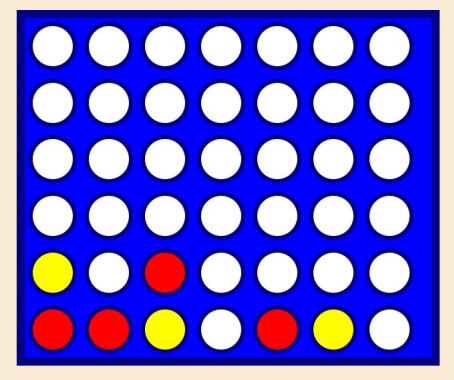
Connect 4



Martin Marcolini
martin.marcolini@studio.unibo.it

April 2023

Il progetto consiste nello sviluppo del gioco Connect 4, in versione multiplayer online.
Le partite saranno esclusivamente composte da due giocatori, in modalità 1vs1.
Il campo da gioco consiste in una griglia 6X7, nella quale gli utenti dovranno posizionare le proprie pedine.
I giocatori, a turno, dovranno scegliere la colonna nella quale posizionare la pedina e questa verrà collocata nella prima cella disponibile a partire dal basso.
Vince la partita l'utente che riuscirà a collegare quattro pedine dello stesso colore di seguito e questo potrà essere fatto in tre diversi modi: orizzontalmente, verticalmente e diagonalmente.

1 Goal/Requirements

Nella seguente sezione verranno riportate una serie di Q/A per spiegare più dettagliatamente i diversi aspetti del progetto, in particolare obbiettivi da raggiungere, requisiti e risulatati attesi.

1.1 Questions/Answers

- *Cosa succede una volta avviato l'applicativo?*
  - Una volta avviato l'applicativo verrà richesto all'utente di inserire un nickname (che non sia già utilizzato da un altro utente) che lo identificherà in modo univoco.
- *Una volta loggati cosa è possibile fare?*
  - L'utente può creare una nuova lobby oppure unirsi ad una lobby creata da un altro utente.
- *Come viene gestita la creazione delle lobby?*
  - Quando viene creata una nuova lobby di gioco, viene avviato un timer di 20 secondi entro il quale un altro utente si dovrà collegare. Al termine del timer, se nessun altro utente si sarà connesso, la lobby verrà eliminata e non sarà più visibile agli altri utenti.
- *Quanti utenti possono giocare contemporaneamente?*
  - Il sistema è stato progettato affichè possa supportare diverse partire in simultanea. Il numero massimo di utenti che possono partecipare ad una singola partita è 2.
- *Quali mosse può effettuare un utente?*
  - Ogni utente, durante il proprio turno, dovrà scegliere la colonna nella quale posizionare la pedina e questa verrà collocata nella prima cella libera a partire dal basso.
    Il turno di ogni utente ha una durata di 15 secondi e se al termine di quest'ultimo non è stata effettuata nessuna mossa, la pedina verrà collocata automaticamente dal sistema.
- *Cosa succede a fine partita?*
  - Una volta terminata la partita gli utenti possono decidere se effettuare un "rematch" oppure uscire dalla lobby e ritornare nella Home Page. Un utente può decidere in qualsiasi momento di abbandonare la partita.

1.2 Scenarios

Ipotizzando un possibile scenario di utilizzo di Connect 4, un utente per poter giocare dovrà connettersi al sistema inserendo un proprio nickname che non sia già stato utilizzato.
Successivamente un utente, una volta loggato, si troverà nella sua Home Page e potrà decidere se creare una nuova lobby oppure scegliere di unirsi ad una delle lobby già create.
Nel caso in cui l'utente decidesse di creare una nuova lobby, verrà avviato un timer di 20 secondi entro il quale un altro utente si dovrà collegare, altrimenti quest'ultima verrà eliminata e non sarà più visibile agli altri utenti.
Quando nella lobby saranno presenti due giocatori potrà aver inizio la partita.
L'utente che ha creato la lobby avrà il diritto ad effettuare la prima mossa.
Il turno di ogni utente ha una durata di 15 secondi, entro il quale l'utente dovrà decidere la colonna nella quale posizionare la propria pedina che verrà collocata nel prima cella libera a partire dal basso.
Se al termine del proprio turno, l'utente non ha effettuato nessuna mossa, la pedina verrà automaticamente collocata nella prima colonna disponibile a partire da sinistra in corrispondenza della prima cella libera a partire dal basso.
Vince la partita l'utente che per primo riesce a collegare quattro pedine di seguito, dello stesso colore, orrizzontalmente, verticalmente o diagonalmente.
Una volta terminta la partita gli utenti potranno decidere se voler effettuare un "rematch" oppure uscire dalla lobby e ritornare nella propria Home Page.

1.3 Self-assessment policy

Al fine di garantire il controllo della qualità e l'efficacia del progetto, sono stati realizzati dei test automatici sfruttando il framework Jest.
I test, hanno come obbiettivo verificare tutte le funzionalità principali del sistema, a partire da test più semplici come per esempio il controllo che due utenti non si loggino con lo stesso nickmane, fino alla realizzazione di test più complessi che simulano una vera e propria partita.

2 Requirements Analysis

Effettuando un attenta analisi dei requisiti è possibile identificare le principali funzionalità del sistema, le quali vengono descritte sfruttando i diagrammi dei casi d'uso.

2.1 Initial Logic

- Inserimento nickname
- Creazione nuova lobby
- Connessione ad una lobby esistente
- Abbandona Home Page

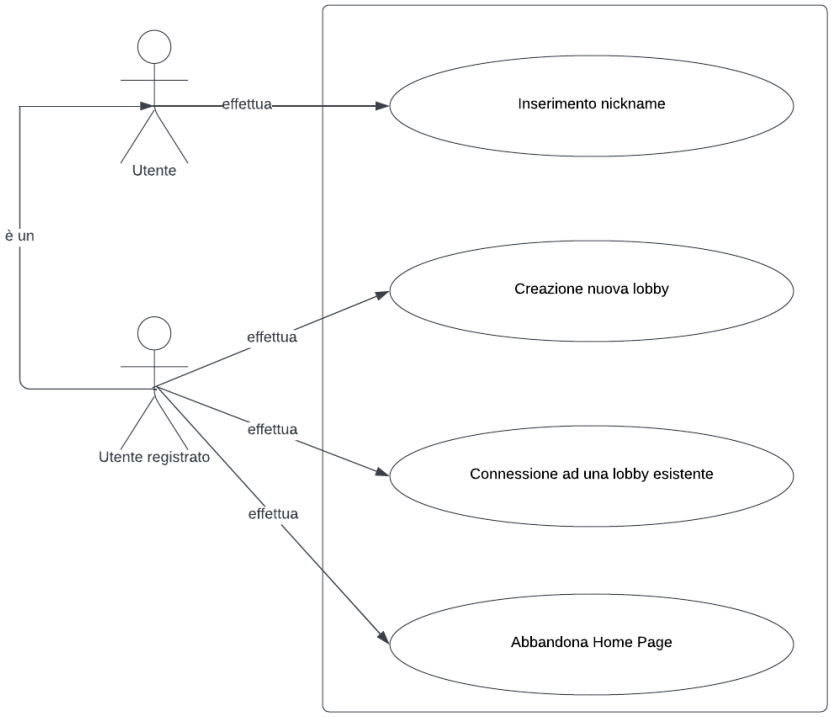


Figure 1: Initial logic Use Case

Quando si avvia la web Application verrà richiesto all'utente di inserire un nickname per poter effettuare il login. Successivamente, nella sua Home Page potrà creare una nuova lobby di gioco oppure connettersi a lobby create da altri utenti. In qualsiasi momento l'utente potrà decidere di abbandonare la Home Page e disconnetersi dal sistema.

2.2 Game Logic

- Posizionamento pedina
- Abbandona partita
- Richiesta rematch

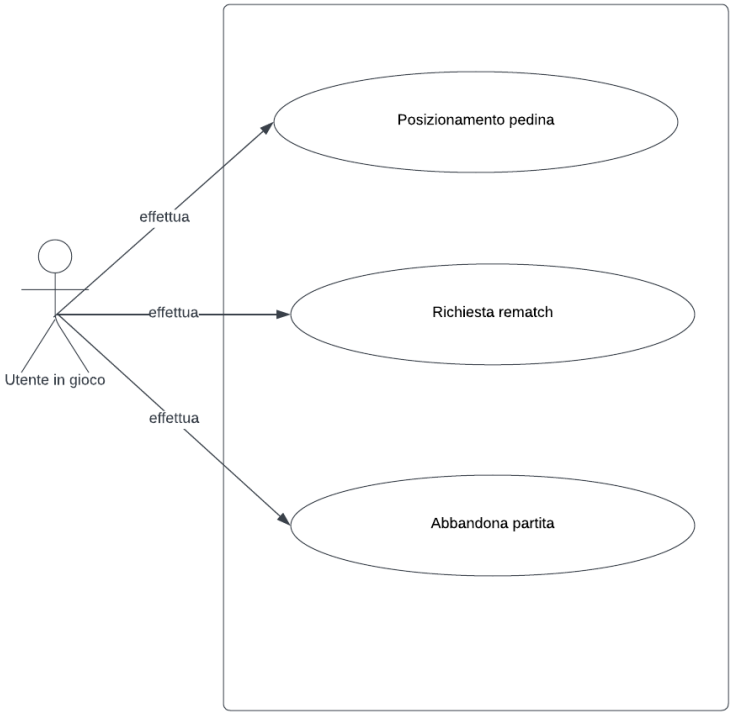


Figure 2: Game logic Use Case

All'avvio della partita l'utente che ha creato la lobby avrà il diritto ad effettuare la prima mossa. A turno ogni utente potrà decidere in quale colonna posizionare la propria pedina. Al termine della partita l'utente potrà richiedere di effettuare un "rematch". In qualsiasi momento l'utente può abbandonare la partita.
Sulla base dell'analisi effettuata è stato deciso di implementare il sistema utilizzando un'architettura client-server, implementando un Web Server sulla base del protocollo di comunicazione real-time WebSocekt.

3 Design

In seguito all'analisi dei requisiti e alla spiegazione dei casi d'uso possibili, si procede con la descrizione del design adottato per la realizzazione di Connect 4 online.

3.1 Structure

Nella figura 3 viene riportato il diagramma della classi che illustra le entità principali del model.
Per ogni entità vengono riportati i suoi attributi e i metodi principali, tra cui solamente alcuni getter/setter per non appesantire troppo lo schema.

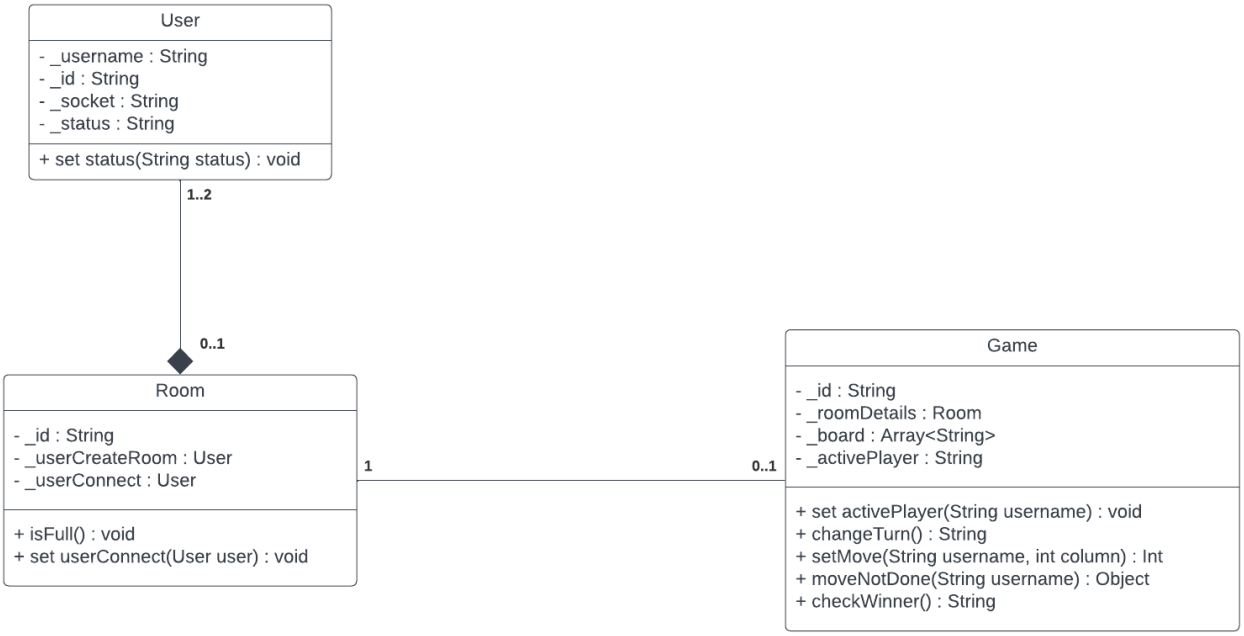


Figure 3: Class Diagram

L'entità **User** rappresenta il concetto di "utente" che si connette al sistema e gioca a Connect 4.
Questa entità è caratterizzata dai seguenti attributi:

- uername: è il nickname dell'utente
- id: è l'identificatore che viene assegnato all'utente
- socket: è l'istanza di connessione dell'utente e viene utilizzata dal server per inviare a lui stesso dei messaggi

- status: rappresenta lo stato dell'utente che può essere di tre tipi differenti
  - logged: l'utente si è loggato e si trova nella sua Home Page
  - room: l'utente ha creato una nuova stanza ed è in attesa che un altro giocatore si connetta
  - game: l'utente sta giocando ad una partita

Inoltre tra i metodi getter/setter dell'entità User, ha un importanza rilevante il metodo **set status(String status)** il quale si occupa di aggiornare lo stato corrente dell'utente.

L'entità **Room** rappresenta il concetto di "stanza" alla quale si connettono al più due utenti per effettuare una partita.
Questa entità è caratterizzata dai seguenti attributi:

- id: è l'identificatore che viene assegnato alla stanza
- userCreateRoom: è un riferimento a User e rappresenta l'utente che ha creato la stanza
- userConnect: è un riferimnto a User e rappresenta l'utente che si è connesso alla stanza creata precedentemente

Questa entità possiede tutti i metodi necessari per la gestione delle stanze:

- isFull(): verifica se la room è piena, ovvero controlla se sono connessi al più due utenti.
- set userConnect(User user): viene utilizzato per settare l'utente che si è collegato alla stanza.

L'entità **Game** rappresenta il concetto di "partita" ed è caratterizzata dai seguenti attributi:

- id: è l'identificatore che viene assegnato alla partita
- roomDetails: è un riferimento a Room e viene utilizzato per poter ricavare le informazioni necessarie sugli utenti che si sono connessi alla stanza di gioco e desiderano avviare la partita.
- board: rappresenta il campo di gioco
- activePlayer: è una stringa che rappresente il nome dell'utente "attivo", ovvero colui che ha il diritto ad effettuare la mossa.

Questa entità possiede tutti i metodi necessari per la progessione del gioco:

- set activePlayer(String username): viene utilizzato per impostare l'utente che ha il diritto ad effettuare la mossa.
- changeTurn(): viene utilizzato per imvertire il turno dei giocatori

- setMove(String username, int column): viene utilizzato per settare la mossa dell'utente. Riceve come parametro l'username dell'utente che ha effettuato la mossa e la colonna della griglia selezionata dall'utente dove posizionare la pedina. Il metodo si occupa di controllare se la colonna selezionata sia libera e successivamente posiziona la pedina in corrispondenza della prima cella libera a partire dal basso.
- moveNotDone(String username): viene utilizzato per posizionare la pedina dell'utente in modo automatico, qualora lui stesso non effettuasse la propria mossa prima dello scadere del timer.
- checkWinner(): viene utilizzato per controllare se un utente ha vinto la partita.

3.2 Behaviour

Nella figura 4 viene riportato il diagramma degli stati che descrive il comportamento della logica iniziale del sistema ovvero, dal momento in cui l'utente avvia l'applicazione fino all'inizio della partita.

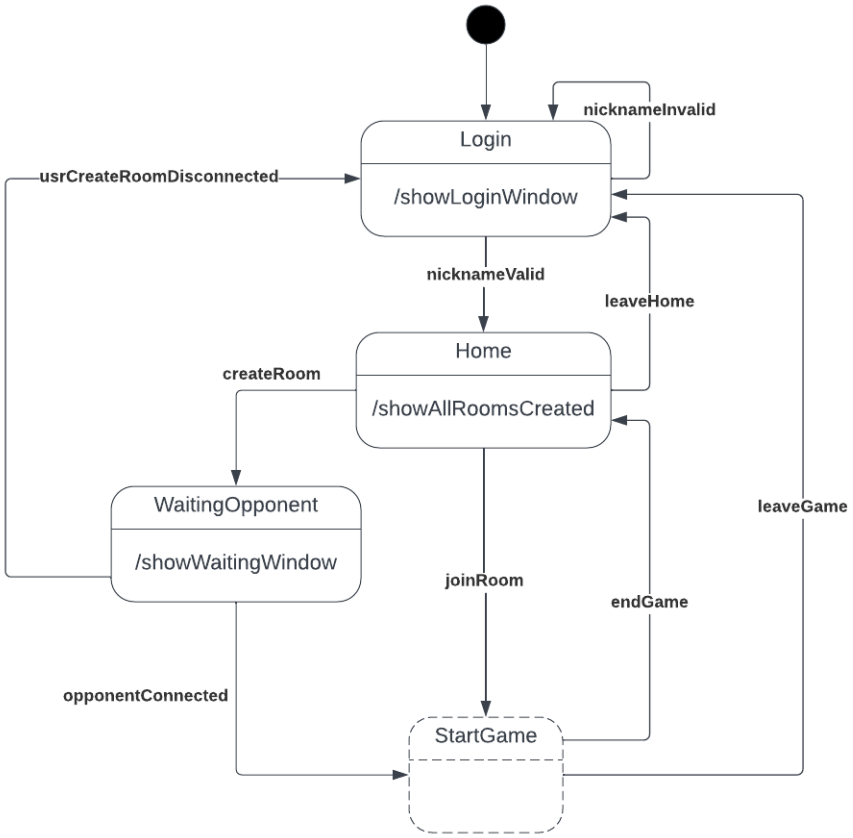


Figure 4: State Diagram - Initial Logic

All'avvio dell'applicazione, viene mostrata all'utente la schermata di login dove viene richiesto di inserire un nickname che non sia già in uso. Successivamente viene mostrata l'Home Page dell'app, dove l'utente potrà visualizzare le stanze già create da altri utenti ed unirsi, oppure crearne lui stesso una nuova. Nel caso un utente decidesse di creare una nuova stanza, dovrà attendere la connessione di un altro giocatore e quando ciò avviene potrà aver inizio la partita. Nel caso invece l'utente volesse collegarsi ad una lobby (stanza) già esistente è sufficiente che ne selezioni una di quelle mostrate nella lista delle lobby create e poi la partita potrà aver inizio. In qualisiasi momento l'utente potrà decidere di scollegarsi dal sistema e verrà riportato alla schermata di Login.

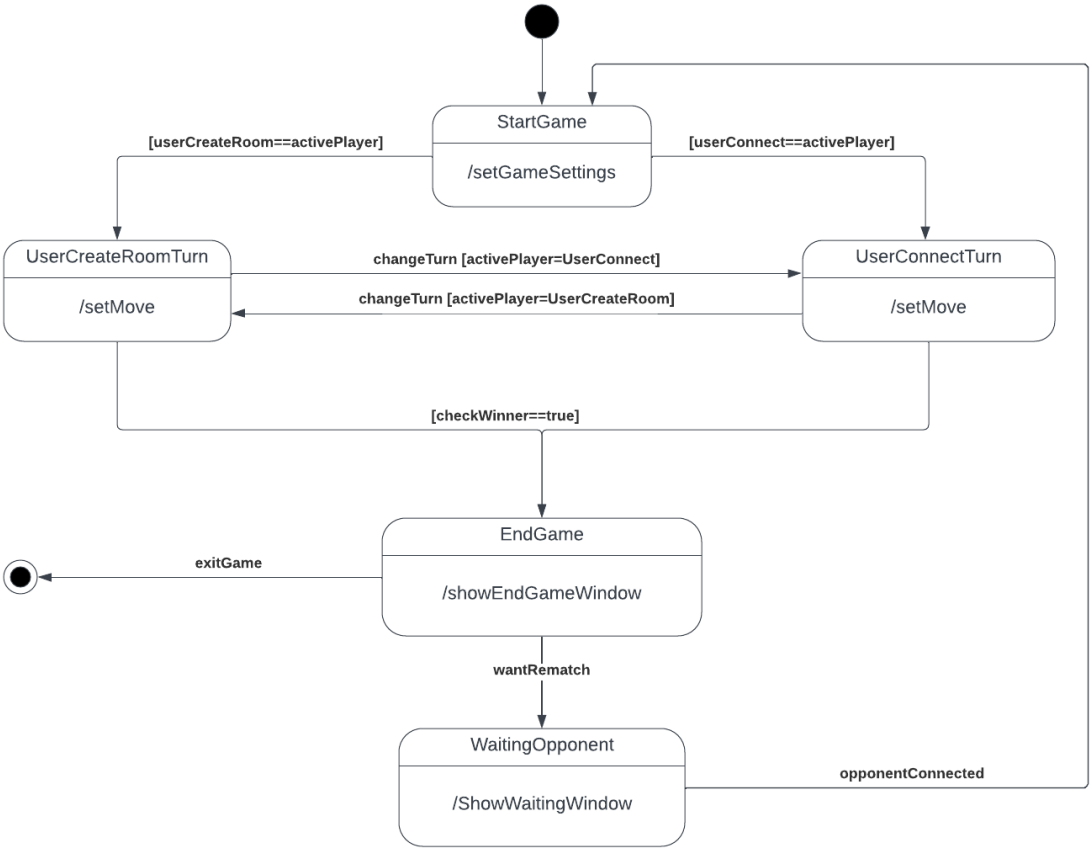


Figure 5: State Diagram - Game Logic

In figura 5 viene mostrato il diagramma degli stati che descrive il comportamento della logica di una partita. Quando si crea una nuova partita, l'utente che ha creato la stanza ha il diritto ad effettuare la prima mossa. Una volta selezionata la colonna nella quale posizionare la propria pedina, il turno passa all'altro giocatore (changeTurn). Ogni volta che un utente effettua la propria mossa, si controlla se quest'ultimo ha vinto la partita (checkWinner) ovvero se ha posizionato quattro pedine dello stesso colore verticalmente, orizzontalmente o diagonalmente. Una volta decretato il vincitore, gli utenti possono decidere se voler rigiocare la partita oppure uscire e ritornare alla propria Home Page. L'utente che desidera effettuare una rivincita attende che l'altro giocatore accetti la richiesta. Se l'avversario esce e ritorna nella sua Home Page, la partitia non viene avviata. Se l'altro giocatore accetta la richiesta di rivincita, allora viene avviata una nuova partita.

3.3 Interaction

In questa sezione verranno mostrati diversi Diagrammi di Sequenza per illustrare le principali interazioni del sistema.
La counicazione tra Client e Server si basa sullo scambio di messaggi nel formato JSON. Ogni risora presente nel sistema (User, Room e Game) possiede un Controller lato server, il quale viene sfruttato dal Service (server) per rispondere alle richieste dei client.

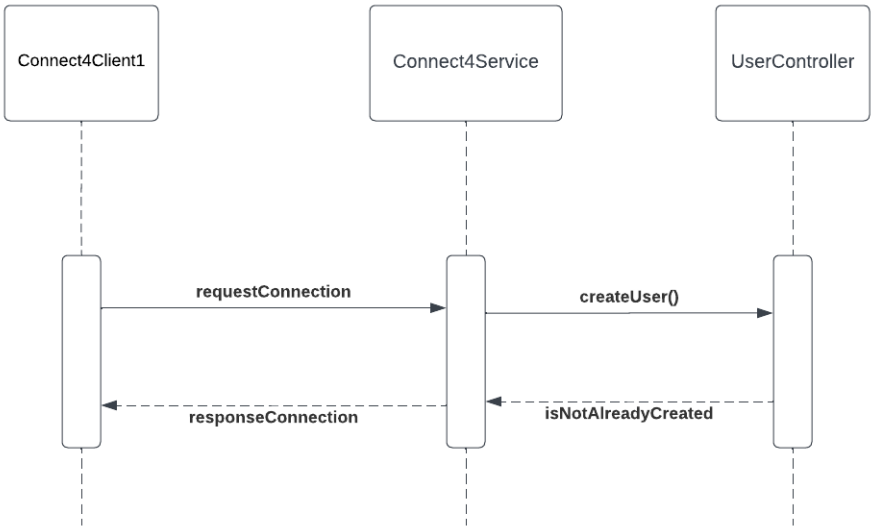


Figure 6: Sequence Diagram - Login logic

In figura 6 viene mostrato un diagramma di sequenza che esemplifica il login di un utente. Un **Client** per loggarsi al sistema invia una richiesta di connessione al server (Connect4Service), la quale poi viene indirizzata allo UserController che ha il compito di verificare se è non gia presente nel sistema un utente con lo stesso nickname. Le risposte alle varie richieste si propagano seguendo la stessa catena di interazioni tra i vari componenti del sistema. In questo caso il Controller comunica al server che non è presente nessun utente con lo stesso nickname e successivamente il server invia la risposta di "Connessione effettuata" al client.

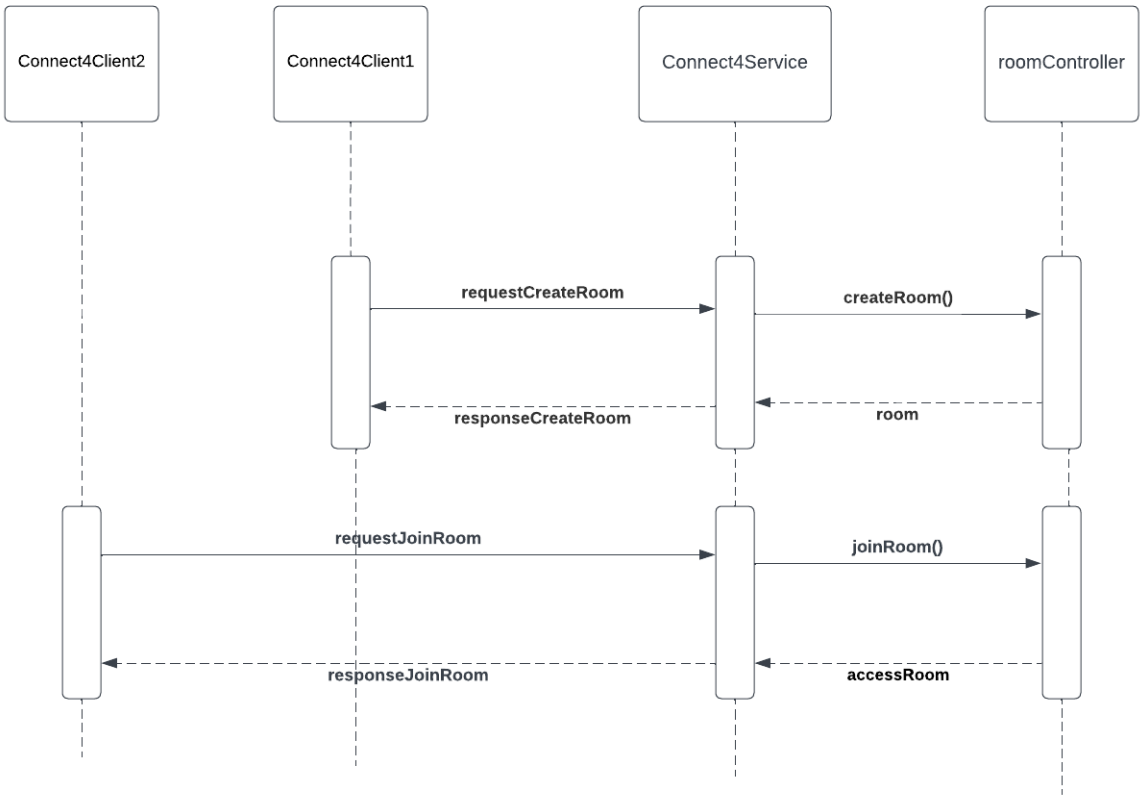


Figure 7: Sequence Diagram - Room logic

In figura 7 viene mostrata la sequenza di iterazioni che avvengono quando un client crea una lobby (room) ed un altro utente si connette alla stessa. Il **Client1** inizialmente effettua una richiesta di creazione di una nuova lobby al **Service**. Successivamente, tale richiesta, verrà indirizzata ial **Controller** il quale si occuperà di creare la nuova lobby. Il **Controller** comunica al **Service** la nuova stanza creata, il quale invierà la risposta alla richiesta client. In seguito il **Client2**, invierà una richiesta al **Service** di potersi connettere alla nuova stanza appena creata. Tale richiesta verrà indirizzata al **Controller**, il quale si occuperà di verificare se la lobby non è già piena. In questo caso il **Controller** comunicherà al **Service** che l'utente può connetersi alla stanza e quest'ultimo invia la risposta di "Connessione effettuata correttamente" al **Client2**.

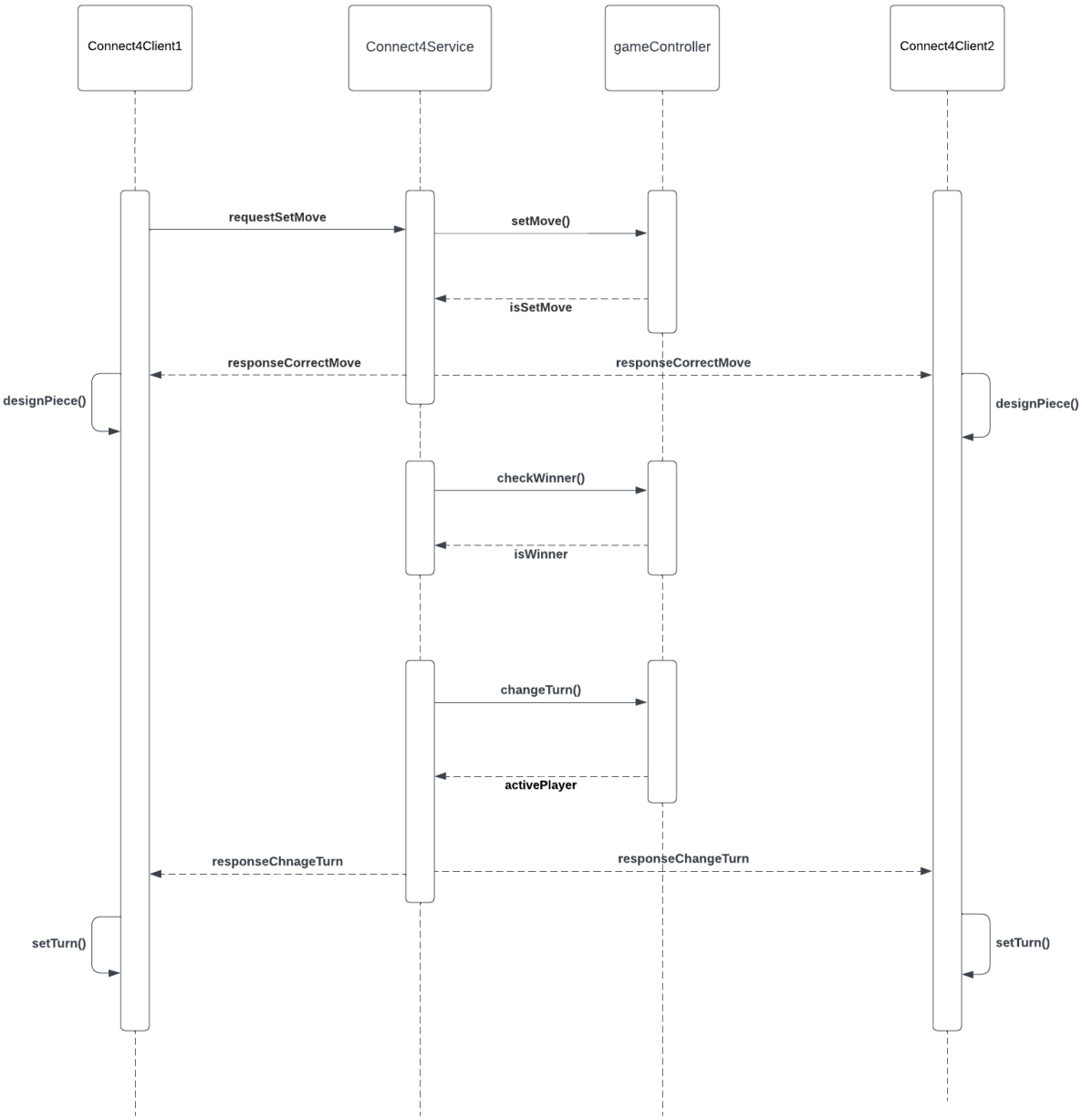


Figure 8: Sequence Diagram - Game logic

In figura 8 viene mostrata la sequenza di iterazioni che avvengono durante il flusso di una partita. Il **Client1** seleziona la colonna nella quale posizionare la propria pedina. La mossa effettuata viene inviata al **Service** (**requestSetMove**) e successivmanete indiriz-

zata al **Controller**, il quale si occuperà di verificare se la colonna seleziona dall'utente, dove posizionare la pedina, non è già piena. In questo caso la mossa effettuata è valida e verra inoltrata la risposta ad entrambi i client, i quali si occuperanno di far visualizzare la pedina (**designPiece()**) sul campo di gioco. Ogni volta che un client effettua la prorpia mossa, il **Service**, sfruttando le funzionalità del **Controller** verifica se quest'ultimo (**checkWinner()**) ha vinto la partita. In questo caso la mossa non è quella vincente per cui la partita prosegue.
Una volta effettuata la propria mossa il turno passa all'altro giocatore. Il **Service** richiede al Controller (**changeTurn()**) di impostare il nuovo utente attivo che avrà il diritto ad effettuare la prossima mossa. Il **Service** comunicherà (**responseChangeTurn**) ad entrambi i client qual'è il nuovo utente attivo e graficamente verrà riavviato il timer del turno ed evidenziato *l'activePlayer* (**setTurn()**).

4 Implementation Details

In questa sezione verranno riassunti i meccanismi e le strategie adottate nella fase di implementazione, citando brevemente le tecnologie utilizzate. Il sistema è composto in due parti principali **Connect4Client** e **Conncet4Server**.

4.1 Connect4Server

Il server è stato realizzando in Javascript come linguaggio di programmazione eseguito su **Node.js**[3].
Il protocollo di comunicazione scelto è **WebSocket**[4], la decisione di adottare quest'ultimo nasce dall'esigenza di voler garantire aggiornamenti automatici ai client connessi, quando un altro utente ha avuto un'interazione con il sistema. Le WebSocket forniscono una comunicazione bidirezionale in tempo reale tra client e server, e quindi sono estremamente utili nella realizzazione, come in questo, di giochi multiplayers online. WebSocket a differenza di HTTP, che è un protocollo unidirezionale, consente al server di inviare i dati al client anche quando quest'ultimo non ha effettuato una richiesta. La server socket sviluppata gestisce tutte le interazioni tra il client e il sistema: dal login, alla creazione e connessione ad una stanza di gioco fino all'intero flusso di una partita.
Quando un client si connette al server, l'istanza di connessione viene salvata in un set che tiene traccia degli utenti connessi. Quando il server deve inviare informazioni ai client sfrutta due metodi:

- *broadcast*: il server invia un messaggio JSON a tutti i client connessi. Questo meccanismo viene usato, per esempio, per comunicare a tutti i client connessi che un determinato utente ha creato una nuova stanza.
- *send*: il server invia un messaggio JSON ad uno specifico client sfuttando l'instanza di connessione che viene memorizzata nel set al momento del login dell'utente. Questo meccanismo viene utilizzato durante il flusso di una partita quando è necessario comunicare con i soli due utenti che stanno giocando.

Il WebSocket server viene creato nel file *app.js* ed è in ascolto alla porta 3001.
Le gestione degli eventi del server e della logica del sistema viene effettuata nel package *Server*, il quale è suddiviso in:

- *Models*: contiene le classi che rappresentano le risorse identificate nelle precedenti sezioni ovvero: *User*, *Room*, *Game*.
- *Managers*: sono dei sotto-moduli del server che si occupano di gestire le richieste, che riguardano una delle risorse definite nel Models, ed inviare le risposte ai client. Ogni risorsa ha un proprio manager: *userManager*, *roomManager*, *gameManager*.
- *Controllers*: contiene i controller associati ad ogni risorsa definita nel Models. I controller vengono sfruttati dai corrispondenti manager per verificare se le operazioni richieste dell'utente, su una determinata risorsa, possono essere soddisfatte. Ogni risorsa ha un proprio controller: *usersController*, *roomsController*, *gamesController*.

4.2 Connect4Client

Il Client è stato sviluppato in Javascript, html e css. In particolare è suddiviso in tre sotto-moduli principali:

- *Pages*: fornisce un'interfaccia grafica con la quale l'utente può interagire ed è composta da tre pagine html: *loginPage*, *homePage*, *gamePage*.
- *CSS*: contiene i fogli di stile associati ad ogni pagina html.
- *Controllers*: contiene i controller associati ad ogni pagina html, i quali hanno il compito di interagire direttamente con l'interfacia grafica ed intercettare i comandi dell' utente.
  Ogni controller implementa la socket client, basata sul protocollo WebSocket, necessaria per inviare e ricevere aggionamenti costanti dal server.

L'utilizzo dei WebSocket lato client è possibile attraverso l'uso di specifiche API che consentono:

- di ricevere informazioni sullo stato della connessione (connecting, open, closed)
- di inviare dati al server e chiudere la connessione attraverso metodi come: *socket.send()* e *socket.close()*
- di gestire una serie di eventi particolari come:
  - *onmessage*: evento che viene richiamato quando il client riceve dei messaggi dal server.
  - *onopen*: evento che viene richiamato quando si instaura la connessione con la socket del server.

– *onclose*: evento che viene richiamato quando lo stato della connessione passa a CLOSED.

– *onerror*: evento che viene richiamato quando si verificano degli errori nella connessione con la socket del server.

Lo scambio di informazioni tra le socket (client e server) si basa sull'invio e ricezione di messaggi nel formato **JSON**[2].

```
const payload={
    "method": "requestConnection",
    "clientUsername": username
}
```

Figure 9: JSON Message Example

In figura 9 viene mostrato un tipico esempio di messaggio JSON inviato dal client e server. La struttura dei messaggi JSON cambia a seconda delle informazioni che devono essere trasmesse. L'unico campo che è sempre presente in ogni messaggio è *method*, il quale specifica il tipo della richiesta che viene effettuata dal client. Può assumere tre valori differenti:

- *requestConnection*: richiesta di connessione al sistema.
- *room*: il client effettua delle richieste al server riguardanti la gestione delle stanze di gioco, per esempio la creazione di una nuova stanza.
- *game*: il client effettua delle richieste al server riguardanti la gestione del flusso di un partita, per esempio il posizionamento di una pedina.

I percorsi sviluppati che consentono all'utente di visualizzare l'interfaccia grafica sono:

- */*: path che mostra la pagina di login.
- */HomePage*: path che mostra la HomePage dell'utente, dove è possibile creare una nuova stanza e visualizzare quelle create da altri utenti ed entrarci.
- */GamePage*: path che mostra la pagina di gioco dove si svolge Connect4.

Il servizio di "frontend" è in ascolto alla porta 3000.

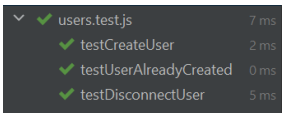
4.3 Ensure reliability

Al fine di evitare il blocco del sistema sono state adottate alcune precauzioni durante le seguenti fasi:

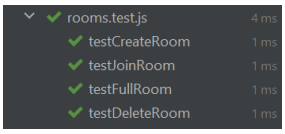
- Creazione stanza: quando un utente crea una nuova stanza di gioco, viene avviato un timer di 20 secondi entro il quale un altro utente si dovrà connettere. Una volta scaduto il timer, l'utente che aveva creato la stanza verrà riportato alla proria home page, e la stanza creata verrà eliminata e non sarà più visibile agli altri utenti. In questo modo l'utente che ha creato la stanza, non rimarrà bloccato sulla schermata di attesa qualora nessun altro utente si connetta.
- Partita:
  - Il turno di un utente ha una durata di 15 secondi, entro il quale quest'ultimo dovrà effettuare la propria mossa. Al termine del poprio turno, se l'utente non ha effettuato alcuna mossa, il sistema in automatico posizionerà la pedina sulla griglia e il turno passerà all'avversario. Impostando una durata massima del turno di un utente, si evita il blocco della partita qualora l'utente attivo non dovesse più interagire con il sistema.
  - Qualora un utente dovesse disconnetersi dalla partita, lato client verra inviato un messaggio a server il quale interromperà la partita e notificherà l'avversario dell'accaduto.
- Richiesta rematch: quando un utente effettua una richiesta di rematch, viene avviato un timer di 20 secondi entro il quale l'avversario potrà accettare la richiesta. Una volta scaduto il timer, l'utente potrà decidere se effettuare un altra richiesta oppure ritornare alla propria home page. In questo modo l'utente che ha effettuato la richiesta di rematch non rimarrà bloccato sulla schermata di attesa, qualora l'avversario dovesse uscire/disconnettersi oppure non interagire più con il sistema.

5 Self-assessment / Validation

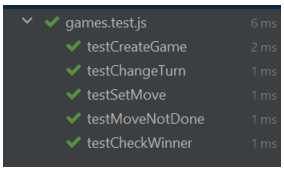
Il testing è stato realizzato tramite test automatici utilizzando il framework **Jest**[1].
I test sono presenti all'interno del progetto nel modulo chiamato **Test**.



(a) Jest - User Test



(b) Jest - Room Test



(c) Jest - Game Test

Sono stati realizzati tre file di test, ognuno riguaradante una specifica risorsa del sistema:

- User Test: contiene i test che simulano la fase di login (verifica se è gia presente un utente con il medesimo nickname e successiva creazione) e la disconnessione di un utente dal sistema.
- Room Test: contiene i test che riguardano la gestione della logica di creazione e connessione alle stanze di gioco.
- Game Test: contiene i test che simulano il flusso di una partita.

E' possibile avviare l'intera suite di test, da terminale, tramite il comando ***npm test***.

6 Deployment Instructions

Inizialmente è necessario scaricare tutte le dipendenze, posizionandosi nella direcotry del progetto con un terminale ed eseguendo il comando: ***npm install***.
A seguire per avviare il server è necessario eseguire il comando, sullo stesso terminale: ***npm start***.
Infine per simulare la partecipazione dei giocatori è necessario aprire una pagina web (una per ogni giocatore) e collegarsi al seguente indirizzo: ***http://localhost:3000/***

7 Usage Examples

Inizialmente verrà richiesto all'utente di loggarsi inserendo il proprio nickname. Nel in caso in cui venisse scelto un nickname già presente nel sistema, verrà visualizzato un messaggio di errore.

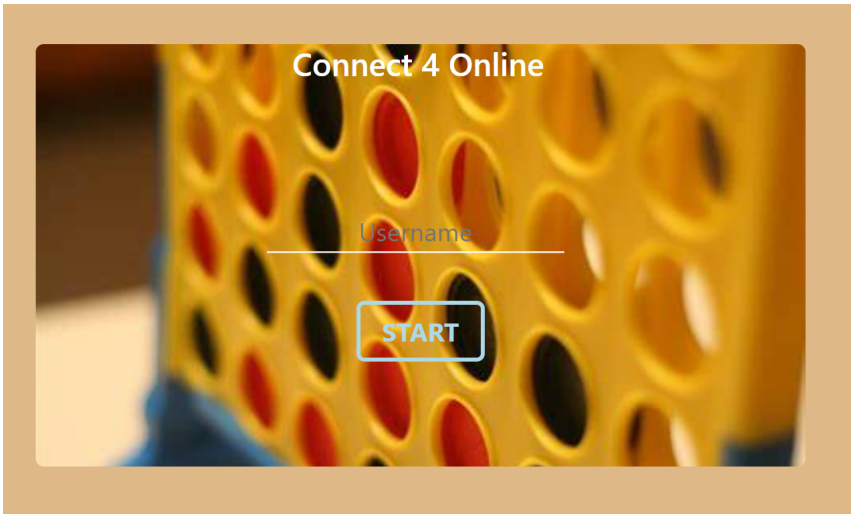


Figure 11: Login Page

Avvenuta la registrazione, l'utente si trovarà nella propria Home Page dove potrà creare una nuova stanza oppure connettersi ad una delle stanza create dagli altri utenti.

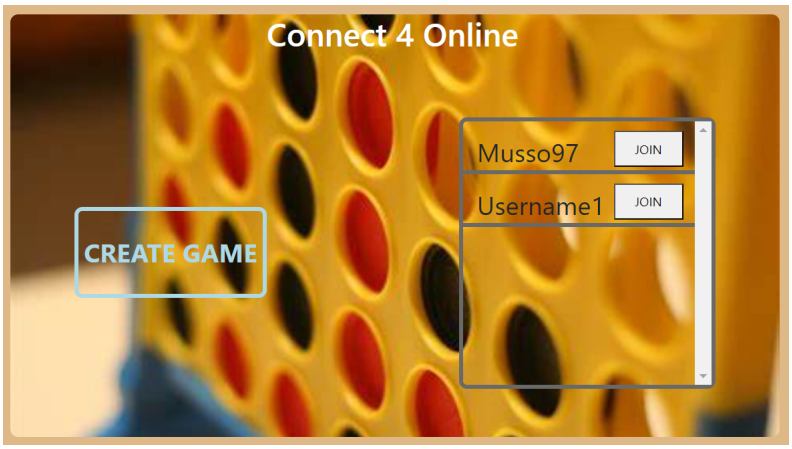
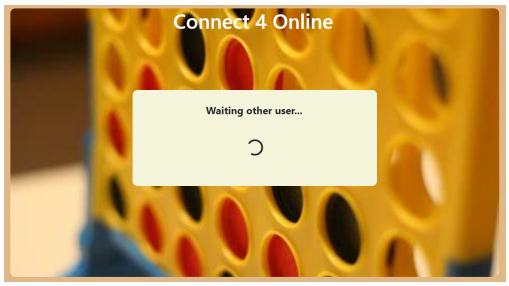
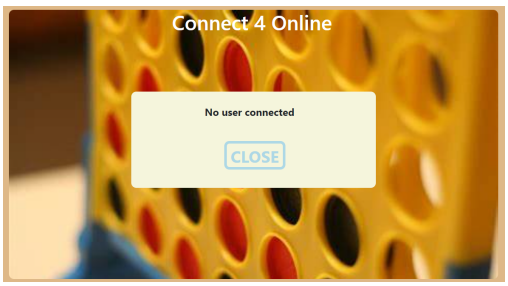


Figure 12: Home Page

Nel caso in cui l'utente decidesse di voler creare una nuova stanza, verrà mostrata una finestra di attesa. Se nessun altro giocatore si dovesse connettere alla stanza, verrà mostrato un messaggio all'utente.



(a) Waiting window



(b) Message about no one user connected

Quando ad una stanza si connettono esattamente due giocatori la partita potrà aver inizio ed entrambi gli utenti verranno riportati alla Game Page.

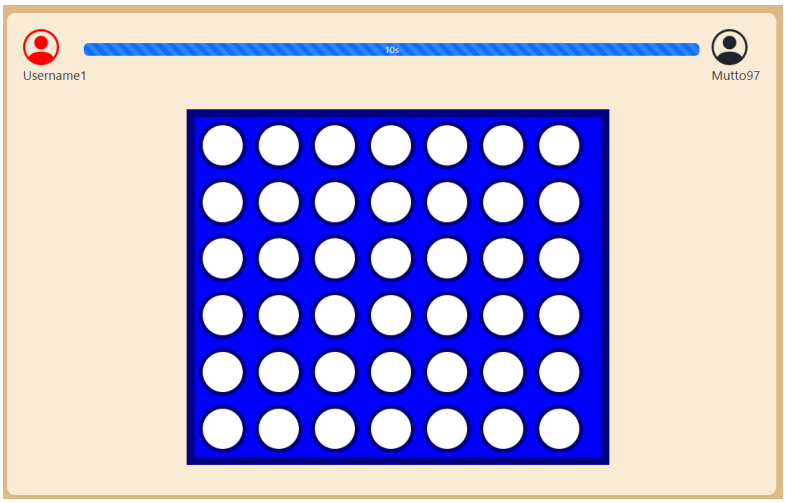


Figure 14: Game Page

L'utente che ha il diritto ad effettuare la mossa, è quello che presenta l'icona evidenziata con il colore delle propria pedina. Il turno di un utente ha una durata di 15 secondi, al termine del quale quale se non è stata effettuata alcuna mossa, di defualt il sistema posizionerà la pedina sul campo da gioco ed il turno passerà all'avversario.

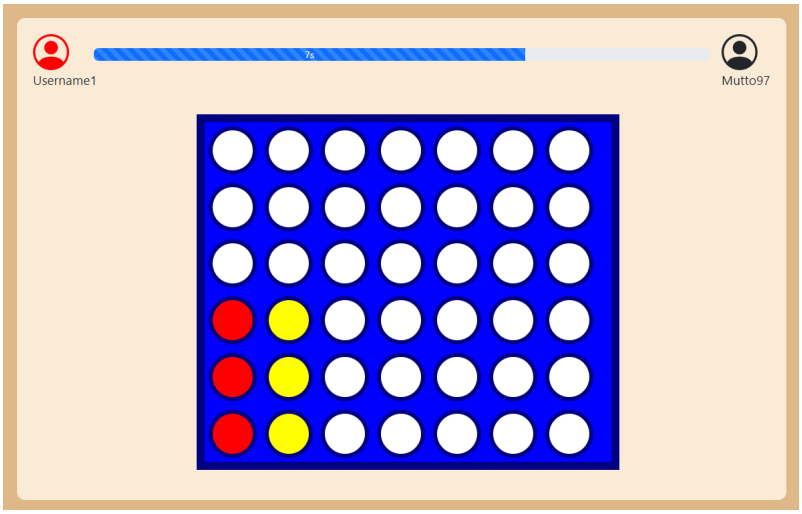
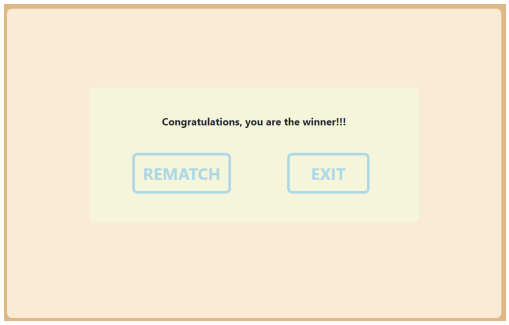
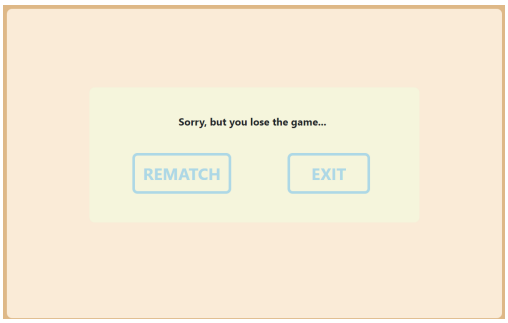


Figure 15: Set Move

Quando un giocatore posizionerà quattro pedine dello stesso colore orizzontalmente, verticalmente o diagonalmente vincerà la partita. Entrambi i giocatori visualizzeranno la finestra di fine gioco, dove verrà mostrato un messaggio personalizzato a seconda se si tratti dell'utente vincitore o perdente. Inoltre gli utenti potranno decidere se voler effettuare un rematch oppure uscire e ritornare alla propria Home Page.



(a) Winner message



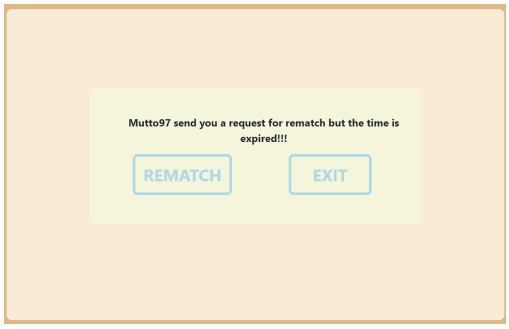
(b) Loser message

Qualora uno dei due utenti volesse effettuare un rematch, supponiamo il perdente, verrà inviata una richiesta all'avversario il quale la visualizzarà tramite messaggio. All'utente che ha effettuato la richiesta verrà mostarta una finestra di attesa.

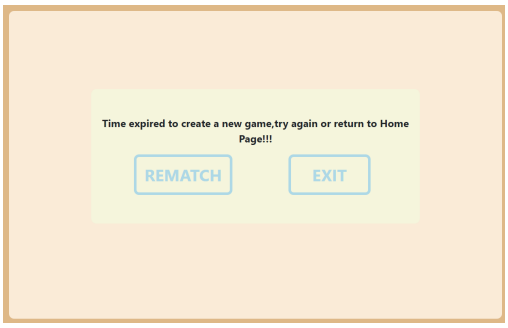
(a) Waiting window rematch

(b) Request rematch

L'utente, che ha ricevuto la richiesta di remtach, ha 20 secondi di tempo per accettarla ed inziare una nuova partita. Una volta scaduti i 20 secondi, se la richiesta non è stata accettata, ad entrambi gli utenti verrà comunicato che il tempo è scaduto e non è stato possibile avviare la partita.



(a) User message that receive a request



(b) User message that send a request

Se un utente dovesse disconnetersi durante una partita oppure decidesse di uscire una volta terminata, all'aversario verrà mostrato il seguente messaggio.

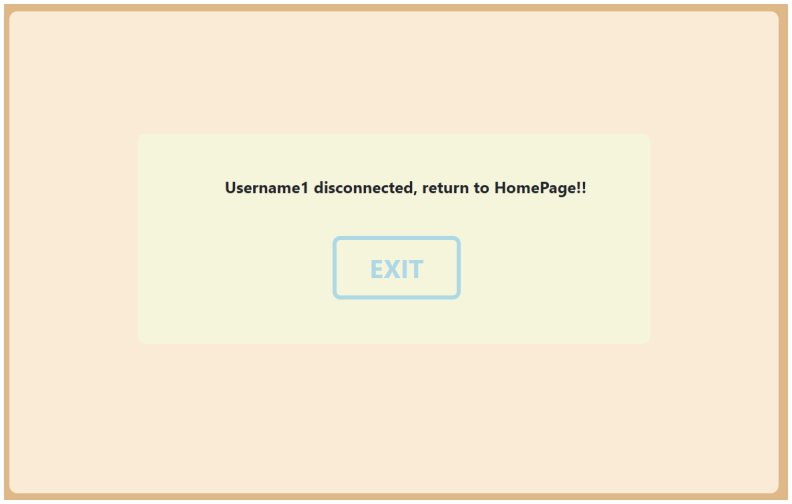


Figure 19: User disconnected

8 Conclusions

Come progetto è stato deciso di realizzare il gioco Connect4 in versione online.
La progettazione e la realizzazione del sistema si sono basati sull'implementazione di un'architettura **Client-Server** con l'utilizzo di **WebSocket** come protocollo di comunicazione.
Connect4 online consiste quindi in un gioco multiplayer in grado di garantire consistenza e scalabilità, riuscendo a gestire un numero variabile di utenti.

8.1 Future Works

In una versione futura si potrebbe aggiungere un database per rendere il sistema persistente. Grazie a questo upgrade architetturale, sarebbe possibile introdurre nuove funzionalità come:

- la realizzazione di una pagina utente personalizzata che mostra tutti i dati dell'utente comprese credenziali e risultati delle partite svolte.
- la realizzazione di una classifica globale, che mostra il numero di vittorie totalizzate da ogni singolo utente.

Inoltre si potrebbe migliorare ulteriemente l'interfaccia grafica, in modo tale da garantire una User Exaperience ancora più coinvolgente.

8.2 What did we learned

Questo progetto mi ha permesso di mettere in pratica vari aspetti appresi durante il corso, tra cui il design e l'implementazione di un sistema distribuito. Inoltre questo elaborato mi ha permesso di ampliare le mie conoscenze, grazie allo studio ed utilizzo di un protocollo di comunicazone che prima non avevo mai sperimentato ovvero **WebSocket**. Infine valuto positivamente questo progetto, poichè oltre agli aspetti che ho descritto precedentmente, mi ha permesso di approfondire maggiormente i principi legati al ciclo di vita di un progetto, a partire dall'analisi inziale dei requisiti fino alla fase di implementazione seguita da una fase di testing finale.

References

[1] Jest. `https://jestjs.io/`.

[2] Json. `https://json.org/json-it.html`.

[3] Node.js. `https://nodejs.org/en`.

[4] Websocket. `https://developer.mozilla.org/en-US/docs/Web/API/WebSocket/open_event`.