicazione deve comprendere un pacchetto per unirsi ad una partita | | |
| **Procedura:** | 1. Avvio l’app 2. Inserisco il token di una partita esistente 3. Clicco sul tasto per unirmi | | |
| **Risultati attesi:** | Viene mostrata la schermata di attesa in cui vedo la lista dei partecipanti correnti. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-004  REQ-004 | **Nome:** | Verifica funzionalità chat |
| **Descrizione:** | Verifico che i client ricevano in boradcast le lettere inviate dai client presenti nella stessa partita | | |
| **Prerequisiti:** | Deve essere possibile creare una partita e unirsi alla stessa  La partita deve essere stata inizializzata dall’admin | | |
| **Procedura:** | 1. Creo / mi unisco ad una partita 2. Scrivo una lettera nell’apposita barra 3. Premo invio | | |
| **Risultati attesi:** | Nell’area della chat appaiono le lettere scritte da altri giocatori | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-005  REQ-005 | **Nome:** | Verifica stampa lista dei giocatori |
| **Descrizione:** | Durante la partita e mentre si sta aspettando che la partita inizi, la lista dei giocatori con il loro punteggio deve essere visibile e aggiornata | | |
| **Prerequisiti:** | La partita deve essere in stato di attesa (l’admin non l’ha ancora fatta partire) | | |
| **Procedura:** | Creo / mi unisco ad una partita | | |
| **Risultati attesi:** | La lista di giocatori viene visualizzata senza problemi | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-006  REQ-006 | **Nome:** | Verifica parole a disposizione |
| **Descrizione:** | Verifico che ci siano ameno 500 parole a disposizione | | |
| **Prerequisiti:** | La classe delle parole deve essere implementata | | |
| **Procedura:** | 1. Avvio la classe delle parole 2. Stampo l’array contenente le parole | | |
| **Risultati attesi:** | Vengono stampate le parole a che si possono usare | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-007  REQ-007 | **Nome:** | Verifica invio dati al server |
| **Descrizione:** | Verifica del ricevimento di vari dati (come ad esempio lettere) da client a server | | |
| **Prerequisiti:** | La partita deve essere stata inizializzata dall’admin | | |
| **Procedura:** | 1. Digito nella barra una lettera | | |
| **Risultati attesi:** | Nella console del server vengono stampati i dati inviati dal client | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-008  REQ-008 | **Nome:** | Verifica gestione errori |
| **Descrizione:** | Verifico che quando un utente sbaglia una lettera il conteggio degli errori sale di uno. Se gli errori dovessero arrivare al massimo, il giocatore perde. | | |
| **Prerequisiti:** | 1. La partita deve essere stata inizializzata dall’admin | | |
| **Procedura:** | Digito nella barra una lettera sbagliata | | |
| **Risultati attesi:** | Contatore errore sale di uno per ogni errore e a 10 errori la partita viene terminata | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-009  REQ-009 | **Nome:** | Verifica dati schermata finale |
| **Descrizione:** | Quando la partita è finita mostra una schermata con i nomi dei giocatori e il relativo punteggio. | | |
| **Prerequisiti:** | La partita è finita perché tutti i giocatori hanno indovinato la parola corrente | | |
| **Procedura:** | 1. Indovino tutte le lettere della parola 2. Vengo trasportato nella schermata finale | | |
| **Risultati attesi:** | Viene mostrata una schermata con giocatori e punteggio | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-010  REQ-010 | **Nome:** | Verifica parametri partita |
| **Descrizione:** | Durante la creazione della partita è possibile parametrizzare alcuni aspetti, come la durata del match | | |
| **Prerequisiti:** | Il protocollo di comunicazione deve comprendere un pacchetto per parametrizzare la partita | | |
| **Procedura:** | 1. Avvio l’applicazione 2. Clicco sul tasto per creare una partita 3. Vengo trasportato nella schermata di parametrizzazione 4. Creo la partita 5. Avvio la partita | | |
| **Risultati attesi:** | Quando si crea una partita viene mostrata una schermata di parametrizzazione, e durante la partita i cambiamenti vengono applicati | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-011  REQ-011 | **Nome:** | Verifica del calcolo del punteggio |
| **Descrizione:** | Quando il giocatore indovina la parola, viene assegnato un punteggio in base al tempo rimasto alla fine della partita | | |
| **Prerequisiti:** | Devo avere indovinato tutte le lettere della parola corrente | | |
| **Procedura:** | 1. Indovino le lettere della parola 2. Viene aggiornato il punteggio nella lista dei giocatori | | |
| **Risultati attesi:** | Il punteggio del giocatore viene calcolato e mostrato correttamente nella lista dei giocatori | | |

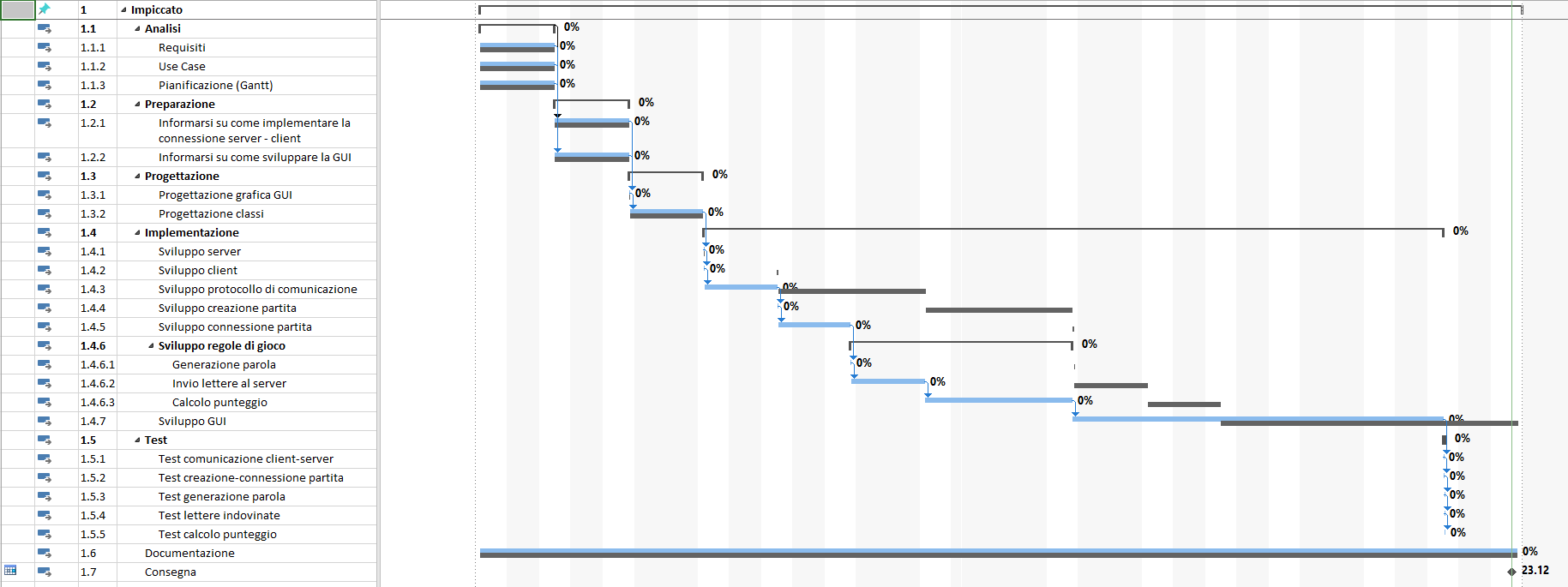
## Risultati test

|  |  |  |  |
| --- | --- | --- | --- |
| Test case | Funzionamento | Commento | Data |
| TC-001 | OK | Ho seguito la procedura e nella console del server viene stampato il token della partita | 16.12.2021 |
| TC-002 | OK | Ho seguito la procedura e l’applicazione mostra gli errori adeguati | 16.12.2021 |
| TC-003 | OK | Ho seguito la procedura e una volta premuto il bottone per unirmi mi trovo davanti alla schermata di aspetto | 16.12.2021 |
| TC-004 | NON OK | Ho seguito la procedura ma non viene stampato niente nella chat | 16.12.2021 |
| TC-004 | OK | Dopo aver messo a posto il problema nella chat vengono stampate le lettere scritte da altri giocatori | 16.12.2021 |
| TC-005 | OK | La lista dei giocatori viene stampata correttamente e viene aggiornata una volta che altri giocatori si uniscono alla partita | 16.12.2021 |
| TC-006 | OK | Non ho seguito la procedura descritta, ma visto che le parole sono contenute in un file, ho semplicemente guardato il conteggio delle righe | 16.12.2021 |
| TC-007 | NON OK | Se la parola dovesse contenere una lettera accentata, il server non riuscirebbe a gestirla correttamente | 16.12.2021 |
| TC-008 | OK | Ho seguito la procedura e dopo aver digitato una lettera sbagliata il conteggio degli errori sale.  Una volta raggiunti dieci errori il turno finisce | 16.12.2021 |
| TC-009 | OK | Ho seguito la procedura e alla fine della partita viene stampata la classifica della partita (i giocatori sono ordinati per punteggio) | 16.12.2021 |
| TC-010 | OK | Ho seguito la procedura e dopo aver inserito correttamente i parametri della partita trovo le modifiche mentre gioco | 16.12.2021 |
| TC-011 | OK | Ho seguito la procedura e dopo aver indovinato la parola e dopo la fine del turno il mio punteggio viene aggiornato | 16.12.2021 |

## Mancanze/limitazioni conosciute

Purtroppo, non sono riuscito a gestire i caratteri accentati nel gioco, questo perché mandando solo un byte per lettera. Visto che Java gestisce i caratteri con utf-16 avrei fare uno studio sui bit in modo da capire quali lettere dovevo convertire in utf-8 e quali potevo lasciare così. Visto il poco tempo che mi è rimasto dopo aver controllato, documentato e implementato tutto quello che ho potuto non sono riuscito ad implementare questo aspetto.

# Consuntivo



Blu: come è andata davvero

Nero: quello che avevo pianificato

Secondo me è andata diversamente da come avevo pianificato perché non avevo mai utilizzato grafica con elementi in java e mi sono quindi dato un grandissimo margine di manovra

# Conclusioni

Questo applicativo è stato sviluppato a scopo ludico, non comporta quindi nessuna miglioria a qualcosa di già esistente. Il progetto è stato sviluppato seguendo i requisiti del committente per fare in modo di avere un applicativo semplice da utilizzare che permetta agli utenti di giocare in momenti di noia.

## Sviluppi futuri

Non ho previsto alcuno sviluppo futuro per questo progetto, ma pensandoci bene come aggiunte si potrebbe aggiungere un database che salva le partite e degli indizi che vengono dati se viene superata certa soglia temporale

## Considerazioni personali

Devo dire che questo progetto mi ha aiutato a consolidare alcuni aspetti della programmazione in Java ed a oggetti, facendomi scoprire il mondo della programmazione concorrente leggermente prima dei miei compagni. Questo mi ha permesso di avere una marcia in più. Sono contento che sono riuscito a rispettare le consegne e le tempistiche, questo mi ha permesso di capire l’importanza di una progettazione fatta bene.

# Bibliografia

## Bibliografia per articoli di riviste:

Non ho utilizzato articoli da riviste

## Bibliografia per libri

Non ho utilizzato libri

## Sitografia

<https://github.com/napolux/paroleitaliane>, consultato il 30-09-2021

<https://www.youtube.com/watch?v=ZIzoesrHHQo>, consultato il 30-09-2021

<https://www.softwaretestinghelp.com/java/java-timer-tutorial>, consultato il 14-10-2021

<https://docs.oracle.com/en/java/javase/16/docs/api/index.html>, consultato il 16-12-2021

# Allegati

Elenco allegati

* Diari di lavoro
* Codici sorgente
* QdC
* File jar del progetto + cartella con i files