DIPLOMEOCY

Eric Michelon e Elena Sofia Pujatti

Introduzione

Il progetto riguarda la creazione di una piattaforma web dedicata al gioco da tavolo "Diplomacy". Questa piattaforma offre agli utenti la possibilità di giocare a Diplomacy online, gestendo partite tramite un sistema di tavoli virtuali e comunicando tramite una chat integrata. È importante perché rappresenta un'applicazione pratica delle competenze acquisite nel corso dell'anno scolastico nel campo dello sviluppo web, evidenziando l'integrazione di diverse tecnologie per creare un'esperienza di gioco coinvolgente e interattiva.

Materiali e Metodi Utilizzati

Il lavoro è stato svolto in modo collaborativo, sfruttando le possibilità offerte da GitHub per la gestione del codice. Questo approccio ha consentito una divisione efficiente dei compiti tra i membri del team e una tracciabilità completa delle modifiche apportate al progetto.

Per quanto riguarda i materiali e le tecnologie utilizzate, sono state adottate diverse risorse:

- Linguaggi utilizzati
  - C#
  - TypeScript
  - Html
  - CSS
  - MySql
- Framework e Librerie
  - ASP.NET Core MVC

    È stato utilizzato come il framework principale per lo sviluppo del backend. Ha fornito una struttura modulare per gestire le richieste HTTP, consentendo la creazione di servizi e la gestione dei dati.
  - Tailwind CSS

    Questa libreria CSS è stata utilizzata per progettare l'interfaccia utente, offrendo una vasta gamma di utility predefinite per lo stile e la formattazione.
  - Webpack

    È stato utilizzato per la gestione dei moduli e delle dipendenze nel front-end, ottimizzando la distribuzione del codice TypeScript e garantendo una maggiore efficienza.
  - SignalR

    Questa libreria è stata integrata per aggiungere funzionalità di chat e aggiornamenti in tempo reale all'applicazione, migliorando l'interattività e la collaborazione tra gli utenti.
  - EntityFrameworkCore

    È stato utilizzato per semplificare l'accesso e la gestione dei dati nel database, facilitando le operazioni CRUD e garantendo una maggiore efficienza nello sviluppo.

- newtonsoft

Questa libreria di .NET è stata utilizzata per la serializzazione e la deserializzazione oggetti. La libreria, nota anche come Json.NET, trasforma oggetti in stringhe JSON e viceversa. All'interno dell'applicazione è utilizzata per salvare i dati all'interno del database e per inviare informazioni riguardanti i comandi e i loro risultati.

- webrtc

Webrtc, Web Real-Time Communication, è una tecnologia che consente la comunicazione in Real-Time di audio, video e dati. All'interno del progetto è stata utilizzata per la realizzazione della chat vocale.

- MariaDB

Database relazionale utilizzato per memorizzare e gestire i dati della piattaforma.

- Librerie personalizzate di Eric Michelon

Durante lo sviluppo del progetto, sono state seguite alcune procedure chiave per garantire un'implementazione efficace e una user experience ottimale:

- Sviluppo della fase di Autenticazione
- Sviluppo dell'algoritmo di gioco
- Integrazione delle funzionalità di chat
- Connessione tra Front-end e Back-end

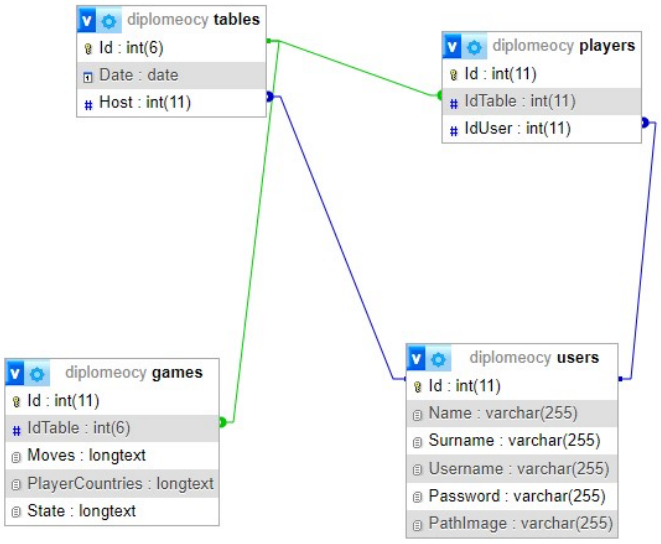
Implementazione

Per la parte di progettazione i compiti sono stati divisi in due. La parte relativa all'algoritmo di gioco e la parte di gestione del database e visualizzazione Web. L'algoritmo gestisce le varie casistiche del famoso gioco da tavolo:

Implementazione database

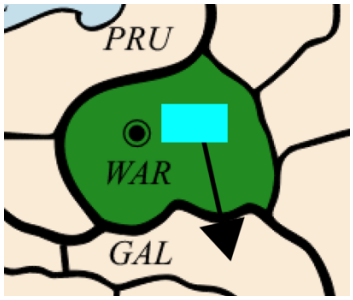
Nel database sono state implementate 4 tabelle:

- User
  Per la gestione degli utenti
- Tables
  Gestisce i tavoli di gioco
- Games
  Gestisce il gioco vero e proprio
- Players
  Gestisce i giocatori all'interno di un tavolo



Sviluppo dell'algoritmo di gioco

È stato dedicato tempo allo di un algoritmo di gioco solido e accurato, basato sulle regole e le dinamiche di Diplomacy. Questo algoritmo ha gestito in modo coerente e bilanciato le azioni dei giocatori e il progresso dello sviluppo partita, garantendo un'esperienza di gioco coinvolgente. La creazione di questo algoritmo ha richiesto l'applicazione di conoscenze avanzate di logica di gioco, gestione degli stati e decisioni algoritmiche.




```
russia.Orders.AddRange(collection: new Orders {
    new MoveOrder {
        Unit = russia.Unit(
            territory: Territories.Moscow),
        Target = game.Board.Territory(
                territory: Territories.Galicia),
    },
});
```

esempio grafico di un'azione eseguita dal giocatore con la corrispondente scrittura sotto forma di codice

Altra parte importante è data dalla chat scritta che consente di simulare le interazioni che dovrebbero avvenire se si giocasse alla versione da tavolo. I giocatori, infatti, posso simulare una stanza privata per inviarsi messaggi. L'ingresso però nel gruppo privato comporta il non poter più ricevere i messaggi degli altri Players come succederebbe se nella versione originale del gioco. In essa i Players dovrebbero lasciare la stanza e andare in un'altra in modo da rendere segreto il loro accordo, ciò comporta però il restare esclusi dalla conversazione principale

Integrazione delle funzionalità di chat

È stata implementata una chat all'interno della piattaforma per consentire ai giocatori di comunicare durante la partita, simulando l'esperienza del gioco da tavolo. Questa funzionalità è stata realizzata con l'ausilio di SignalR, una libreria che permette la comunicazione in tempo reale tra client e server. Utilizzando SignalR, i giocatori possono scambiare messaggi istantanei, pianificare strategie e negoziare alleanze, creando un'atmosfera coinvolgente e interattiva. La chat non solo arricchisce l'esperienza di gioco, ma promuove anche l'interazione sociale tra i partecipanti, riproducendo fedelmente l'aspetto sociale e collaborativo del gioco da tavolo Diplomacy.

```
public Task SendMessageToGroup(string json) {
    MessageData? data = JsonConvert.DeserializeObject<MessageData>(value: json);
    if (data is null || data.Action ≠ MessageData.MessageDataAction.SendMessageToGroup) return Task.CompletedTask;
    return Clients.Group(groupName: data.Group).SendAsync(method: "ReceiveMessage", arg1: json);
}
```

porzione di codice C# presente in una SignalR per gestire la chat

```
import chatData from './chat_data'

const sendButton = document.getElementById('send-button') as HTMLButtonElement
sendButton.disabled = true

const joinGroupButton = document.getElementById('join-group') as HTMLButtonElement
joinGroupButton.disabled = true

chatData.connect(() => {
        sendButton.disabled = false
        joinGroupButton.disabled = false

        chatData.user = userInput.value
        userInput.value = chatData.user
        chatData.joinGroup('all')
})

const userInput = document.getElementById('user-input') as HTMLInputElement
console.log(userInput)
console.log(userInput.value)
const messageInput = document.getElementById('message-input') as HTMLInputElement

sendButton.onclick = (_: Event) => {
        const user: string = userInput.value
        const message: string = messageInput.value

        chatData.user = user
        chatData.sendMessageToGroup(message)
        messageInput.value = ''
}

const joinGroupInput = document.getElementById('join-group-input') as HTMLSelectElement
joinGroupButton.onclick = async (_: Event) => {
        console.log(joinGroupInput.value)
        if (joinGroupInput.value === '') return

        const newGroup = joinGroupInput.value

        await chatData.leaveGroup(chatData.group)
        await chatData.joinGroup(newGroup)
}
```

porzione di codice typeScript per inviare messaggi e creare i gruppi

Prima di accedere alle funzionalità di gioco è necessario essere registrati ed effettuare il log in in modo da consentire il corretto funzionamento e la sicurezza necessaria per giocare.

Sviluppo della fase di autenticazione

L'autenticazione degli utenti avviene in modo semplice ma efficace, utilizzando una combinazione di sessioni e accesso al database. Questo processo garantisce la sicurezza dell'accesso alla piattaforma, consentendo agli utenti di autenticarsi in modo rapido e sicuro.

```
// POST: Users/LogIn
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> LogIn(String? username, String? password) {
    List<(string Field, string ErrorMessage)> errors = new();

    if (username is null) {
        errors.Add(("username", "Username is required"));
    }

    if (password is null) {
        errors.Add(("password", "Password is required"));
    }

    if (context.Users is null) {
        // return NotFound();
        errors.Add(("context", "Missing context"));
    }

    if (errors.Any()) {
        return this.JsonError(errors.ToArray());
    }

    User? user = await context.Users!
       .FirstOrDefaultAsync(m => m.Username == username && m.Password == password);

    if (user is not null) {
        // HttpContext.Session.SetString("UserId", user.Id.ToString());
        HttpContext.Session.Set("User", user);
    }

    return user is null ? this.JsonNotFound("user") : this.JsonRedirect(Url.Action("Details", new { user.Id })!);
}
```

esempio della funzione di login all'interno di un controller in Asp.Net MVC utilizzata nel progetto

Connessione tra Front-end e Back-end

È stata garantita una stretta integrazione tra il frontend e il back-end dell'applicazione. Utilizzando approcci moderni come architetture a microservizi e API RESTful, è stata facilitata una comunicazione fluida e efficiente tra le diverse componenti del sistema. Ciò ha consentito al frontend di accedere in modo sicuro e rapido ai dati e alle funzionalità fornite dal back-end, assicurando un'esperienza utente coesa e senza soluzione di continuità.