Cat Lady - Final Report

Serafino Pandolfini
serafino.pandolfini@studio.unibo.it

Laura Leonardi
laura.leonardi9@studio.unibo.it

September 2024

Il progetto prevede lo sviluppo del gioco da tavolo **Cat Lady** in versione online. Ogni giocatore può creare o unirsi a una lobby creata da un altro giocatore. Ogni lobby può contenere fino a 4 giocatori, raggiunto il numero minimo di 2 partecipanti è possibile iniziare una partita. Il gioco si svolge a turni, ad ogni turno il giocatore può acquisire una riga o colonna da una matrice 3x3 di carte e giocare specifiche carte per avvantaggiarsi. Il gioco termina quando non ci sono più carte per riempire la matrice. L'obiettivo del gioco è quello di ottenere il maggior numero di punti vittoria. I punti vittoria sono ottenibili tramite le varie tipologie di carte presenti nel gioco.

Indice

1 Obiettivi e Analisi dei Requisiti

L'obiettivo del progetto è quello di realizzare una versione distribuita del gioco da tavolo **Cat Lady** [1].

1.1 Requisiti funzionali

In seguito sono riportati in forma di domanda-risposta i vari requisiti funzionali che l'applicazione dovrà rispettare.

- **Come può venire creata una lobby?**
  Quando l'applicazione viene avviata offre la possibilità a ogni utente di creare una propria lobby, che può essere pubblica o privata.

- **Cosa differenzia le tipologie di lobby?**
  Una lobby pubblica può essere visualizzata da tutti i giocatori, quando viene inizialmente eseguita l'applicazione viene fornito all'utente l'elenco delle lobby pubbliche a cui può accedere. Una lobby privata richiede all'utente che ne vuole fare parte sia il nome della lobby che un codice di accesso fornito al creatore della lobby.

- **Come si può entrare in una lobby?**
  Per accedere a una lobby pubblica è necessario selezionarla dall'elenco e inserire un nome utente. Nel caso si voglia accedere a una lobby privata è necessario inserire, oltre al proprio nome utente, anche il nome della lobby e il suo codice di accesso. Nel caso l'utente selezioni un nome già in uso all'interno della lobby gli verrà chiesto di utilizzare un nome differente.

- **Quanti giocatori può contenere una lobby?**
  Una lobby può contenere da 1 fino a 4 giocatori. Le lobby pubbliche che hanno raggiunto il numero massimo di giocatori non vengono più mostrate nell'elenco delle lobby pubbliche. Nel caso un giocatore provasse ad accedere a una lobby già con 4 giocatori, la sua richiesta verrà respinta specificandone il motivo.

- **Come avviene la comunicazione tra giocatori all'interno di una lobby?**
  I giocatori all'interno di una lobby possono comunicare tra loro tramite la chat della lobby. Quando un nuovo giocatore si unisce alla lobby questo potrà vedere tutti i messaggi inviati dal suo accesso in poi ma non potrà vedere i messaggi mandati prima del suo ingresso nella lobby.

- **Chi e in che modo può avviare la partita?**
  Solo il creatore di una lobby può avviare una partita. Per poter iniziare una partita è richiesto che siano presenti almeno 2 giocatori nella lobby.

- **Come si svolge una partita?**
  Una partita si suddivide in tre fasi: una iniziale in cui i giocatori eseguono i loro turni, pescando e giocando carte. Dopo aver terminato il mazzo di carte a disposizione i giocatori dovranno utilizzare le loro risorse per assegnarle alle loro

carte gatto. Infine dopo che tutti i giocatori hanno completato l'assegnamento delle risorse viene mostrata la classifica finale dei punteggi. Raggiunta la classifica finale la partita è considerata conclusa. Durante ogni momento della partita ai giocatori è consentito di abbandonare la lobby.

- **Come è composta la dinamica di gioco?**
  Il gioco è composto dalle seguenti tipologie di carte (si utilizza la notazione x/y/z per indicare il numero di carte per partite rispettivamente da 2, 3 e 4 giocatori):
  - 15/21/22 `Cats`: Ognuna di queste carte rappresenta un gatto. Quando viene ottenuta una di queste carte viene posizionata davanti al giocatore che l'ha pescata. A ogni gatto sono assegnati da 1 a 3 colori (nero, arancione e/o bianco). Alcune carte gatto possiedono effetti che garantiscono punti in base a specifiche condizioni indicate sulla carta stessa. A fine della partita ogni gatto deve essere nutrito utilizzando `Food Resources` della tipologia indicata sulla carta gatto. Se il gatto viene nutrito si ricevono punti vittoria pari a quanto indicato sulla carta e i punti dell'eventuale effetto. Nel caso non venga nutrito per mancanza di risorse si perdono 2 punti vittoria.
  - 13/13/13 `Stray Cats`: Uguali alle carte gatto, non sono presenti nel mazzo e possono essere ottenute tramite l'utilizzo della carta `Lost Cat`. Ognuno di loro possiede un effetto specifico.
  - 26/30/34 `Food`: quando viene ottenuta una di queste carte viene subito utilizzata e si ottiene la `Food Resource` indicata su di essa. Ci sono 4 tipologie di carte: chicken, tuna, milk e wild. Alcune carte permettono di ottenere 2 unità della risorsa rappresentata. A fine partita il giocatore con più cibo avanzato (cioè non utilizzato per nutrire gatti) perde 2 punti vittoria, in caso di pareggio tra due o più giocatori tutti perdono 2 punti vittoria. Nel caso nessun giocatore abbia cibo avanzato non vengono applicate penalità al punteggio.
  - 7/8/9 `Costumes`: Queste carte rappresentano costumi per i gatti. Quando una di queste carte viene ottenuta viene tenuta nella mano del giocatore. A fine partita il giocatore con più costumi in mano guadagna 6 punti vittoria, in caso di pareggio i punti sono divisi equamente. Se a fine partita un giocatore non ha nessun costume in mano perde 2 punti vittoria.
  - 5/6/7 `Catnip`: Queste carte rappresentano l'erba gatta. Quando una di queste carte viene ottenuta viene tenuta nella mano del giocatore. A fine della partita se si possiede una sola carta di questo tipo si perdono 2 punti vittoria, se si possiedono 2 o 3 carte si riceve 1 punto vittoria per ogni tuo gatto nutrito, se si possiedono 4 o più carte si ricevono 2 punti vittoria per ogni tuo gatto nutrito.
  - 10/10/15 `Toys`: Queste carte rappresentano giocattoli per gatti. Quando una di queste carte viene ottenuta viene tenuta nella mano del giocatore. Ci sono 5 diverse tipologie di giocattoli. Alla fine della partita si ricevono punti in

base a ogni set di carte giocattolo diverse nella tua mano. I punti ottenuti sono basati sulla dimensione di ogni set come indicato dalla tabella 1.

Carte giocattolo diverse	1	2	3	4	5
Punti vittoria	1	3	5	8	12

Tabella 1: Punti assegnati per le carte giocattolo.

- 8/10/10 `Lost Cat`: Quando una di queste carte viene ottenuta viene tenuta nella mano del giocatore. Questa carta può essere giocata in coppia assieme a un altra carta Lost Cat. Quando viene giocata il giocatore può scegliere una di due azioni: Ottenere uno degli Stray Cat e usarlo come carta gatto oppure ottenere 2 punti vittoria.
- 3/4/5 `Spray Bottles`: Quando una di queste carte viene ottenuta viene tenuta nella mano del giocatore. Questa carta può essere giocata con l'effetto di spostare la posizione del Cat Token su una riga o colonna a piacere.

Gli altri elementi del gioco presenti sono:

- `Cat Token`: Questo token viene posizionato sull'ultima riga o colonna da cui un giocatore ha pescato. Un giocatore non può pescare la riga o colonna di carte su cui si trova il Cat Token. È possibile spostare il Cat Token durante il proprio turno utilizzando una carta Spray Bottle.
- `Food Resources`: Rappresentano le risorse di cibo ottenute tramite le carte Food. Il numero di risorse che un giocatore possiede è visibile a tutti i giocatori. In totale sono presenti risorse per 20 chicken, 20 tuna, 15 milk e 5 wild. Le risorse wild possono essere utilizzare come jolly per qualsiasi altra risorsa.

A inizio di ogni partita il mazzo composto da tutte le carte ad eccezione degli Stray Cats. Il mazzo viene mescolato e viene creata la matrice 3x3 iniziale. Vengono selezionati 3 dei 13 Stray Cats e resi visibili ai giocatori. Il primo giocatore, che coincide sempre con il creatore della lobby, inizia la partita. Il turno si svolge con le seguenti azioni:

- (facoltativo) Il giocatore gioca Lost Cat o Spray Bottle.
- Il giocatore prende una riga o colonna di carte dalla matrice. Questa viene riempita con nuove carte dal mazzo e viene posizionato il Cat Token sulla riga o colonna da cui il giocatore ha pescato.
- Le carte pescate, in base alla loro tipologia, saranno immediatamente utilizzate, posizionate davanti al giocatore o aggiunte alla mano del giocatore.
- (facoltativo) Il giocatore gioca Lost Cat o Spray Bottle.

- **Cosa succede quando il mazzo di carte termina?**
  Nel momento in cui non sono presenti abbastanza carte per completare la matrice, al termine del turno corrente si procede con la fase di assegnamento delle risorse.

- **Come funziona la fase di assegnamento delle risorse?**
  Ogni giocatore utilizza le `Food Resources` accumulate per nutrire i propri `Cats` e `Stray Cats`.

- **Cosa succede quando tutti i giocatori hanno terminato l'assegnamento delle risorse?**
  Dopo che ogni giocatore ha completato l'assegnamento delle risorse viene calcolato il punteggio finale di ogni giocatore e viene decretato il vincitore.

- **Durante la partita, quali informazioni sono note al giocatore?**
  Un giocatore conosce sempre le carte che ha in mano, le sue risorse, le carte gatto che ha in gioco, il numero di carte nel mazzo, gli stray cats ancora disponibili e le carte nella matrice. Inoltre conosce, per ogni altro giocatore, le loro carte gatto e il numero di carte nella loro mano.

- **Come viene gestito il tempo che il giocatore ha a disposizione per effettuare il tuo turno?**
  Durante la prima fase di gioco, per impedire ai giocatori di rallentare eccessivamente il ritmo del gioco, viene utilizzato un timer di 2 minuti. Allo scadere del timer il turno del giocatore corrente termina forzatamente, indipendentemente dalle azioni effettuate. Questo timer non viene applicato nella fase di assegnamento delle risorse in quanto la sua maggiore complessità dal punto di vista strategico potrebbe potenzialmente richiedere più tempo.

- **Cosa succede se un giocatore abbandona la lobby o perde la connessione con il server mentre si trova in una lobby?** Nel caso un giocatore dovesse abbandonare la lobby, a seconda della specifica situazione viene adottato un comportamento differente:
  - Se la partita non è ancora iniziata il giocatore abbandona la lobby senza ulteriori conseguenze sugli altri partecipanti della lobby. Nel caso sia il creatore della lobby ad abbandonarla allora la lobby viene cancellata e gli utenti al suo interno vengono rimossi forzatamente.
  - Se la partita è in corso il giocatore viene eliminato dalla sequenza dei turni. I suoi dati relativi alla partita rimangono visualizzabili agli altri giocatori. Questo si verifica anche nel caso sia il creatore della lobby ad abbandonare mentre la partita è in corso al fine di prevenire comportamenti scorretti.
  - Se durante la partita un singolo giocatore dovesse rimanere nella lobby questo sarà decretato automaticamente vincitore e si passerà alla visualizzazione dei punteggi.

1.2 Requisiti non funzionali

I requisiti non funzionali si declinano in due categorie: quelli relativi all'aspetto architetturale e quelli relativi alla user experience dei giocatori.

- L'interfaccia grafica deve essere intuitiva per i giocatori e al contempo mostrare tutte le informazioni rilevanti.
- Agli utenti deve essere chiaro quando eseguono azioni normalmente non consentite tramite appositi messaggi di errore.
- Agli utenti non devono mai essere comunicate informazioni relative ad altri utenti di cui non dovrebbero venire a conoscenza, ad esempio le carte nella loro mano. Questo serve a prevenire l'utilizzo di client modificati in grado di ottenere queste informazioni e sfruttarle per compromettere il gioco.
- Al termine di ogni azione lo stato del server e quello dei client in merito alle lobby e all'eventuale partita in corso devono corrispondere, nel caso di inconsistenze dovute ad alterazioni dei client queste vanno segnalate dal server ai client tramite appositi messaggi di errore.
- L'architettura realizzata deve permettere di gestire più lobby e partite in simultanea.
- L'architettura deve essere in grado di gestire disconnessioni improvvise in modo opportuno, nello specifico la disconnessione di un client mentre si trova in una lobby deve essere considerato come se il giocatore avesse abbandonato la lobby.

Inoltre, sono presenti dei requisiti tecnologici necessari al funzionamento del sistema realizzato. A seguito della fase di analisi e della definizione dell'architettura si è deciso di utilizzare: **java.net.Socket** e **java.net.ServerSocket** per la connessione, per lo scambio di informazioni tra client e server sarà utilizzata la libreria **com.google.gson.Gson** [2] e per la GUI sarà utilizzato **Java Swing**.

1.3 Scenari

Gli utenti che utilizzano l'applicazione possono trovarsi nelle seguenti situazioni:

1.3.1 Scenario: avvio dell'applicazione

All'avvio dell'applicazione l'utente può o creare una nuova lobby o partecipare a una lobby già esistente selezionandola dall'elenco delle lobby pubbliche o inserendo nome e codice di accesso di una lobby privata. L'utente può anche aggiornare le lobby pubbliche per verificare la presenza di nuove lobby non precedentemente presenti.

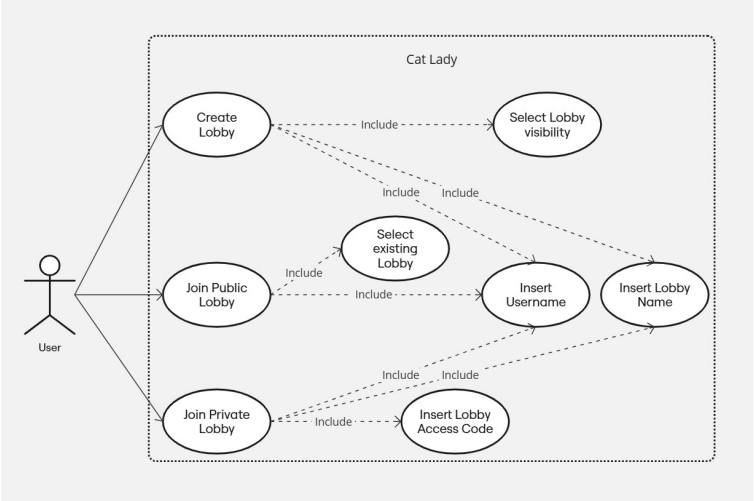

Figura 1: Diagramma dei casi d'uso per l'avvio dell'applicazione.

1.3.2 Scenario: lobby

Dopo essere entrato in una lobby l'utente può utilizzare la chat per mandare messaggi oppure abbandonare la lobby. Nel caso l'utente sia anche il creatore della lobby può decidere di iniziare la partita.

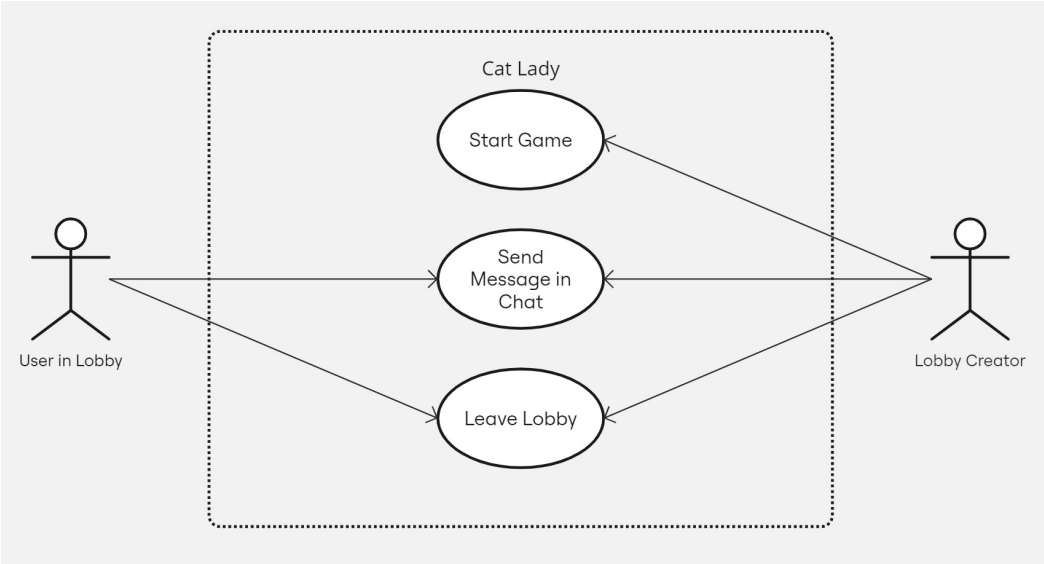

Figura 2: Diagramma dei casi d'uso per le azioni interne alla lobby.

1.3.3 Scenario: inizio del gioco

All'inizio della partita viene selezionato il mazzo di carte da utilizzare sulla base del numero di giocatori, formata la matrice iniziale e preparati tre stray cats da utilizzare durante la partita. Durante il suo turno il giocatore può pescare una linea (riga o colonna) di carte dalla matrice, giocare una carta spray bottle o una coppia di carte lost cat e terminare il suo turno. Inoltre, cliccando una carta sia della propria mano che in gioco, è possibile visualizzare la carta stessa in modo da poter leggere il suo funzionamento. Ogni giocatore può, in ogni momento, abbandonare la partita.

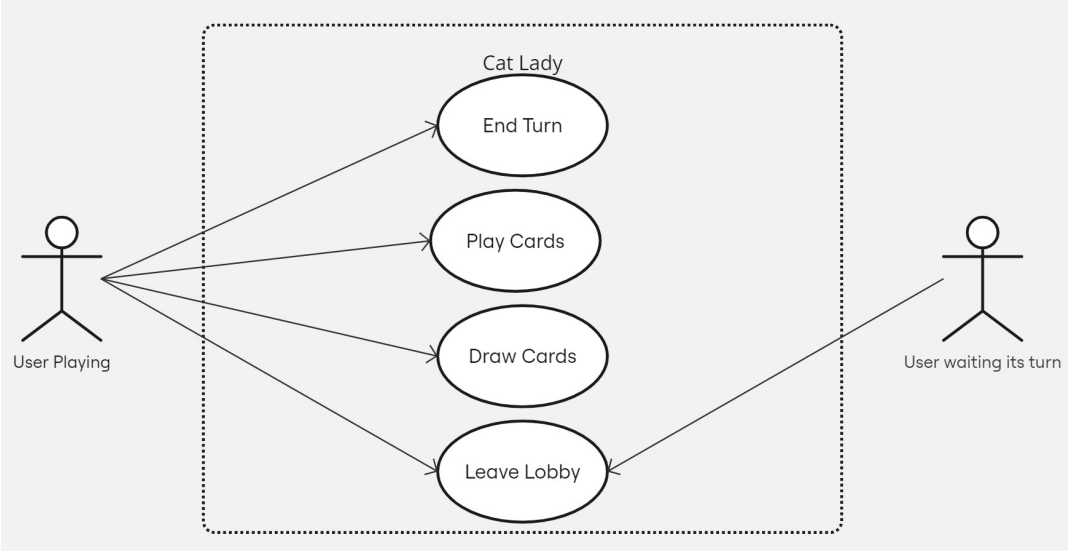

Figura 3: Diagramma dei casi d'uso per la fase iniziale di gioco.

1.3.4 Scenario: assegnamento delle risorse

Terminato il mazzo di carte ogni giocatore assegna alle carte gatto ottenute durante la partita le risorse che ha accumulato. Nel momento in cui il giocatore si ritiene soddisfatto del modo con cui ha distribuito le risorse può inviare la richiesta per ottenere il punteggio finale. Ogni giocatore può, in ogni momento, abbandonare la partita.

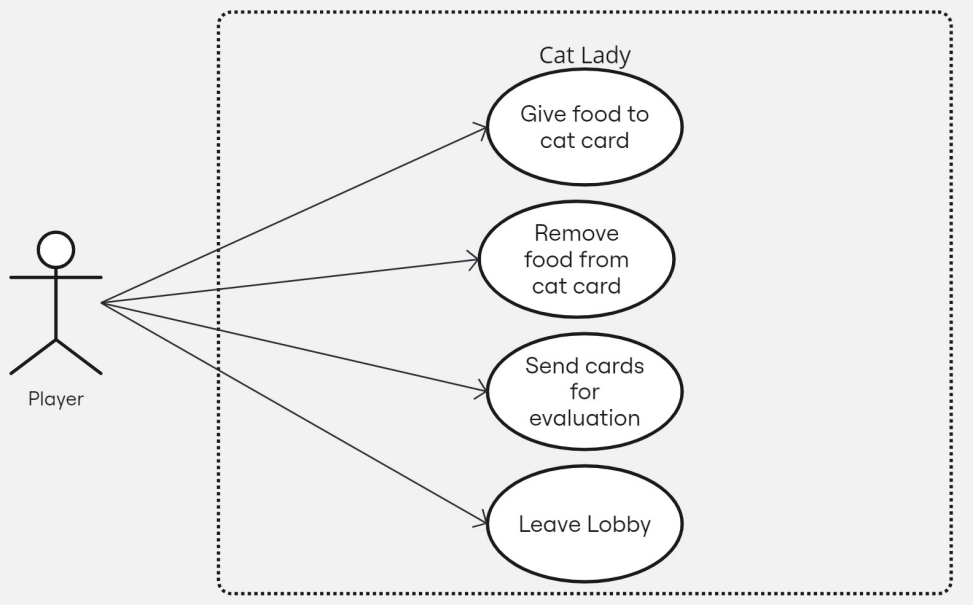

Figura 4: Diagramma dei casi d'uso per la fase di assegnamento delle risorse.

1.3.5 Scenario: fine della partita

Dopo che tutti i giocatori hanno completato l'assegnamento delle risorse (o abbandonato la partita) viene mostrata la schermata con i punteggi finali per ogni categoria e indicati i giocatori con più punti accumulati. Raggiunto questo punto del gioco l'utente può ritornare alla lobby in attesa di una nuova partita o abbandonarla.

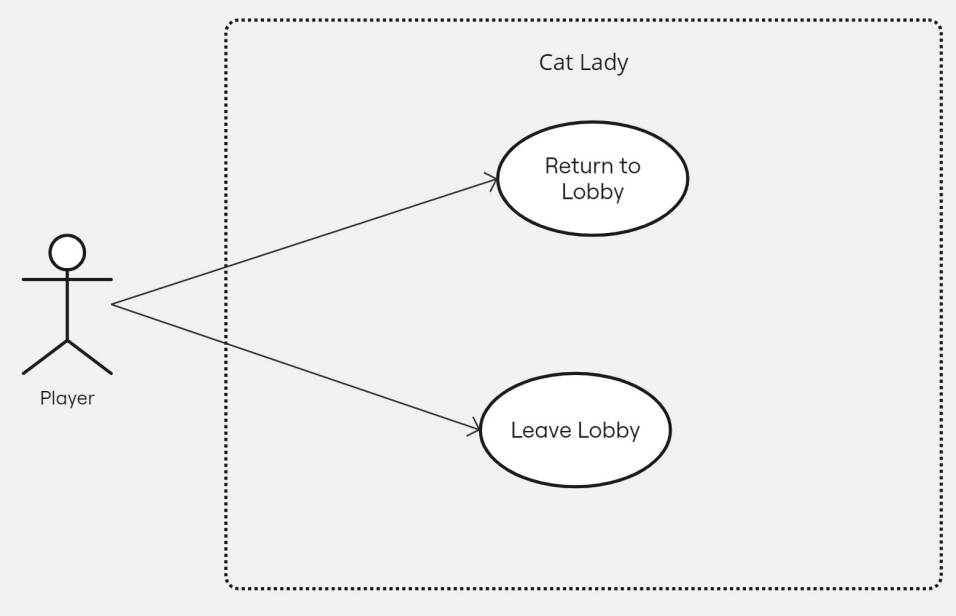

Figura 5: Diagramma dei casi d'uso per la conclusione della partita.

1.4 Politiche di validazione

La qualità del prodotto finale sarà valutata tramite i framework **JUnit** [3] e **Mockito** [5]. Considerando la natura del progetto, gli aspetti principali di cui testare il funzionamento sono la logica del gioco e delle lobby, la connessione tra client e server e l'interfaccia grafica. Si prevede di testare i vari aspetti della logica del gioco e della lobby tramite classici test **JUnit**; per la parte di connessione saranno realizzati appositi test per verificare la corretta serializzazione e deserializzazione delle informazioni e, tramite **Mockito**, creati dei test per verificare l'interazione tra client e server. Per quanto riguarda gli aspetti relativi all'interfaccia grafica si prevede di effettuare test manuali per verificarne il corretto funzionamento.

2 Design

Per la realizzazione di questo progetto sono state inizialmente prese in considerazione due possibili architetture: **client-server** e **peer-to-peer**. Durante l'analisi dei requisiti sono emersi diverse possibili problematiche relative all'approccio **peer-to-peer** per la nostra applicazione. In particolare, la mancanza di un autorità centrale che mantenesse e verificasse lo stato del sistema avrebbe permesso di effettuare azioni non legittime durante le partite, di conseguenza si è deciso per un architettura **client-server**.

La comunicazione bidirezionale tra client e server avviene tramite una connessione TCP: il server mantiene il **socket** di ogni client che si connette e lo elimina al momento della disconnessione.

La comunicazione avviene nel seguente modo:

1. Il client invia una richiesta al server.
2. Il server processa la richeista e produce una risposta da inviare al client.
3. Nel caso la richiesta del client abbia portato a cambiamenti nello stato del server, questo invia una notifica a ogni altro client interessato al cambiamento.
4. Talvolta, ad esempio durante lo scadere del timer del turno corrente, è il server a inviare notifiche senza che abbia ricevuto alcuna richiesta.

Quasi ogni cambiamento relativo allo stato del client è dovuto a una risposta o notifica da parte del server, questo serve a prevenire che un cambiamento avvenga prima che il server lo abbia autorizzato verificando la sua legittimità.

2.1 Struttura del Client

Il client è stato realizzato utilizzando il pattern architetturale **MVC**. La scelta di questa architettura è prevalentemente dovuta alla necessità di separare l'aspetto di connessione e logica dall'aspetto grafico dell'applicazione.
La struttura generale del client è mostrata in figura 6.

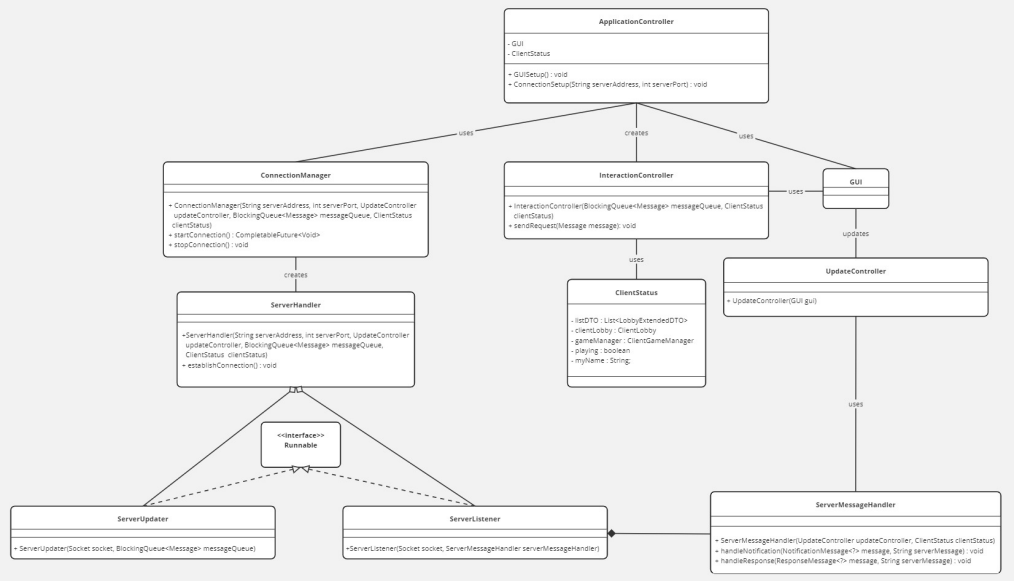

Figura 6: Diagramma delle classi rappresentante l'architettura generale del client.

La classe `ApplicationController` è il controller principale dell'applicazione che gestisce sia la parte di scambio dei messaggi che la parte di view.

La gestione dello scambio dei messaggi avviene tramite il `ConnectionManager` che esegue in modo asincrono il `ServerHandler` che si occupa di stabilire la connessione. La verifica del successo della connessione avviene tramite il risultato della **CompletableFuture** ritornata dal metodo `startConnection`.

Dopo aver stabilito la connessione con il server, il `ServerHandler` crea e assegna a degli **ExecutorService** il `ServerListener`, che gestirà i messaggi in arrivo sul client e il `ServerUpdater` che invierà messaggi al server. Per la gestione dei messaggi in arrivo viene utilizzato il `ServerMessageHandler` che si occupa di processare ogni messaggio a seconda della sua tipologia e di aggiornare lo stato del client e la `GUI` tramite `UpdateController`.

L'invio delle richieste al server parte da un iteraizone dell'utente con la `GUI`, che richiede all' `InteractionController` di inviare la richiesta. L' `InteractionController` verifica tramite il client status che la richiesta sia eseguibile (secondo il client) e la inserisce nella `messageQueue`. Il `ServerUpdater`, nel momento in cui trova un messaggio nella coda, lo estrae, lo serializza e lo invia al server.

2.2 Struttura del Server

Per la gestione dello scambio dei messaggi il server replica la struttura utilizzata dal client (figura 7).

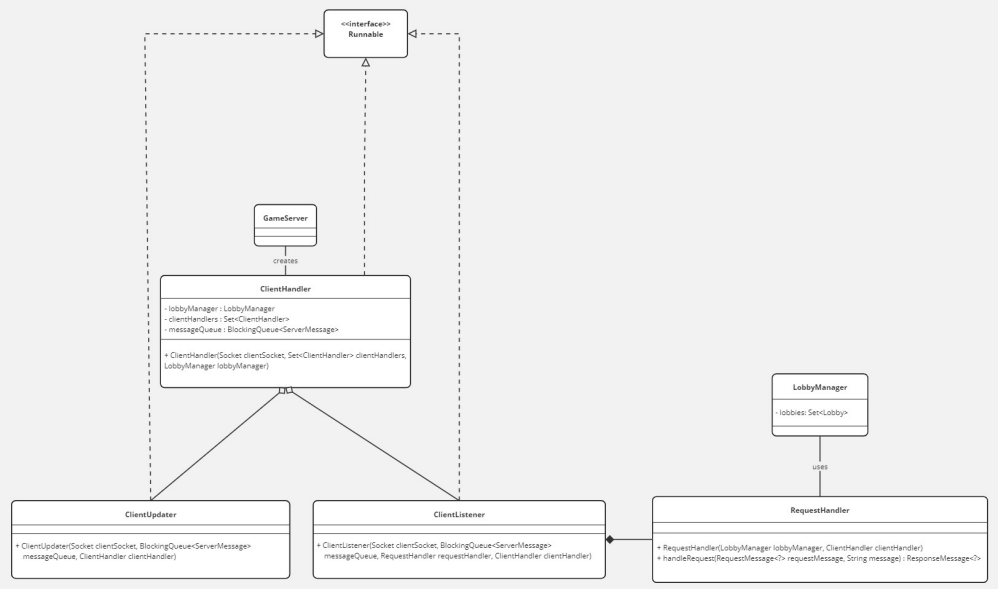

Figura 7: Diagramma delle classi rappresentante la struttura del server e la sua gestione dei messaggi.

I messaggi sono ricevuti dal `ClientHandler` associato all'utente che invia il messaggio, questi sono gestiti dal `RequestHandler` che elabora le richieste, aggiorna lo stato del server definito in `LobbyManager` e produce una risposta appropriata che viene inserita nella `messageQueue` del server. Nel caso siano prodotte notifiche per gli altri client connessi alla stessa lobby questi sono inseriti nelle rispettive code dal `LobbyManager`. Le risposte e le notifiche prodotte dal server sono inviate tramite i rispettivi `ClientUpdater` associati agli utenti destinatari.

In figura 8 è presentata la struttura della logica delle lobby e delle partite in corso dal punto di vista del server.

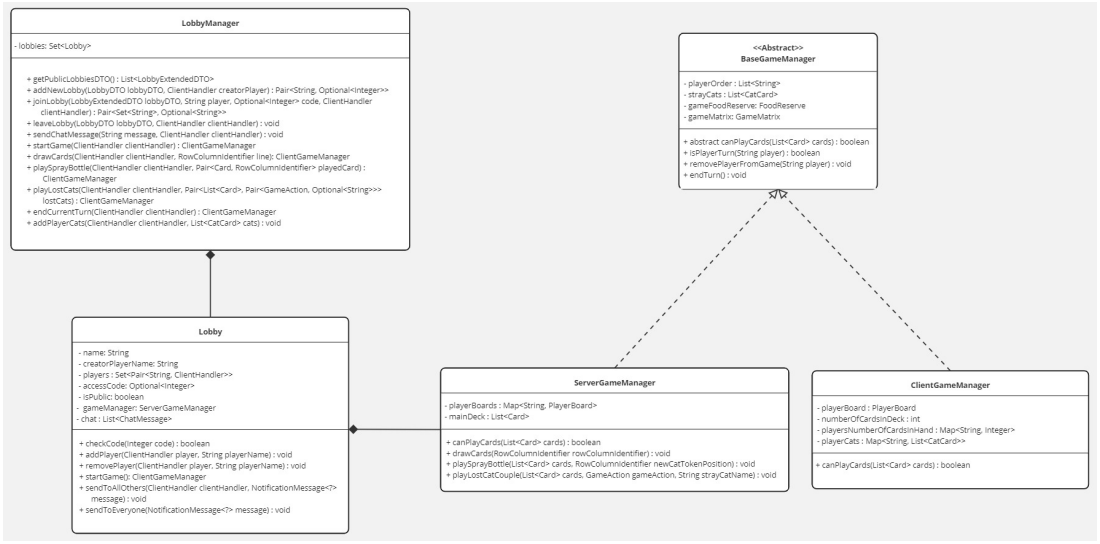

Figura 8: Diagramma delle classi rappresentante la logica per la gestione delle lobby e del gioco.

`LobbyManager` si occupa di gestire tutte le lobby attualmente attive e gestisce tutte le interazioni relative alle lobby e alle partite. I suoi metodi servono ad aggiornare lo stato delle lobby e delle partite a seguito delle richieste degli utenti e a verificare che queste siano legittime.

`Lobby` modella la singola lobby, dal suo nome identificativo ai giocatori al suo interno alla chat tra i partecipanti. Ogni `Lobby` è associata a un `ServerGameManager` nel momento in cui inizia una partita. Il `ServerGameManager` modella lo stato del gioco per tutti gli utenti partecipanti e le azioni eseguibili durante la partita. Per aggiornare gli utenti sullo stato corrente del gioco dopo ogni azione, le informazioni del `ServerGameManager` sono utilizzate per produrre un `ClientGameManager` distinto per ogni client. Questo è fondamentale per evitare che le informazioni di gioco di ogni giocatore siano rivelate a tutti gli altri partecipanti.

Il model del gioco è sommariamente esposto in figura 9, rappresentante le entità principali utilizzate.

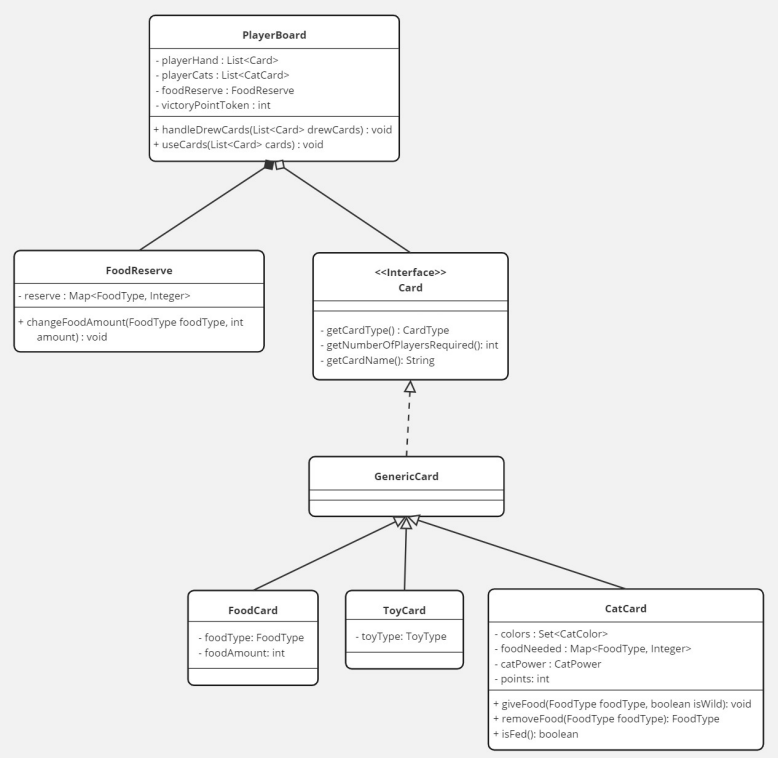

Figura 9: Diagramma delle classi rappresentante una versione ridotta del model del gioco.

`PlayerBoard` definisce lo stato di un giocatore e gestisce le azioni di pescare e giocare carte. Per modellare le risorse di cibo raccolte dal giocatore viene utilizzata la `FoodReserve` che conserva il cibo accumulato dal giocatore e ne permette l'utilizzo nella fase di assegnamento delle risorse.

L'interfaccia `Card` modella la generica carta utilizzata all'interno del gioco. La sua implementazione, `GenericCard` implementa i metodi comuni a tutte le tipologie di carte. Tra le varie tipologie di carte le più importanti sono le `CatCard` che modellano sia le carte gatto normali che gli stray cats. Ogni gatto è definito dal suo nome e dalle risorse di cibo necessario per sfamarlo. La `CatCard` fornisce metodi per assegnare cibo al gatto e rimuverlo, fondamentali per la fase di assegnamento delle risorse e per il calcolo del punteggio finale. Ogni gatto può possedere un potere che ne modifica i punti guadagnati durante la fase finale del gioco.

2.3 Struttura dei Messaggi

All'interno del progetto sono utilizzate tre tipologie di messaggi mostrati in figura 10: `Request`, `Response` e `Notification Message`. Response e Notification Message implementano l'interfaccia `ServerMessage` che a sua volta estende l'interfaccia `Message`; Request invece implementa direttamente `Message`. Ogni tipologia di messaggio è dotato

di un campo type che identifica il tipo di messaggio inviato, utilizzato per deserializzare correttamente il contenuto del messaggio.

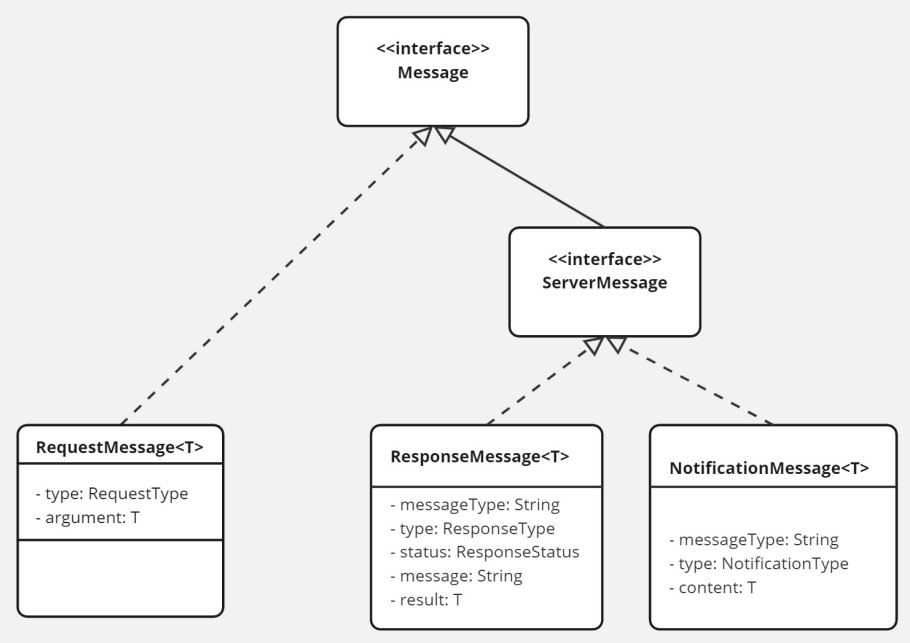

Figura 10: Diagramma delle classi rappresentante i messaggi inviabili e ricevibili da client e server.

I `RequestMessage` sono utilizzati dal client per inviare richieste al server, queste possono essere di dviersa tipologia come richiedere tutte le lobby pubbliche a cui si può accedere o pescare carte durante la partita.

I `ResponseMessage` sono inviati dal server a seguito di una richiesta da parte di un client. Il campo status indica l'esito della richiesta, sommariamente divisibile in successo o fallimento; il campo message viene utilizzato per associare un messaggio che spieghi l'esito della richiesta, normalmente usato per comunicare il motivo specifico del fallimento. Il risultato della risposta contiene o le informazioni richieste dal client o il nuovo stato della lobby o della partita.

I `NotificationMessage` sono inviati dal server per informare i client di un cambiamento dello stato della lobby o della partita in cui si trovano. Un loro utilizzo è, ad esempio, indicare che un giocatore è entrato nella lobby o che un giocatore ha terminato il suo turno.

In seguito (figura 11) sono elencati tutti le tipologie di messaggio e di status della risposta utilizzati.

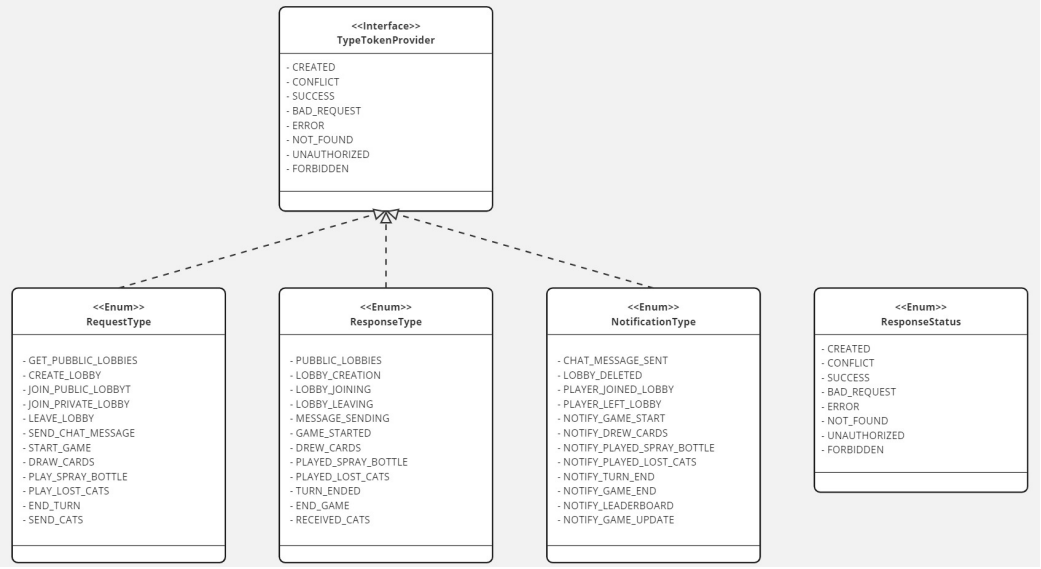

Figura 11: Diagramma delle classi rappresentante le tipologie di messaggi e i possibili status delle risposte.

2.4 Comportamento del sistema

All'avvio dell'applicazione l'utente si trova in una schermata iniziale da cui può creare o unirsi a una nuova lobby. Per poter creare una nuova lobby è sufficiente inserire il nome della lobby, il nome dell'utente all'interno della lobby e indicare se si vuole creare una lobby pubblica o privata. Per unirsi a una nuova lobby l'utente può scegliere una delle lobby pubbliche presenti oppure inserire nome e codice di una lobby privata. In entrambi i casi per unirsi a una lobby è necessario che siano soddisfatte le seguenti condizioni: l'utente deve aver inserito un nome non già in uso all'interno della lobby, la lobby non deve essere piena e non ci devono essere partite in corso. Inoltre, per le lobby private, è richiesto che il codice inserito sia corretto.
All'interno della lobby gli utenti attendono l'inizio della partita. Il giocatore che ha creato la lobby può avviare la partita a condizione che siano presenti almeno due giocatori nella lobby. In ogni momento ogni giocatore può abbandonare la lobby, se a farlo è il creatore della lobby questa viene cancellata e tutti i giocatori rimossi.
All'inizio della partita è sempre il giocatore che ha creato la lobby ad effettuare il primo turno, durante lo svolgimento del turno il giocatore può pescare e giocare carte, solo dopo aver pescato il giocatore può terminare il suo turno. Se il giocatore non completa il suo turno entro due minuti il turno viene terminato forzatamente e si passa al giocatore successivo.
Nel momento in cui non sono più presenti abbastanza carte nel mazzo per popolare completamente la matrice da cui le carte sono pescate il gioco entra nella fase di assegnamento delle risorse. Quando tutti i giocatori hanno terminato questa fase viene mostrata la classifica finale e l'utente può o ritornare alla lobby o abbandonarla.

Questo comportamento è descritto dal diagramma degli stati in figura 12.

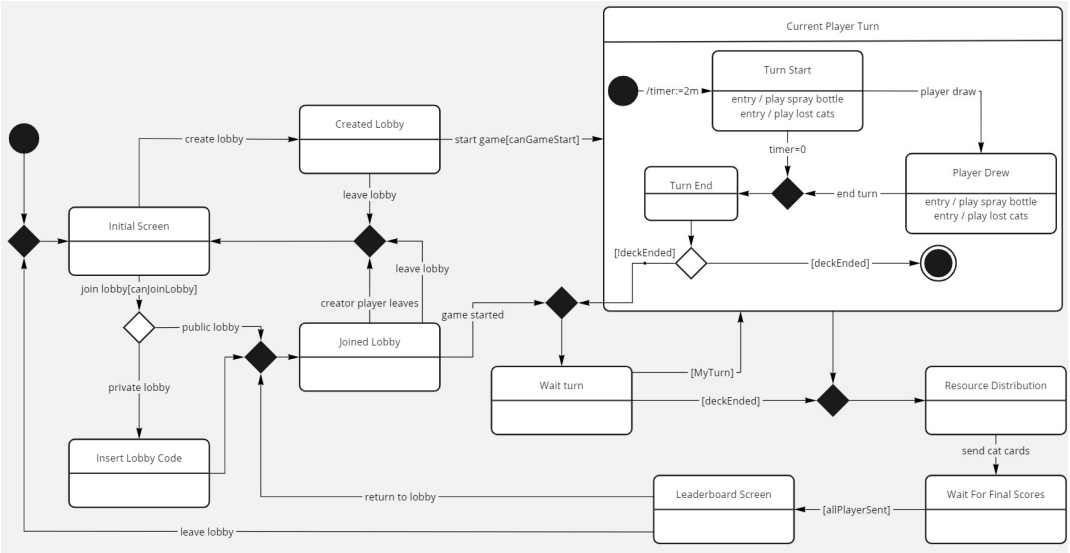

Figura 12: Diagramma degli stati per il comportamento dell'utente all'interno del sistema

Oltre a quanto riportato nel diagramma, l'utente può abbandonare la lobby anche durante la partita, sia per sua decisione che per una possibile disconnessione. Nel caso sia il turno del giocatore che abbandona la lobby il suo turno termina automaticamente, mentre se ciò avviene nella fase di assegnamento delle risorse viene solamente escluso dalla classifica finale. Inoltre, dovesse rimanere un unico giocatore all'interno della lobby, questo sarà automaticamente il vincitore e si passerà direttamente alla classifica finale.

2.5 Interazione

Il sistema realizzato prevede che, per ogni client che si connette al server, siano creati da quest'ultimo un `ClientHandler`, a sua volta responsabile per la creazione di un `ClientListener` e di un `ClientUpdater`, responsabili rispettivamente per la ricezione e l'invio di messaggi. Il client invia messaggi al server tramite `ServerUpdater` e li riceve tramite `ServerListener`, le rispettive controparti per l'invio e la ricezione dei messaggi. Quando una richiesta viene ricevuta dal server questa viene inviata al `RequestHandler` che processa la richiesta, verifica la sua correttezza, invia le eventuali notifiche e produce una risposta da inviare al client. Il client dopo aver ricevuto la risposta del server la invia al `ServerMessageHandler`, controparte del `RequestHandler`, che la processa e aggiorna sia il model che la GUI.

In figura 13 viene descritta l'azione di pescare delle carte da parte di un utente.

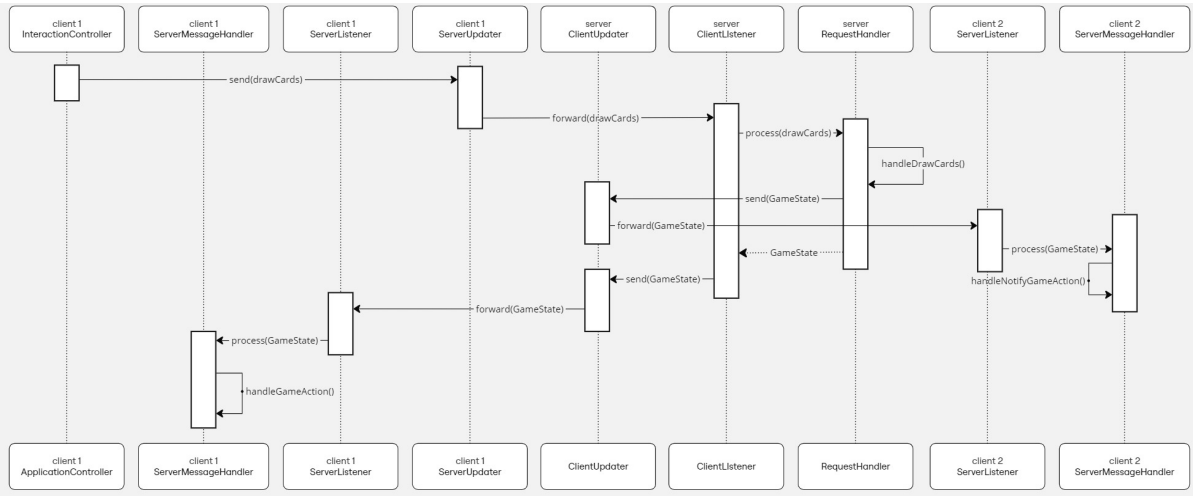

Figura 13: Diagramma delle attività rappresentante l'azione di pescare delle carte da parte di un client

L'`InteractionController` invia un messaggio nella coda, il `ServerUpdater` estrae il messaggio e lo serializza per inviarlo al server. Nel `ClientListener` viene ricevuto il messaggio e inviato al `RequestHandler` che elabora la richiesta eseguendo l'azione, poi invia tramite notifica all'altro client in gioco (client 2) il nuovo stato del gioco, passando per il `ClientUpdater`. Successivamente viene restituito al `ClientListener` l'esito della richiesta tramite risposta, questa viene inviata al `ClientUpdater` che la invierà al client. Il client riceve la risposta nel `ServerListener`, che viene infine processata dal `ServerMessageHandler`.

3 Dettagli Implementativi

Il sistema realizzato è stato strutturato in diversi package, la sua struttura è formata da:

- **client**
- **server**
- **message**
- **serialization**
- **test**

Ogni package citato è a sua volta composto da ulteriori package per suddividere il contenuto in moduli chiari e distinti per argomento.
Il sistema è stato documentato tramite **javadoc**, consultabile al link:
`https://github.com/SerafinoPandolfini/Cat-Lady-Javadoc`.

3.1 Client

Il client, come precedentemente descritto nella parte di design, è stato realizzato seguendo il pattern **MVC**. All'interno di questo package si trova l'interfaccia grafica del sistema, la parte di connessione con il server e i controller che permettono la comunicazione tra la parte di connessione e la GUI.

3.2 Server

Questo package contiene, oltre alla parte di connessione con i client, anche l'intera logica del gioco e delle lobby, costituendo il **model** del sistema. Il sistema infatti condivide con i client solo una copia delle informazioni sullo stato delle lobby e delle partite, mentre il server mantiene lo stato di verità del sistema bloccando eventuali operazioni illecite.

3.3 Message

Nel modulo message sono contenute le varie tipologie di messaggio che possono essere inviate all'interno del sistema. Ogni tipologia di messaggio, sia questo richiesta, risposta o notifica, è associato tramite `TypeToken` al tipo di contenuto che deve contenere.

3.4 Serialization

Questo package contiene le varie classi che svolgono il ruolo di serializzare e deserializzare il contenuto dei messaggi e le classi di utilità per semplificare questi processi.

4 Suddivisione del lavoro

Lavorando in un team di due persone si è deciso di adottare come approccio **Git Flow**. Questa scelta è stata effettuata data la precedente esperienza con questo approccio da parte dei membri del gruppo.

Per la suddivisione del lavoro si è inizialmente definito uno schema per suddividere gli aspetti principali del progetto tra i membri, ridefinito periodicamente sulla base del quantitativo di lavoro effettivo necessario per le singole parti.

Al termine del progetto il lavoro è stato svolto nel seguente modo:

Funzionalità	Sviluppatore
Architettura Server	Pandolfini
Architettura Client	Leonardi
Lobby	Leonardi
Inizio del gioco	Pandolfini
Assegnamento risorse	Pandolfini
Fine della partita	Leonardi, Pandolfini
Messaggi	Leonardi, Pandolfini

Tabella 2: Suddivisione del lavoro per macro-funzionalità

Alcune parti del lavoro, come indicato dalla tabella 2, sono state realizzate da entrambi gli sviluppatori in **pair programming**. Questa scelta è stata fatta o per la complessità della logica o per la sua importanza per la realizzazione di funzionalità successive. Per la parte di serializzazione e deserializzazione e per la realizzazione dell'interfaccia grafica ognuno si è occupato delle sezioni di sua competenza sulla base della distribuzione del lavoro.

5 Validazione

Per la validazione del software realizzato sono stati utilizzati sia test automatici che test manuali. Per i test automatici sono stati utilizzati i framework **JUnit** [3] e **Mockito** [5] tramite i quali sono stati verificati i vari aspetti della logica del gioco, delle lobby e del funzionamento della connessione. In particolare **Mockito** è stato utilizzato per effettuare dei test in grado di simulare le interazioni tra client e server nella lobby e in alcune delle fasi di gioco permettendo di sincronizzare il sistema tramite il mocking di alcune classi come `ServerMessageHandler`.

Nello specifico gli aspetti testati sono:

- Creazione, unione, abbandono di una lobby.
- Funzionamento complessivo della logica di gioco e di tutte le azioni relative come pescare, giocare carte, terminare il turno e punteggi.
- Serializzazione e deserializzazione delle classi per lo scambio di informazioni.
- Interazione tra client e server per le lobby e per la partitia.

Complessivamente sono stati realizzati 47 test.

Per verificare di aver coperto la maggior parte del codice con test adeguati abbiamo utilizzato **JaCoCo** come strumento di coverage. Al termine del progetto la coverage raggiunta sulle classi è del 72%, in figura 14 è presentata l'analisi della coverage per i singoli package testati.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.catlady.server.lobby.server		52%		40%	69	130	140	311	31	83	0	3
org.catlady.server		47%		50%	25	54	144	264	9	29	1	6
org.catlady.client.controller		22%		12%	71	80	100	133	43	52	2	3
org.catlady.server.game.manager		53%		26%	63	104	72	152	25	62	0	3
org.catlady.client.connection		65%		51%	34	71	92	251	14	37	2	5
org.catlady.server.game		82%		55%	49	149	31	267	18	96	1	10
org.catlady.server.game.card		68%		63%	32	81	32	112	10	37	2	9
org.catlady.serialization.game		83%		66%	6	37	16	95	2	25	0	16
org.catlady.server.game.matrix		78%		47%	13	34	11	67	2	17	0	2
org.catlady.message.response		79%		0%	9	31	7	45	2	24	0	15
org.catlady.serialization.message		71%		54%	5	16	13	45	0	10	0	6
org.catlady.utils.exceptions.game		6%		n/a	13	14	26	28	13	14	6	7
org.catlady.message.notification		79%		0%	8	26	6	30	3	21	0	14
org.catlady.utils		75%		46%	18	28	16	49	3	12	0	4
org.catlady.message.request		80%		0%	7	25	5	28	2	20	0	14
org.catlady.serialization.lobby		79%		50%	2	12	10	47	0	10	0	5
org.catlady.utils.exceptions.lobby.player		8%		n/a	9	10	18	20	9	10	4	5
org.catlady.utils.exceptions.lobby		8%		n/a	9	10	18	20	9	10	4	5
org.catlady.client		0%		0%	3	3	11	11	2	2	1	1
org.catlady.client.model		92%		n/a	2	23	4	52	2	23	0	1
org.catlady.serialization		95%		70%	8	24	4	70	2	14	0	6
org.catlady.server.lobby.client		89%		100%	2	17	4	37	2	16	0	3
org.catlady.server.lobby		100%		n/a	0	12	0	24	0	12	0	4
Total	3.558 of 10.028	64%	377 of 669	43%	457	991	780	2.158	203	636	23	147

Figura 14: Report coverage per i singoli package.

Lavorando in gruppo, per verificare che non ci fossero problemi di interazione tra il codice sviluppato dai membri, si è deciso di adottare come pratica di **Continuous Integration** una semplice pipeline realizzata tramite *GitLab CI/CD* che verificasse ad ogni commit il funzionamento dei test realizzati fino a quel momento.

Per valutare specifiche situazioni come crash da parte dei client o del server e per il funzionamento della GUI sono stati effettuati test manuali.

6 Istruzioni per il Deployment

Il progetto realizzato utilizza **Java** e **Maven** [4], pertanto è sufficiente che il dispositivo su cui verrà installato disponga di una JVM (versione 17 o superiore) e vi sia installato Maven (versione 3.6.x o superiore).

Successivamente è necessario collocarsi con un terminale all'interno della repository ed eseguire il comando:

- `mvn clean install`

In seguito sarà possibile eseguire il server e il client rispettivamente tramite i seguenti comandi:

- `mvn exec:java -Prun-server`

- `mvn exec:java -Prun-client`

Questi comandi permettono l'esecuzione del server e del client sulla porta 4444. Nel caso si voglia utilizzare una porta differente è possibile utilizzare i seguenti comandi:

- `mvn exec:java -P run-server -Dport=`*port*

- `mvn exec:java -P run-client -Dport=`*port*

In questo caso sarà necessario specificare al posto di *port* la porta che si desidera utilizzare.

7 Esempi d'Uso

Dopo aver avviato un'istanza del **server**, sarà possibile eseguire uno o più **client**.

Dopo l'avvio del client, se la connessione ha avuto successo, sarà possibile creare lobby o unirsi a lobby esistenti.

Il caso della creazione di una lobby pubblica, mostrato in figura 15, porta alla schermata principale della lobby, mostrata in figura 16. Per creare una lobby è necessario inserire nome della lobby, nome del giocatore e specificare se è pubblica o privata tramite radio-button.

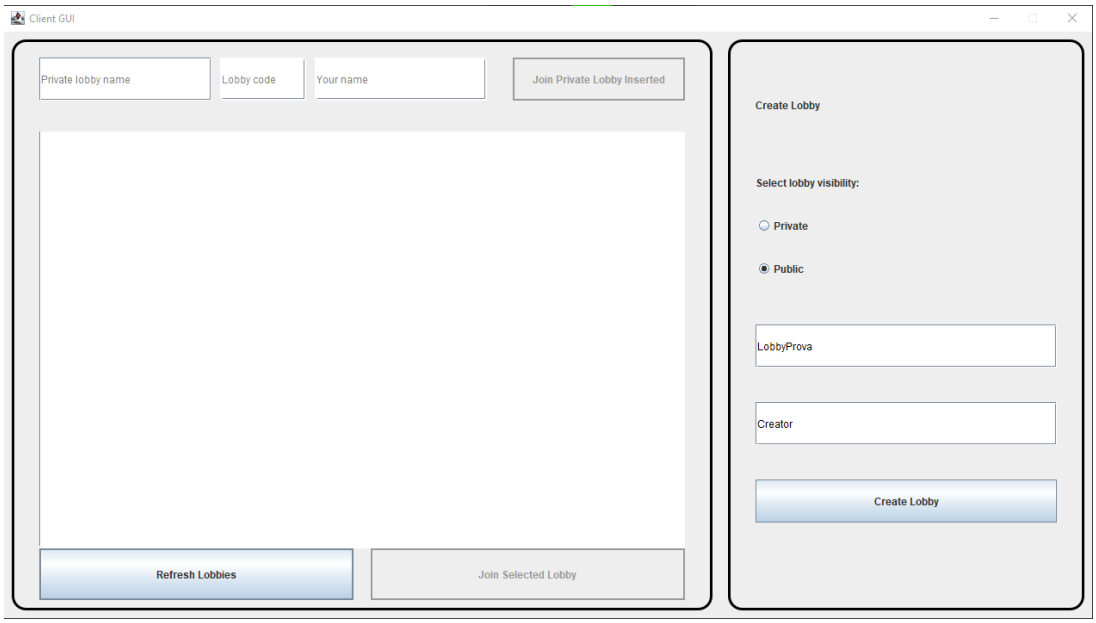

Figura 15: Creazione di una lobby pubblica

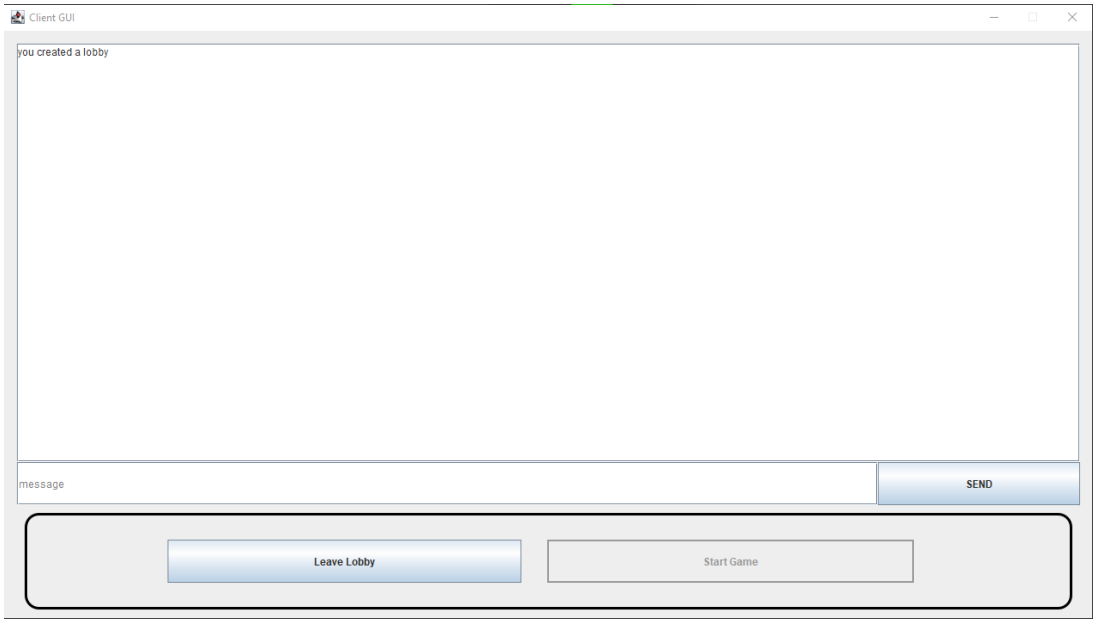

Figura 16: Lobby pubblica creata con successo

Le lobby pubbliche a cui è possibile accedere sono visualizzabili tramite il pulsante refresh, ed è possibile unirsi ad esse come mostrato in figura 17.

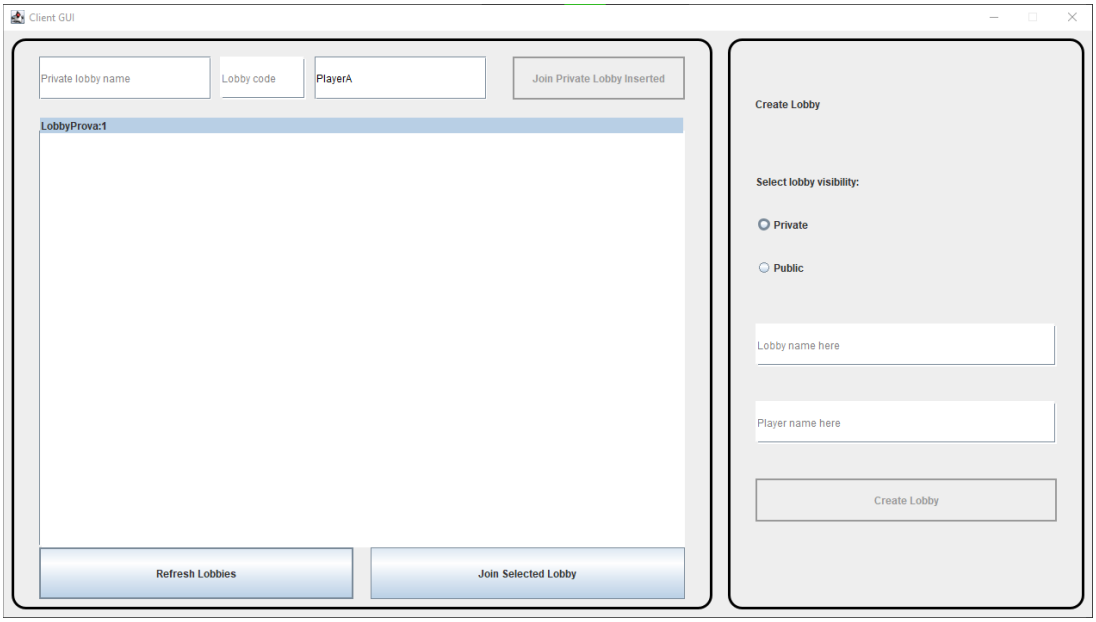

Figura 17: Unione ad una lobby pubblica esistente

Il caso della creazione di una lobby privata, mostrato in figura 18, porta alla schermata

principale della lobby mostrata in figura 19.

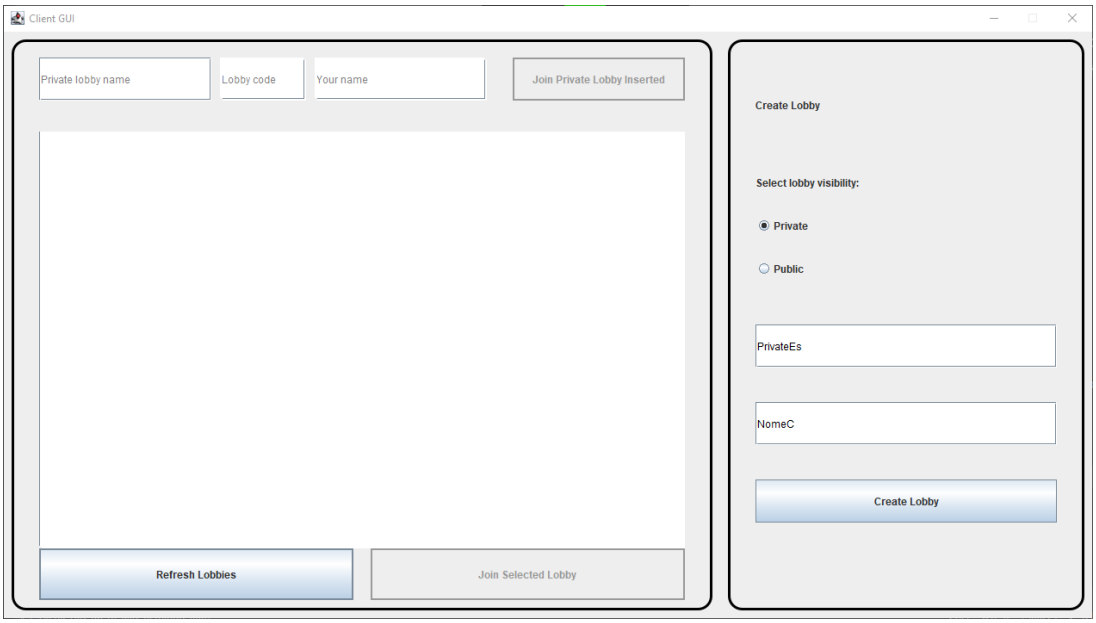

Figura 18: Creazione di una lobby privata

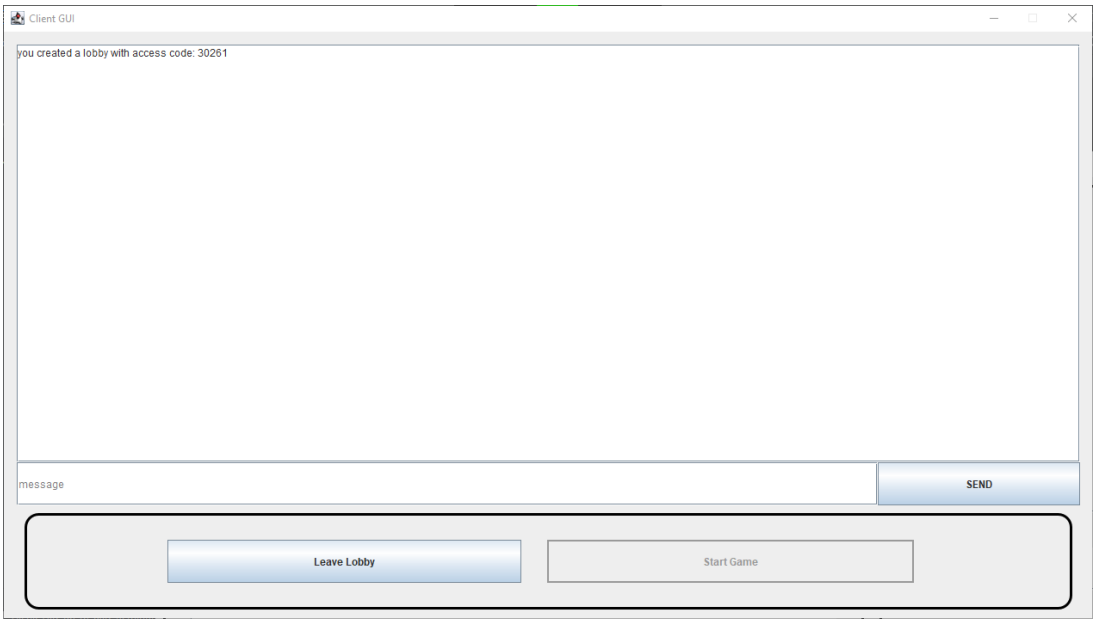

Figura 19: Lobby privata creata con successo

Per accedere ad una lobby privata è possibile compilare il form con i dati necessari,

come mostrato in figura 20. I dati richiesti sono nome della lobby, codice di accesso e nome del giocatore.

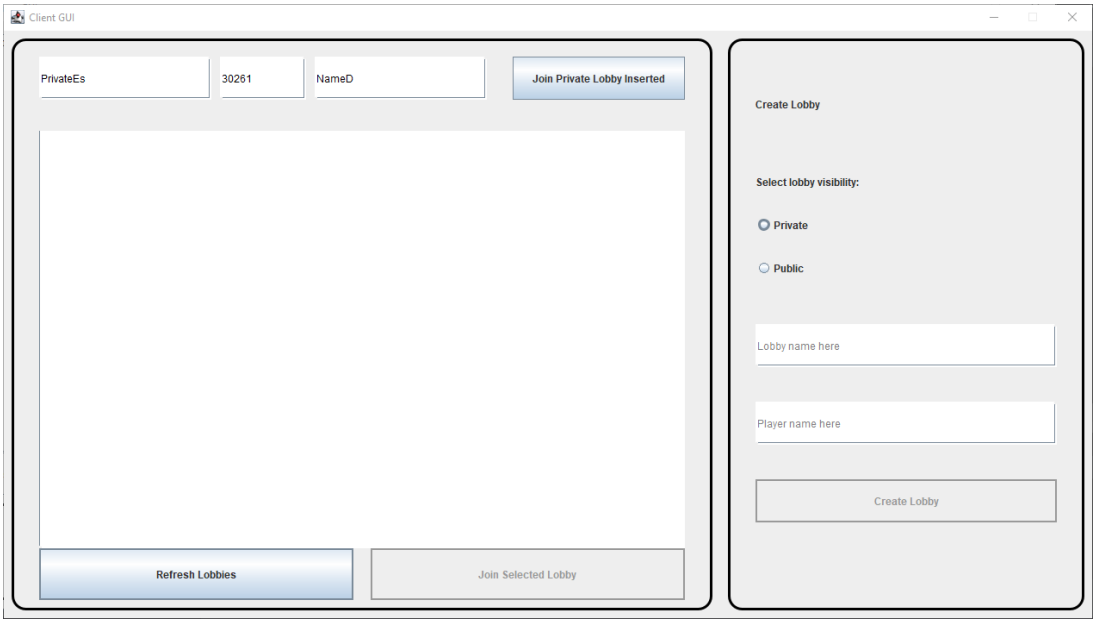

Figura 20: Unione ad una lobby privata esistente

All'interno della lobby sarà possibile inviare messaggi, come mostrato in figura 21.

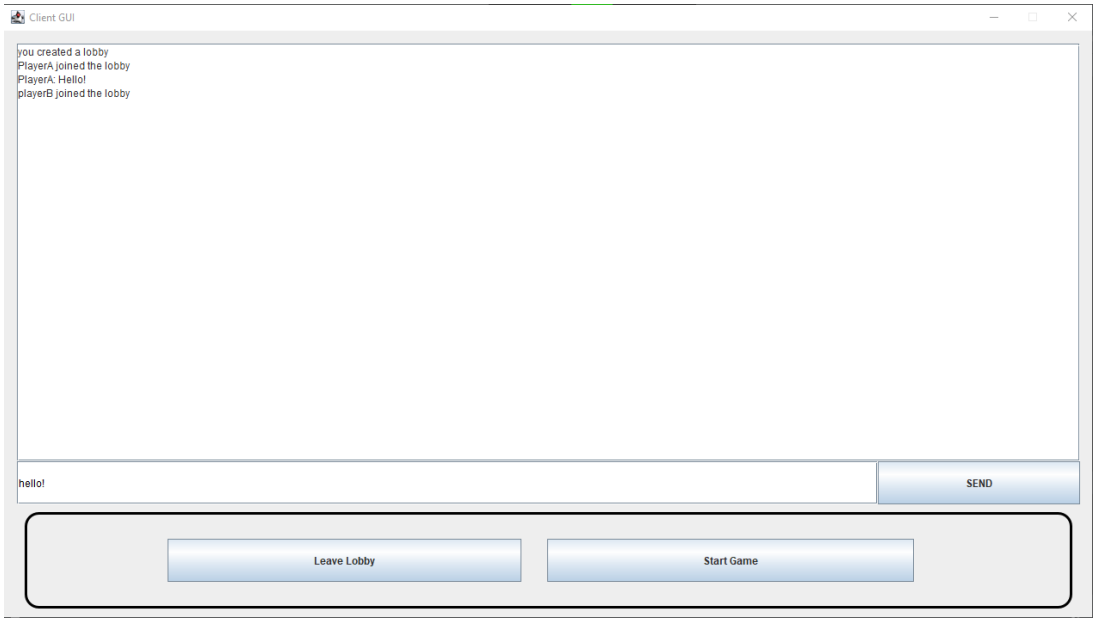

Figura 21: Invio di messaggi nella lobby

Nel caso avvenga un errore di connessione apparirà un messaggio di errore simile a quello mostrato in figura 22:

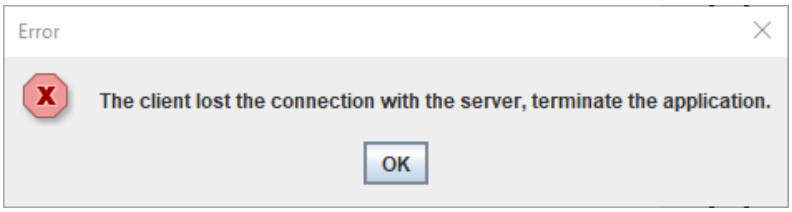

Figura 22: Messaggio di errore

Quando il numero di giocatori nella lobby raggiungerà il minimo consentito, il creatore della lobby potrà avviare la partita, e apparirà una schermata simile a quella mostrata in figura 23. In tale schermata sono visibili:

- la matrice di gioco
- le risorse disponibili
- il numero di carte restanti nel mazzo
- il nome del giocatore di turno
- gli stray cat che possono essere ottenuti
- il proprio mazzo di carte
- i gatti posseduti da ogni giocatore
- una finestra che mostra qualunque carta si selezioni o la chat della lobby
- le risorse possedute

Nella figura 23 si può notare che è stata selezionata una carta per osservarla meglio, inoltre si può notare che il giocatore è il primo a giocare poichè il token gatto non è ancora presente, infatti i pulsanti con le zampe rappresentanti la colonna o riga della matrice di carte da pescare sono tutti selezionabili. Si può inoltre osservare che ogni giocatore vede la sua board sempre nella posizione in basso a sinistra con le label evidenziate in rosso e il nome seguito dalla particella [YOU].

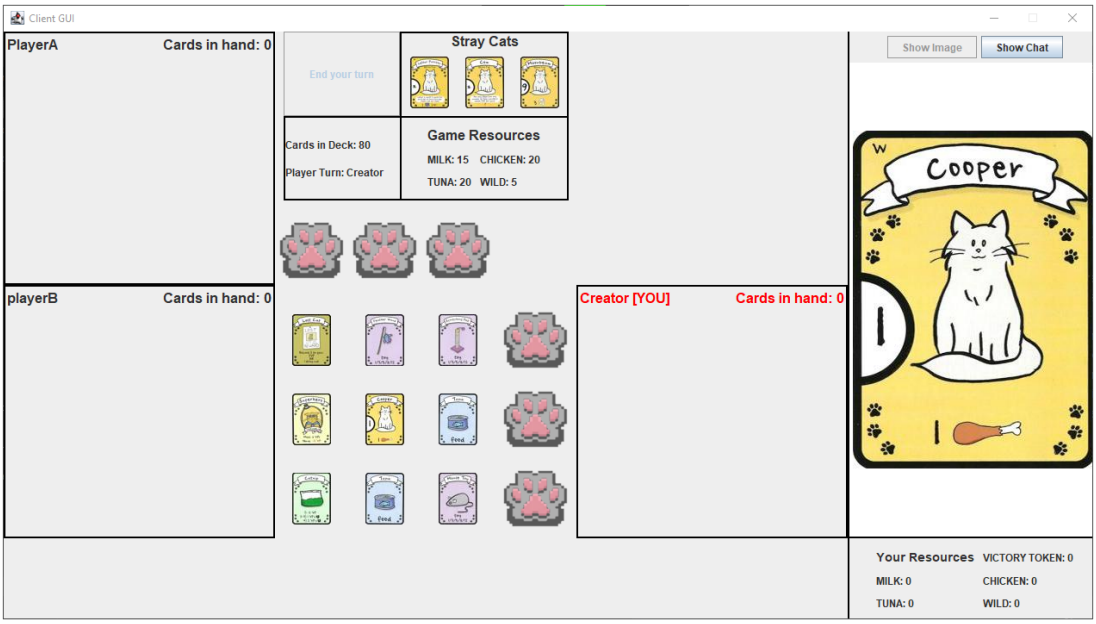

Figura 23: Schermata iniziale della partita

Nella figura 24 è possibile osservare la presenza del cat token, che rende la colonna non selezionabile fino al suo prossimo spostamento. Inoltre si può osservare che dove nell'immagine 23 era visibile il dettaglio di una carta, ora è visibile la chat della lobby, insieme al pulsante per lasciarla. Il cambio di visuale è ottenibile tramite i pulsanti show chat e show image. Si può inoltre osservare che pescando il giocatore ha acquisito delle carte nella mano e nelle risorse, ed avendo pescato può terminare il turno. Nella chat si può notare come ogni giocatore visualizzi nei propri messaggi la particella [YOU] che segue il nome.

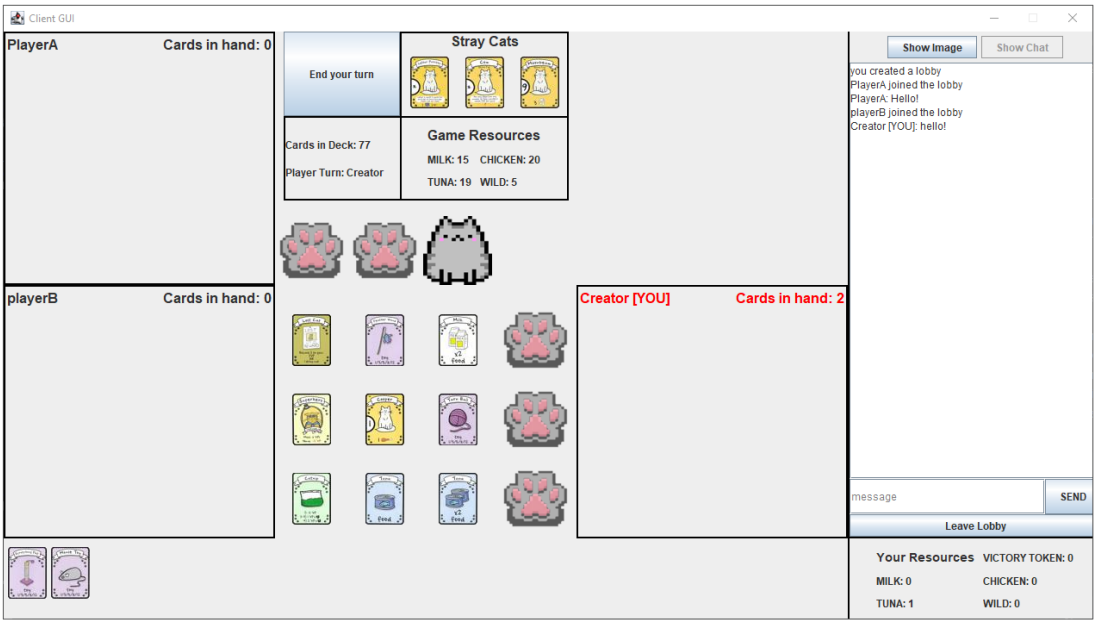

Figura 24:

Nell'immagine 25 si può vedere uno stadio avanzato della partita, in cui i giocatori hanno pescato carte gatto. Si può notare come, selezionando la spray bottle, se il giocatore la possiede la può utilizzare spostando il cat token dove preferisce

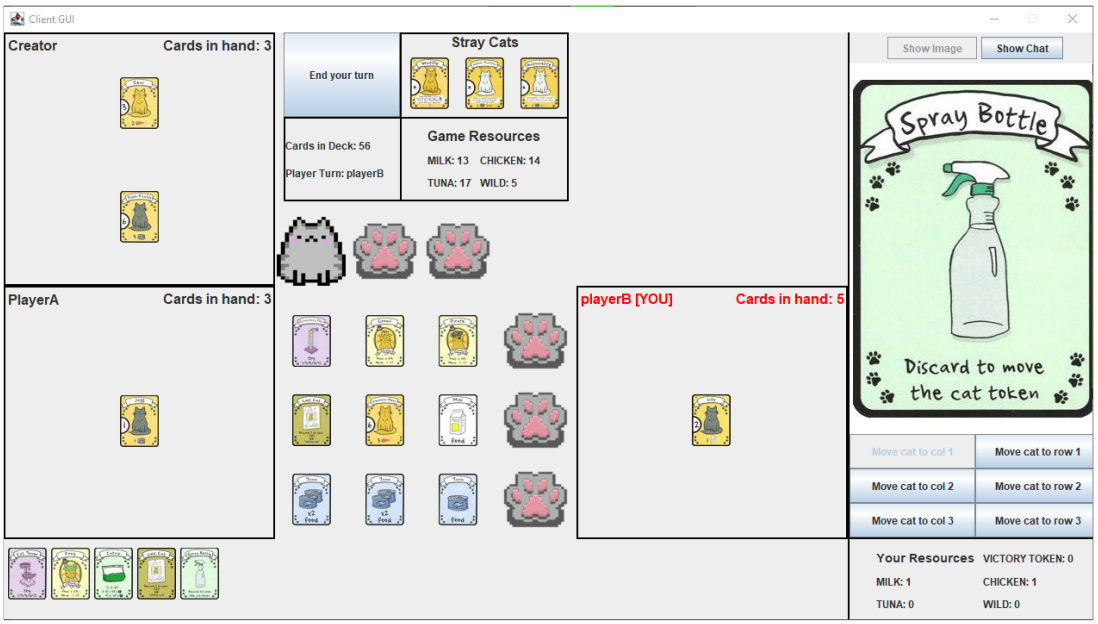

Figura 25: Dettaglio spray bottle

Nella figura 26 si può notare come il giocatore in possesso di lost cat, selezionandone uno, può decidere di utilizzarlo scambiandolo con stray cat o cat token dal valore di 2 punti vittoria

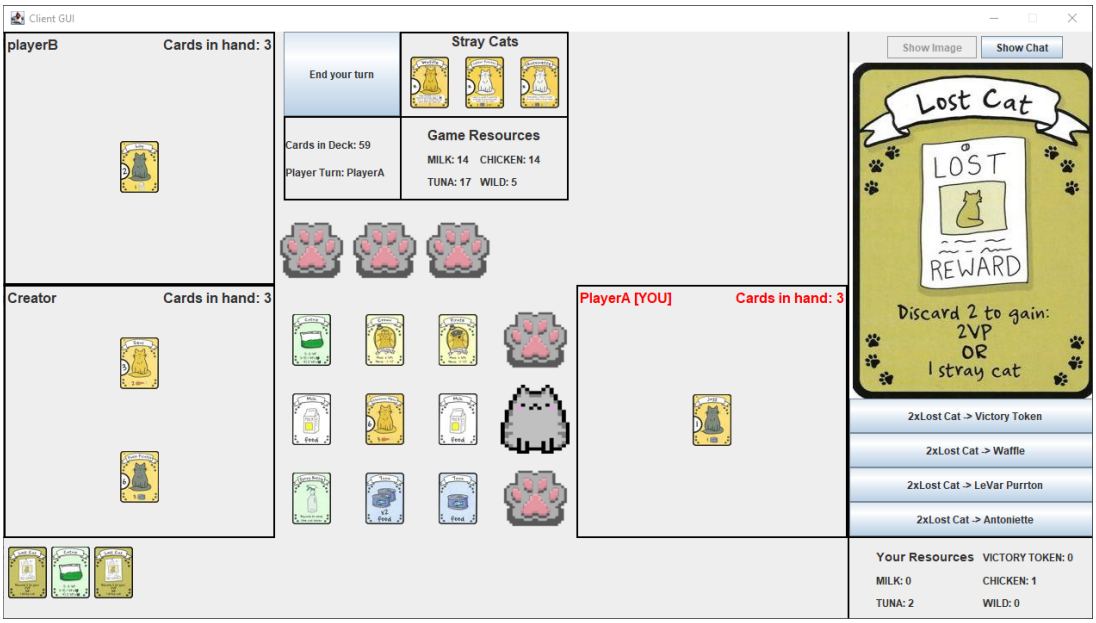

Figura 26: Dettaglio lost cat

Al termine delle carte nel mazzo di pesca sarà mostrata la schermata in cui si possono assegnare le risorse cibo ai gatti posseduti. Un esempio di tale schermata è la figura 27

Figura 27: Fase di assegnazione risorse

Quando tutti i giocatori avranno finito di assegnare il cibo, sarà mostrata la classifica, mostrata in figura 28, in cui sono evidenziati in giallo i punteggi massimi per ogni categoria, mentre nel conteggio del totale sono evidenziati primo, secondo e terzo classificato rispettivamente in giallo, grigio, e arancione. Inoltre è presente una label che riporta il nome del vincitore, o dei vincitori in caso di pareggio. A questo punto ogni giocatore può decidere se lasciare la lobby o tornarvi.

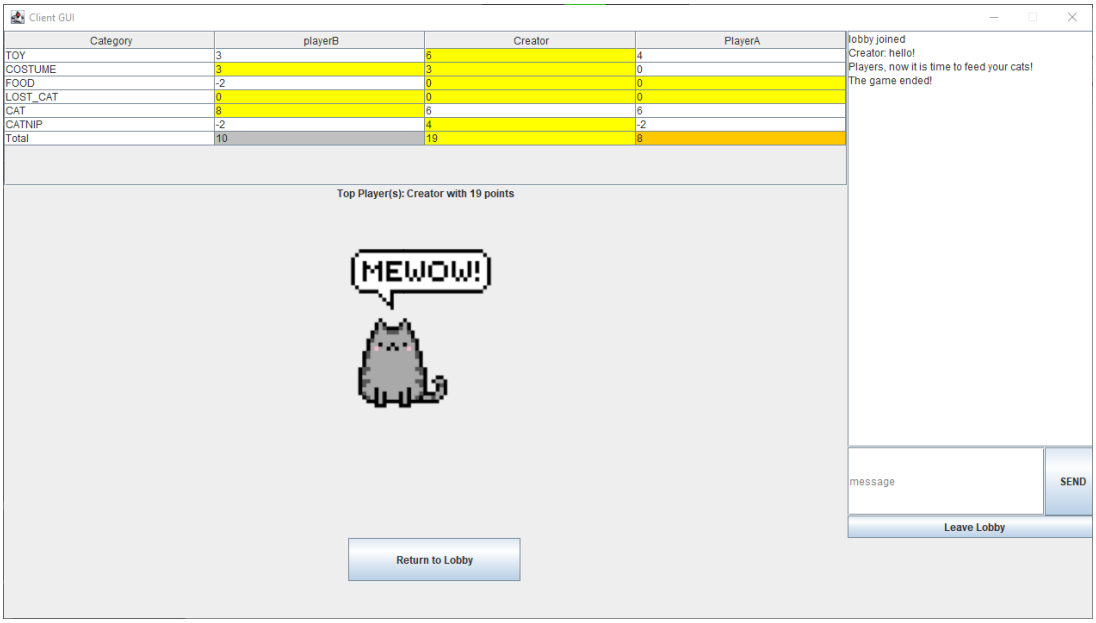

Figura 28: Classifica finale

8 Conclusioni

Il risultato del progetto è una versione completa e funzionante del gioco da tavolo **Cat Lady**. Rispetto ai requisiti posti all'inizio del progetto il risultato ottenuto è pienamente soddisfacente sia dal punto di vista delle funzionalità che del funzionamento del software realizzato.

8.1 Sviluppi Futuri

Considerando lo stato attuale del progetto, esistono alcuni miglioramenti e aggiunte che si potrebbero considerare per versioni future. Nello specifico:

- **Migliorare l'interfaccia grafica**: Attualmente l'interfaccia grafica, data la sua minore rilevanza all'interno delle finalità del corso, è basica e minimale; si limita ad offrire le funzionalità necessarie al funzionamento del sistema.
- **Account permanenti**: L'idea sarebbe quella di dare la possibilità ai giocatori di avere un proprio account tramite il quale autenticarsi all'applicazione. In questo modo ogni giocatore potrebbe mantenere lo storico delle partite giocate e dei punteggi realizzati.
- **Ampliamento del gioco base**: Il gioco possiede un espansione ufficiale che introduce nuove carte e nuove meccaniche, nonché la possibilità di fare partite fino a 6 giocatori. Si potrebbe offrire la possibilità di giocare partite sia includendo l'espansione che senza.

8.2 Cosa abbiamo imparato

Questo progetto ci ha permesso di apprendere nel dettaglio il funzionamento di un architettura client-server basata sui socket. Il fatto che il progetto sia stato realizzato in team ci ha permesso di sviluppare la capacità di organizzare e suddividere il lavoro in modo efficace per questa tipologia di architettura.

Riferimenti bibliografici

[1] Cat lady rulebook. https://www.alderac.com/wp-content/uploads/2017/08/cat-lady_rulebook.pdf.

[2] Gson. https://github.com/google/gson.

[3] Junit. https://junit.org/junit5/.

[4] Maven. https://maven.apache.org/.

[5] Mockito. https://site.mockito.org.