Basi di Dati

Progetto A.A. 2021/2022

SISTEMA DI GIOCO RISIKO ONLINE

0278821

Edoardo Manenti

Indice

1) Descrizione del Minimondo

Si vuole realizzare un servizio online che consenta di giocare ad un clone del famoso gioco Risiko, in modalità "conquista del mondo".
Al sistema hanno accesso due tipologie di utenti: i giocatori e i moderatori. I moderatori hanno la possibilità di creare stanze di gioco, in funzione della quantità di giocatori che utilizzano attualmente il sistema. In particolare, i moderatori hanno la possibilità di visualizzare, tramite un report, quante stanze hanno attualmente partite in corso e quanti giocatori stanno partecipando alla partita. Inoltre, in questo report, gli amministratori possono visualizzare il numero totale di giocatori che hanno effettuato almeno un'operazione negli ultimi 15 minuti che non sono all'interno di alcuna stanza di gioco.
Una stanza permette ad un numero massimo di sei giocatori di entrare e partecipare alla partita. Una partita coinvolge almeno tre giocatori. Quando il terzo giocatore entra in una stanza, viene attivato un countdown tale da avviare la partita dopo due minuti. In questi due minuti altri giocatori possono entrare, fino al massimo concesso.
All'avvio della partita, gli stati del tabellone vengono assegnati casualmente ai giocatori. I turni "girano" in funzione del tempo di ingresso dei giocatori nella stanza (chi è entrato prima gioca prima). Un turno prevede che il giocatore possa compiere una delle seguenti azioni:
* Posizionare un numero arbitrario di carri armati in uno stato
* Scegliere uno stato da cui fare partire un attacco verso uno stato adiacente
* Spostare carri armati da uno stato ad un altro adiacente (almeno un carro armato deve restare nello stato di partenza)

L'attacco viene svolto nel seguente modo. La fase di attacco si svolge tra il giocatore che attacca e quello che difende attraverso il lancio dei dadi. Il numero dei dadi da lanciare è stabilito dal numero di armate che si decide di schierare in guerra meno una, fino ad un massimo di tre dadi per volta. Ognuno dei giocatori lancia il numero di dadi corrispondenti, e poi si confrontano i valori ottenuti, il più alto dell'attaccante con il più alto del difensore, il secondo con il secondo e così via. Per ogni punteggio più alto, il perdente deve togliere un'armata dal tabellone. In caso di parità il punto va al difensore. Il lancio dei dadi viene simulato mediante la generazione di numeri pseudocasuali.
Se lo stato attaccato perde tutte le armate, questo viene conquistato e vengono spostate automaticamente in esso un numero di armate pari a quelle sopravvissute all'attacco.
Un apposito timer determina quando il tempo per svolgere un'azione da parte di un giocatore scade e il turno passa quindi al giocatore successivo.
Al termine del turno, se è stata svolta almeno un'azione, il giocatore riceve un numero di carri armati da posizionare pari al numero di stati posseduti diviso tre, arrotondato per eccesso.
Un'apposita procedura consente al client di sapere, quando è il turno del giocatore, tutto lo stato di gioco e quindi far scegliere quale azione effettuare. Si ricorda, comunque, che tutta la logica applicativa è implementata nel DBMS.
Un giocatore può sempre visualizzare lo storico di tutte le partite giocate.

2) Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
4-5	In funzione della quantità...	/	non aggiunge informazioni, in quanto non necessario viene rimosso
5	… di gioco. In particolare, i moderatori hanno…	…di gioco e di…	espressione ridondante
6-7	e quanti giocatori…	e quanti sono i giocatori in ciascuna di queste partite	espressione più chiara
7	amministratori	moderatori	viene unificato il termine in quanto sinonimo
8	operazione	azione	viene unificato il termine con azione (sinonimo utilizzato anche altrove nel testo)
11-12	quando il terzo giocatore entra in una stanza	quando il terzo giocatore entra nella stanza contenente la partita	espressione più specifica
13	fino al massimo concesso	...altri tre giocatori...	espressione più chiara ed esplicativa
13	/	Se un giocatore abbandona una partita in corso i suoi territori, se ancora ne possiede, vengono redistribuiti equamente in maniera casuale ai rimanenti giocatori.	aggiunta di specifiche a seguito di chiarimento con committente
14	All'avvio della partita...	...insieme ad una dotazione...	aggiunta di specifiche a seguito di chiarimento con committente
14, 18, 19, 20 , 21, 30, 35	stati/o	territori/io	termine più specifico (si evita inoltre l'ambiguità con il termine "stato di gioco" a riga 38)
15	i turni "girano"...(chi viene…)	i turni vengono assegnati secondo l'ordine di ingresso dei giocatori	espressione più chiara e concisa

18	..numero arbitrario di carri armati in …	…di carri armati ancora a disposizione…	espressione più chiara
24, 28, 30 , 31	armate/a	carri/o armati/o	viene unificato il termine in quanto sinonimo
24	guerra	attacco	viene unificato il termine in quanto sinonimo
25	giocatori	giocatori coinvolti	espressione più specifica
26,28	attaccante/difens ore	giocatore che attacca/difende	vengono unificati i termini in quanto sinonimi
28	tabellone	territorio interessato	si evitano ambiguità (il carro armato và tolto dal territorio da cui si sta attaccando/difendendo), “tabellone” è troppo generico e in questo caso usato erroneamente.
28	il punto và al difensore	il perdente è il giocatore che attacca	espressione più specifica
31	…un numero di armate pari a…	…un numero di carri armati dell’attaccante pari a…	si evitano ambiguità (si rende chiaro che i carri armati da spostare sono quelli dell’attaccante)
32-33	Un apposito timer...successivo	Un apposito timer determina la durata di ciascun turno.	espressione più concisa
34	…almeno un’azione	una delle azioni specificate precedentemente	espressione meno ambigua
34	…almeno un’azione	Il giocatore può decidere di saltare il turno.	aggiunta di specifiche a seguito di chiarimento con committente

Specifica disambiguata

Si vuole realizzare un servizio online che consenta di giocare ad un clone del famoso gioco Risiko, in modalità "conquista del mondo".
Al sistema hanno accesso due tipologie di utenti: i giocatori e i moderatori. I moderatori hanno la possibilità di creare stanze di gioco e di visualizzare, tramite un report, quante stanze hanno attualmente partite in corso e quanti sono i giocatori in ciascuna di queste partite. Inoltre, in questo report, i moderatori possono visualizzare il numero totale di giocatori che hanno effettuato almeno un'azione negli ultimi 15 minuti che non sono all'interno di alcuna stanza di gioco. Una stanza permette ad un numero massimo di sei giocatori di entrare e partecipare alla partita. Una partita coinvolge almeno tre giocatori. Quando il terzo giocatore entra nella stanza contenente la partita, viene attivato un countdown tale da avviare la partita dopo due minuti. In questi due minuti altri tre giocatori possono entrare. Se un giocatore abbandona una partita in corso i suoi territori, se ancora ne possiede, vengono redistribuiti equamente in maniera casuale ai rimanenti giocatori.

All'avvio della partita, i territori del tabellone vengono assegnati casualmente ai giocatori insieme ad una dotazione iniziale di carri armati così composta:
* 35 carri armati se si gioca in 3
* 30 carri armati se si gioca in 4
* 25 carri armati se si gioca in 5
* 20 carri armati se si gioca in 6

Si assume che ogni giocatore cominci la partita con un carro armato in ciascun territorio assegnato. I turni vengono assegnati secondo l'ordine di ingresso dei giocatori nella stanza.
Un turno prevede che il giocatore possa compiere una delle seguenti azioni:
* Posizionare un numero arbitrario di carri armati ancora a disposizione in un territorio
* Scegliere un territorio da cui fare partire un attacco verso un territorio adiacente
* Spostare carri armati da un territorio ad un altro adiacente (almeno un carro armato deve restare nel territorio di partenza)

L'attacco viene svolto nel seguente modo. La fase di attacco si svolge tra il giocatore che attacca e il giocatore che difende attraverso il lancio dei dadi. Il numero dei dadi da lanciare è stabilito dal numero di carri armati che si decide di schierare in attacco meno uno, fino ad un massimo di tre dadi per volta. Ognuno dei giocatori coinvolti lancia il numero di dadi corrispondenti, e poi si confrontano i valori ottenuti, il più alto del giocatore che attacca con il più alto del giocatore che difende, il secondo con il secondo e così via. Per ogni punteggio più alto, il perdente deve togliere un carro armato dal territorio interessato. In caso di parità il perdente è il giocatore che attacca. Il lancio dei dadi viene simulato mediante la generazione di numeri pseudocasuali.
Se il territorio attaccato perde tutti i carri armati, questo viene conquistato e vengono spostati automaticamente in esso un numero di carri armati dell'attaccante pari a quelli sopravvissuti all'attacco.
Un apposito timer determina la durata di ciascun turno.
Al termine del turno, se è stata svolta una delle azioni specificate precedentemente, il giocatore riceve un numero di carri armati da posizionare pari al numero di territori posseduti diviso tre, arrotondato per eccesso. Il giocatore può decidere di saltare il turno.
Un'apposita procedura consente al client di sapere, quando è il turno del giocatore, tutto lo stato di gioco e quindi far scegliere quale azione effettuare. Si ricorda, comunque, che tutta la logica applicativa è implementata nel DBMS.
Un giocatore può sempre visualizzare lo storico di tutte le partite giocate.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Territorio	Porzione del tabellone assegnata ai giocatori. Può essere conquistato	Stato	Giocatore, Partita

	e da esso possono partire attacchi solo verso territori ad esso adiacenti. Su di esso è possibile posizionare un certo numero di carri armati.		
Giocatore	Utente partecipante alla partita, possiede territori e una dotazione di carri armati.	/	Partita, Territorio, Turno
Turno	Porzione della partita in cui il giocatore che svolge il turno può svolgere una (o nessuna) delle seguenti azioni: attaccare, posizionare carri armati, spostare carri armati.	/	Partita, Giocatore
Moderatore	Utente con privilegi superiori, può creare nuove stanze e visualizzare report sulle informazioni di gioco.	Amministratore	Stanza di gioco
Stanza di gioco	Luogo virtuale creato dai moderatori e dove i giocatori possono entrare e partecipare alla partita contenuta al suo interno.	Stanza	Partita, Moderatore
Partita	Sessione di gioco contenuta in una stanza di gioco, possono parteciparvi dai 3 ai 6 giocatori.	/	Stanza di gioco, Territorio, Giocatore, Turno

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative ai Giocatori
Un massimo di sei giocatori si affrontano in una partita. Un giocatore, all'inizio della partita, ottiene una dotazione di territori e carri armati da posizionare.

Un turno prevede che il giocatore possa compiere una delle seguenti azioni:
* Posizionare un numero arbitrario di carri armati ancora a disposizione in un territorio
* Scegliere un territorio da cui fare partire un attacco verso un territorio adiacente
* Spostare carri armati da un territorio ad un altro adiacente (almeno un carro armato deve restare nel territorio di partenza)

Al termine del turno, se è stata svolta una delle azioni specificate precedentemente, il giocatore riceve un numero di carri armati da posizionare pari al numero di territori posseduti diviso tre, arrotondato per eccesso.

Se un giocatore abbandona una partita in corso i suoi territori, se ancora ne possiede, vengono redistribuiti equamente in maniera casuale ai rimanenti giocatori.

Un giocatore può sempre visualizzare lo storico di tutte le partite giocate.

Frasi relative ai Turni

I turni vengono assegnati secondo l'ordine di ingresso dei giocatori nella stanza. Un turno prevede che il giocatore possa compiere una delle seguenti azioni:
* Posizionare un numero arbitrario di carri armati ancora a disposizione in un territorio
* Scegliere un territorio da cui fare partire un attacco verso un territorio adiacente
* Spostare carri armati da un territorio ad un altro adiacente (almeno un carro armato deve restare nel territorio di partenza)

Un apposito timer determina la durata di ciascun turno.

Un'apposita procedura consente al client di sapere, quando è il turno del giocatore, tutto lo stato di gioco e quindi far scegliere quale azione effettuare.

Al termine del turno, se è stata svolta una delle azioni specificate precedentemente, il giocatore riceve un numero di carri armati da posizionare pari al numero di territori posseduti diviso tre, arrotondato per eccesso.

Frasi relative ai Moderatori

I moderatori hanno la possibilità di creare stanze di gioco e di visualizzare, tramite un report, quante stanze hanno attualmente partite in corso e quanti sono i giocatori in ciascuna di queste partite. Inoltre, in questo report, i moderatori possono visualizzare il numero totale di giocatori che hanno effettuato almeno un'azione negli ultimi 15 minuti che non sono all'interno di alcuna stanza di gioco.

Frasi relative alle Partite

Una partita coinvolge da un minimo di tre ad un massimo di sei giocatori.
Si vuole tenere traccia delle partite giocate e di quali partite sono in corso.
Di queste ultime, si tiene inoltre traccia di quanti sono i giocatori partecipanti.

Frasi relative alle Stanze di gioco

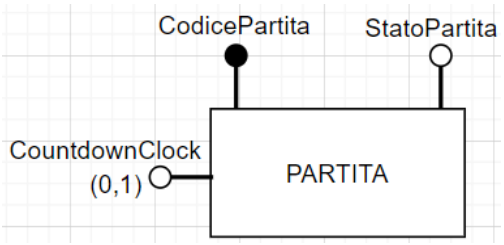
Una stanza permette ad un numero massimo di sei giocatori di entrare e partecipare alla partita. Quando il terzo giocatore entra nella stanza che contiene la partita, viene attivato un countdown tale da avviare la partita dopo due minuti. In questi due minuti altri tre giocatori possono entrare.

Frasi relative ai Territori
I territori del tabellone vengono assegnati casualmente ai giocatori all'inizio della partita e vengono popolati inizialmente da un carro armato. Un territorio può essere coinvolto in un attacco, se il territorio attaccato perde tutti i carri armati, questo viene conquistato e vengono spostati automaticamente in esso un numero di carri armati dell'attaccante pari a quelli sopravvissuti all'attacco. Se un giocatore abbandona una partita in corso i suoi territori, se ancora ne possiede, vengono redistribuiti equamente in maniera casuale ai rimanenti giocatori.

3) Progettazione concettuale

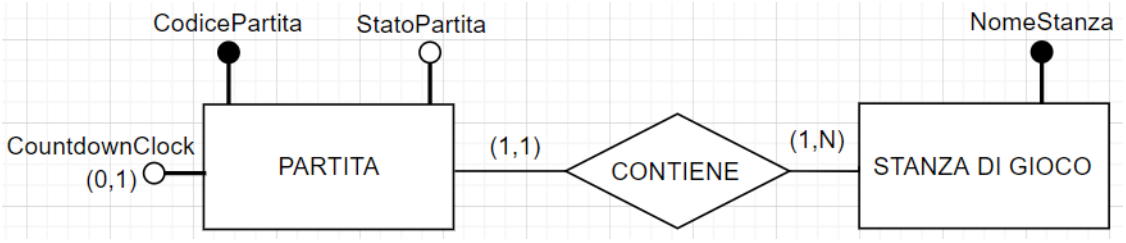
Costruzione dello schema E-R

La prima entità è 'PARTITA' di cui si vuole tenere traccia delle informazioni riguardanti la gestione delle tempistiche della partita con l'attributo 'CountdownClock' che salverà l'istante di inizio del countdown per l'avvio della partita (sarà NULL finché non verrà inizializzato quando il terzo giocatore partecipa alla partita) e delle informazioni riguardanti lo stato della partita (finalizzato alla storicizzazione e per tenere traccia del fatto che se la partita è iniziata ad esempio andrà impedito a nuovi giocatori di entrare). Come identificativo di partita si è scelto, dato il contesto dei giochi online, un codice univoco di partita.



A questo punto viene inserita l'entità 'STANZA DI GIOCO' identificata da un nome della stanza che sarà quindi univoco nella base di dati e conterrà la partita che si sta svolgendo al suo interno.

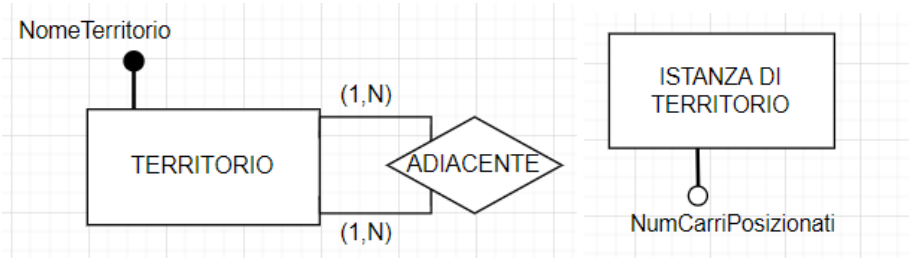
Per il motivo precedentemente espresso l'entità 'STANZA DI GIOCO' e posta in relazione con l'entità 'PARTITA' attraverso una relazione che esplicita il fatto che la seconda è contenuta nella prima.



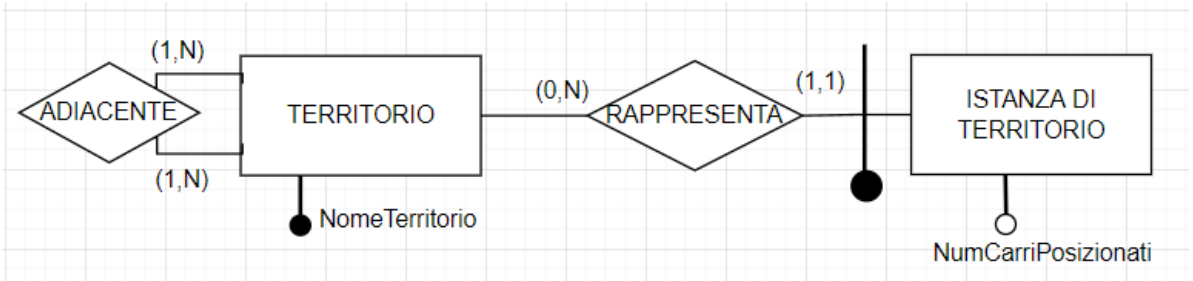
Nel caso dei territori bisogna invece fare una distinzione tra:

- l'astrazione 'TERRITORIO' che avrà determinate adiacenze fissate (per cui aggiungiamo la relazione ricorsiva 'ADIACENTE') e un nome che ragionevolmente lo identifica tra i 42 presenti nel gioco di Risiko.

-l''ISTANZA DI TERRITORIO' che sarà invece un'altra entità che rappresenta tale territorio in una determinata partita e che terrà traccia ad esempio del numero di carri armati posizionati su di esso in quella determinata partita

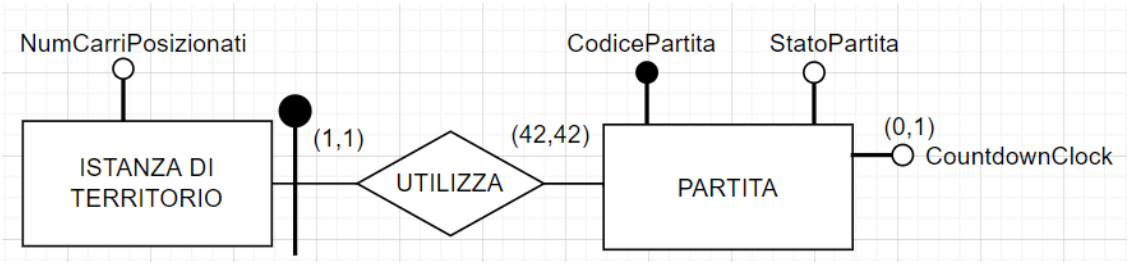


Le due entità appena introdotte sono relazionate nel seguente modo dalla relazione 'RAPPRESENTA' che vede un 'TERRITORIO' che può avere fino a N rappresentazioni (in N partite diverse) e vede 'ISTANZA DI TERRITORIO' debole rispetto al territorio che essa rappresenta (di conseguenza mettiamo cardinalità massima e minima pari a 1).

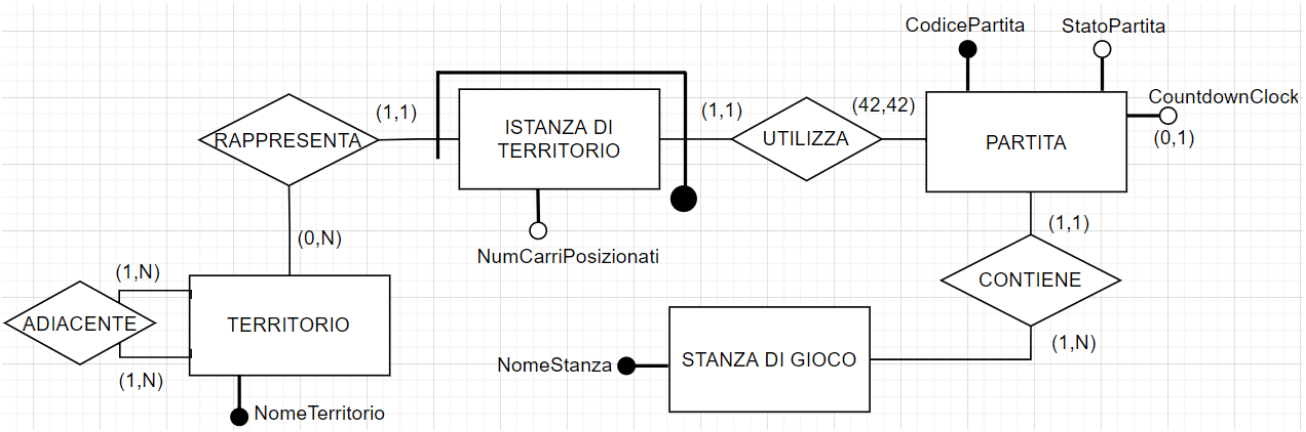


L'entità 'ISTANZA DI TERRITORIO' è, come anticipato, debole anche rispetto alla partita che la utilizza (in particolare in ogni partita avremo sempre 42 istanze di territorio e quindi mettiamo

cardinalità massima e minima a quel valore mentre anche qui essendo entità debole le cardinalità dell'istanza sono messe pari ad 1)



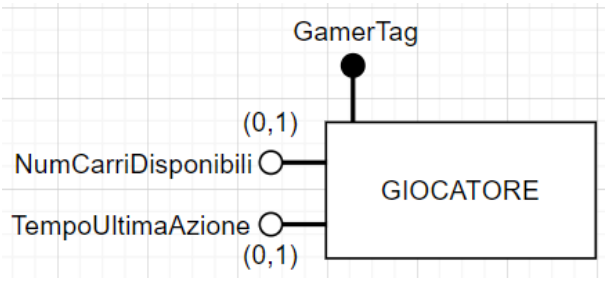
Unendo quindi i precedenti schemi si ottiene il seguente schema



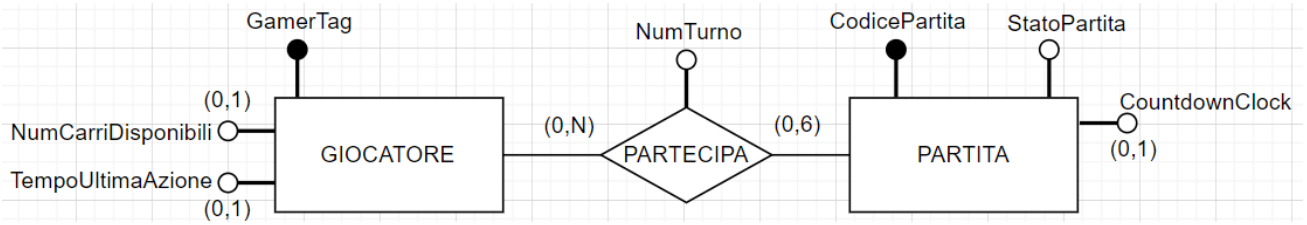
Per quanto riguarda l'entità 'GIOCATORE', si avranno due attributi opzionali per indicare il numero di carri armati disponibili (ovvero da poter ancora posizionare sui territori in suo possesso, inizializzato solamente una volta assegnata la dotazione iniziale all'inizio di una partita) e un attributo per tenere traccia dell'istante in cui è avvenuta l'ultima azione del giocatore (anch'esso opzionale dato che verrà inizializzato solo in seguito alla prima azione svolta da un giocatore) utile per il report che tiene traccia dei giocatori che hanno compiuto un'azione negli ultimi 15 minuti.

Come identificativo invece, sempre pensando al contesto dei giochi online, si è scelto un "GamerTag".

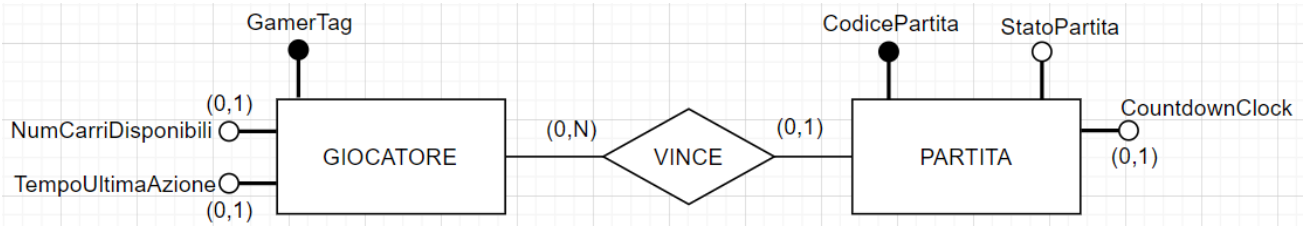
(NB: si è volutamente evitato di inserire le entità 'UTENTE' e 'MODERATORE' e della conseguente generalizzazione di 'GIOCATORE' e 'MODERATORE' su 'UTENTE' dato che il concetto di moderatore riguarda uno specifico ruolo nel sistema e tale gestione può essere fatta successivamente con l'introduzione della tabella utenti, dei ruoli e della procedura di login come visto in classe)



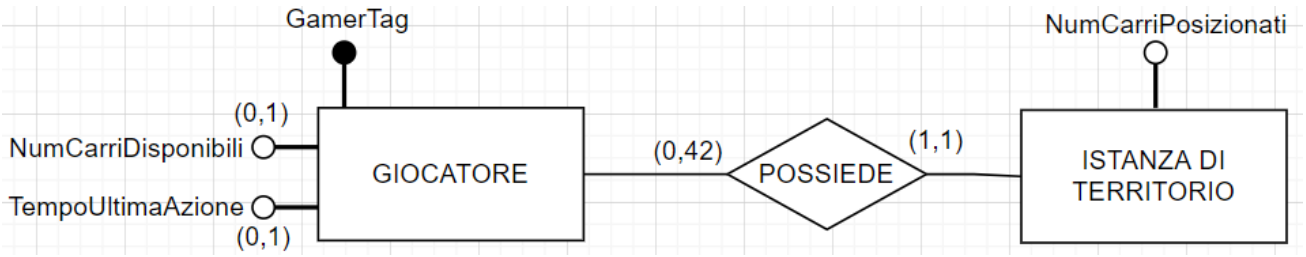
Tra le entità ‘GIOCATORE’ e ‘PARTITA’ introduciamo quindi una relazione per modellare la partecipazione ad una partita tale anche da rendere possibile la storicizzazione di eventuali partite giocate in passato (qualora l’attributo StatoPartita fosse impostato a ‘terminata’ per intenderci). La relazione di partecipazione dovrà tenere poi traccia di un attributo che indica il numero di turno assegnato in base all’ordine di partecipazione. Inoltre per le cardinalità della relazione ‘PARTECIPA’ ragionevolmente un giocatore può aver giocato da 0 ad N partite e per ogni partita dal regolamento possono partecipare fino a 6 giocatori. (NB: Non si è volutamente inserita una relazione ridondante tra giocatore e ‘STANZA DI GIOCO’ dato che come concordato con il committente l’entrata in una stanza comporta in automatico la partecipazione alla partita contenuta in essa)



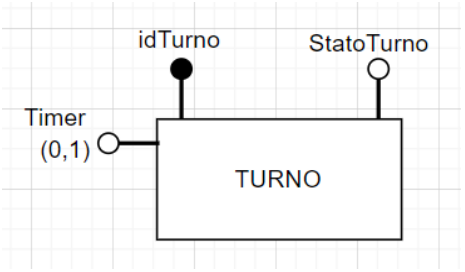
Per tenere traccia del vincitore introduciamo un’altra relazione ‘VINCE’ (per le cardinalità ovviamente un giocatore può non aver mai vinto o può aver vinto N partite e ogni singola partita avrà un solo vincitore ma solo se questa è terminata e quindi cardinalità minima 0 e massima 1).



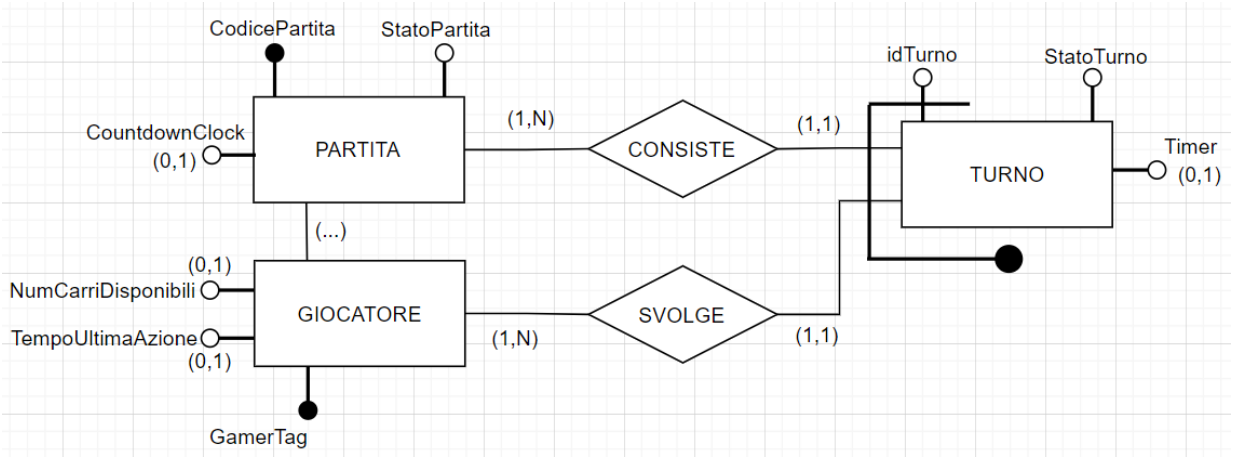
Non ci resta che modellare la relazione di proprietà ‘POSSIEDE’ tra l’entità giocatore che ragionevolmente può avere da 0 (in caso di perdita) a N (in caso di vittoria) territori e ‘ISTANZA DI TERRITORIO’ che avrà un unico proprietario.



Come da specifica *'la logica applicativa è implementata nel DBMS'*, per fare ciò abbiamo bisogno di introdurre un'altra entità che è quella di 'TURNO'.

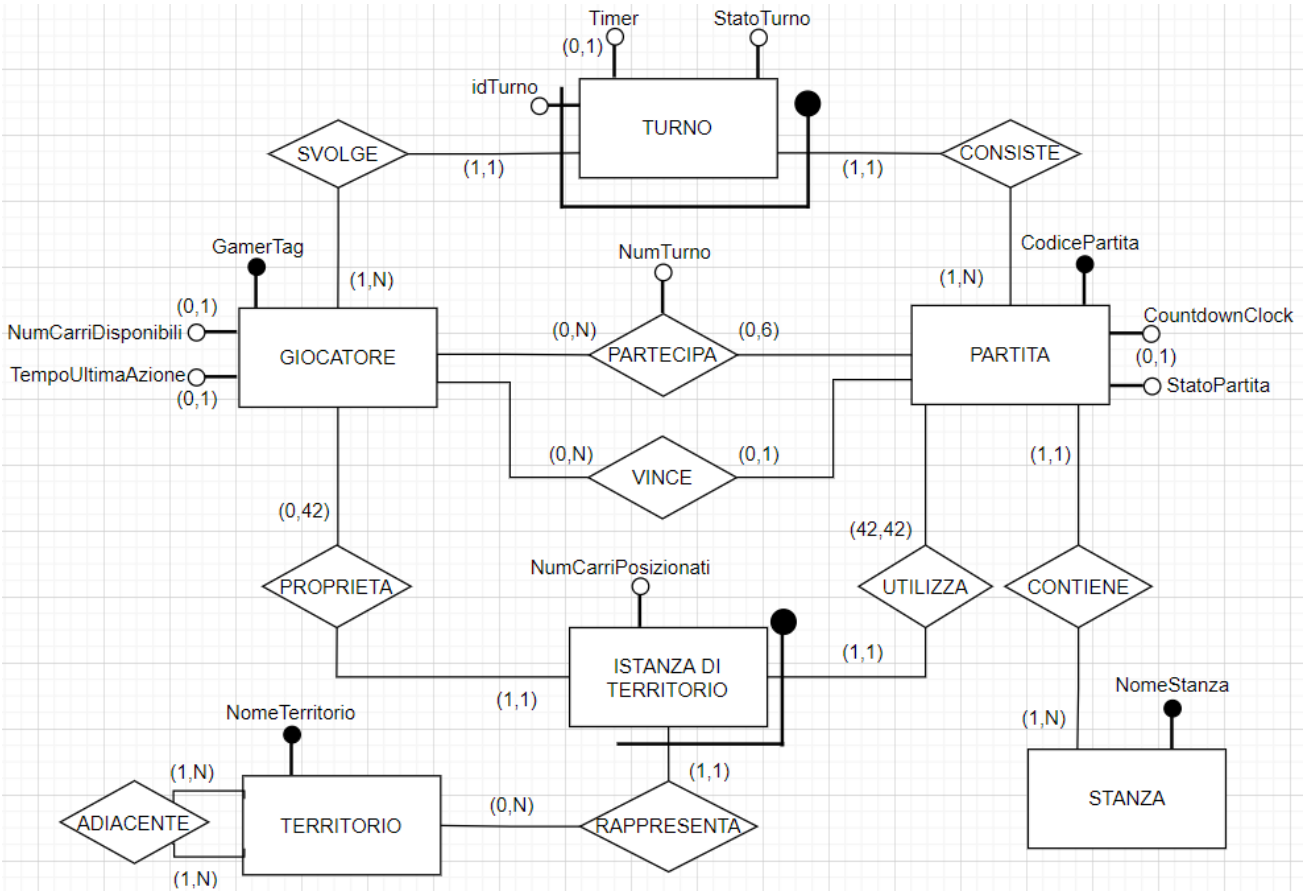


A questo punto è ragionevole introdurre le due relazioni 'CONSISTE' e 'SVOLGE' che legano tale entità rispettivamente alle entità di 'PARTITA' (che consiste in una serie di turni, da 1 a N per l'appunto) e 'GIOCATORE' (il quale all'interno di quella partita potrà svolgere da 1 a N turni), chiaramente per quel giocatore in quella partita ci possono essere più turni che verranno quindi distinti grazie ad un ulteriore attributo 'idTurno'. Per questa entità si è voluto poi tenere traccia, con l'attributo 'Timer' (che sarà NULL finché il turno non avrà inizio), del tempo di inizio del turno in modo tale da effettuare il cambio di turno allo scadere di un tempo prestabilito e, con l'attributo 'StatoTurno' (che di default sarà 'noaction') per capire se il giocatore sta effettuando un'azione o meno (se il valore rimane 'noaction' non verranno ad esempio assegnati, allo scadere del timer, i carri armati come da specifica).



NB: Tra giocatore e partita il '(..)' sta ad indicare le relazioni tra le due entità che sono state omesse per motivi di leggibilità di questo schema parziale

Integrazione finale



Regole aziendali

1. L'attributo StatoPartita in Partita può assumere esclusivamente i seguenti valori: 'wait', 'exec', 'end'
2. L'attributo StatoTurno in Turno può assumere esclusivamente i seguenti valori: 'noaction', 'action','end'.
3. L'attributo NumTurno in Partecipa può assumere esclusivamente valori nell'intervallo [1,6]
4. In una stanza deve esserci un'unica partita con stato 'exec'.
5. Un giocatore può partecipare ad una partita per volta (fatta eccezione chiaramente di tutte le partecipazioni a partite con StatoPartita 'end').
6. L'attributo 'NumCarriPosizionati' in istanza di territorio ha sempre valore >= 1.
7. Una partita inizia solo se ci sono almeno 3 giocatori.

8. Un giocatore non può attaccare i suoi stessi territori, anche se adiacenti.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
PARTITA	Sessione di gioco contenuta in una stanza	CodicePartita, CountdownClock, StatoPartita	CodiceParita
STANZA DI GIOCO	Luogo virtuale creato dai moderatori e dove i giocatori possono entrare e partecipare alla partita contenuta al suo interno.	NomeStanza	NomeStanza
TERRITORIO	Astrazione di territorio.	NomeTerritorio	NomeTerritorio
TURNO	Porzione della partita in cui il giocatore che svolge il turno può svolgere una (o nessuna) delle seguenti azioni: attaccare, posizionare carri armati, spostare carri armati.	idTurno, StatoTurno, Timer	(idTurno,Partita,Giocatore)
ISTANZA DI TERRITORIO	Rappresentazione del territorio all'interno della singola partita all'interno della quale è legata al giocatore proprietario. Tiene traccia del numero di carri armati posizionati su di esso.	NumCarriPosizionati	(Partita, Territorio)
GIOCATORE	Partecipante alla partita, possiede territori e una dotazione di carri armati.	GamerTag, NumCarriDisponibili, TempoUltimaAzione	GamerTag

4) Progettazione logica

Volume dei dati

Concetto nello schema	Tipo[1]	Volume atteso
GIOCATORE	E	50000
PARTITA	E	5000000
STANZA DI GIOCO	E	10000 (almeno 8334 per far giocare tutti ma dobbiamo anche considerare che non tutte le partite iniziano con 6 giocatori all'interno e c'è quindi necessità di un numero maggiore di stanze di gioco)
TERRITORIO	E	42 (costante)
TURNO	E	< 900000000(limite superiore: media di 30 turni per giocatore contando 6 giocatori per ogni partita)
ISTANZA DI TERRITORIO	E	210000000
CONTIENE	R	5000000 (= partita)
SVOLGE	R	< 900000000 (= turno)
CONSISTE	R	< 900000000 (= turno)
PARTECIPA	R	< 30000000 (limite superiore: meno di 6 giocatori a partita)
VINCE	R	< 5000000 (limite superiore: solo quelle terminate)
PROPRIETA	R	210000000 (=istanza di territorio)
UTILIZZA	R	210000000 (=istanza di territorio)
ADIACENTE	R	81 (costante, si omettono le adiacenze simmetriche)
RAPPRESENTA	R	210000000 (=istanza di territorio)

Tavola delle operazioni

Si considera una frequenza attesa delle operazioni nel caso di sistema a regime.

Cod.	Descrizione	Frequenza attesa
OP1	Registrare un nuovo giocatore	5/g
OP2	Creare Stanza di Gioco	1/w
OP3	Visualizzare quante stanze hanno attualmente partite in corso e quanti sono i giocatori in ciascuna di queste partite. Inoltre visualizzare il numero totale di giocatori che hanno effettuato almeno un'azione negli ultimi 15 minuti che non sono	10/d

[1] Indicare con E le entità, con R le relazioni

	all'interno di alcuna stanza di gioco. (visualizzare report moderatori)	
OP4	Partecipare ad una partita	100000/d (a regime assumiamo che i giocatori faranno un paio di partite al giorno)
OP5	Abbandonare una partita	10000/d (assumiamo che il 10& di questi giocatori abbandonerà almeno una di quelle due partite)
OP6	Visualizzare stato di gioco	18000000/d (assumendo che per ogni turno nella partita ogni giocatore guardi una volta lo stato di gioco)
OP7	Posizionare carri armati in territorio	< 6000000/d (assumendo che qualche turno venga saltato, nel resto dei turni assumiamo per semplicità che venga fatta in media un terzo delle volte)
OP8	Spostare carri armati da un territorio ad un altro	< 6000000/d (assumendo che qualche turno venga saltato, nel resto dei turni assumiamo per semplicità che venga fatta in media un terzo delle volte)
OP9	Attaccare da un territorio ad un altro	< 6000000/d (assumendo che qualche turno venga saltato, nel resto dei turni assumiamo per semplicità che venga fatta in media un terzo delle volte)
OP10	Visualizzare storico partite	50000/d (assumendo che per qualche giocatore che controllerà più di una volta al giorno lo storico delle sue partite c'è un giocatore che non gioca alcuna partita e non controlla lo storico delle partite)

Costo delle operazioni

Si assume il costo di un accesso in scrittura pari al doppio di un accesso in lettura (che viene preso come unità).

Operazione 1:

Accesso in scrittura per l'inserimento dell'entità 'GIOCATORE'.

Concetto	Costrutto	Accessi	Tipo
GIOCATORE	E	1	S

COSTO TOTALE OPERAZIONE = 1*2 = 2

Operazione 2:

Accesso in scrittura sulla 'STANZA DI GIOCO' che viene inserita e conseguente accesso in scrittura per inserire la prima 'PARTITA' all'interno della stanza che sarà pronta ad ospitare giocatori.

Concetto	Costrutto	Accessi	Tipo
STANZA DI GIOCO	E	1	S
PARTITA	E	1	S

COSTO TOTALE OPERAZIONE = 1*2 + 1*2 = 4

Operazione 3:

Per ogni ‘STANZA DI GIOCO’ sarà necessario un accesso in lettura e assumendo N ‘PARTITA’ in corso (con N minore uguale del numero di stanze di gioco essendoci al massimo una partita in corso per stanza di gioco) avrò N accessi in lettura e assumendo 6 giocatori a partita avrò 6*N accessi in lettura su ‘PARTECIPA’ per il conteggio dei partecipanti. Per il resto del report avremo 50000 accessi in lettura su ‘GIOCATORE’ dove verrà controllato l’attributo ‘TempoUltimaAzione’ e per ognuno dei giocatori che avrà il timer con un tempo maggiore rispetto ad un intervallo di 15 minuti passati (assumiamo che questo avvenga per M giocatori) faremo K accessi in lettura su ‘PARTECIPA’ (dove K si è assunto essere il numero di partecipazioni per quel giocatore)

Concetto	Costrutto	Accessi	Tipo
STANZA DI GIOCO	E	10000	L
PARTITA	E	N	L
PARTECIPA	R	6*N + M*K	L
GIOCATORE	E	50000	L

COSTO TOTALE OPERAZIONE = 10000 + N + 6*N + M*K + 50000 = 60000 + 7N + M*K

Operazione 4:

Accesso in scrittura per l’inserimento della partecipazione su ‘PARTECIPA’.

Concetto	Costrutto	Accessi	Tipo
PARTECIPA	R	1	S

COSTO TOTALE OPERAZIONE = 1*2 = 2

Operazione 5:

Come per l’operazione precedente (in questo caso invece dell’inserimento avremo un accesso in scrittura per una rimozione in ‘PARTECIPA’) ma bisognerà inoltre controllare se la partita è in corso (accesso in lettura su ‘PARTITA’) e in tal caso controllare se è il turno del giocatore (unico caso in cui questa operazione è concessa in caso di partita in corso, e quindi un accesso in lettura a ‘TURNO’ dovendo leggere lo stato dell’ultimo turno del giocatore in questione) e in seguito aggiornare il numero di turno delle partecipazioni degli altri giocatori (nel caso peggiore 5 giocatori e di conseguenza 5 accessi in scrittura su ‘PARTECIPA’) e redistribuire le istanze di territorio che quel giocatore possedeva (sarà necessario leggere se il giocatore in questione è il proprietario per tutte e 42 le ‘ISTANZA DI TERRITORIO’ e assumendo che, nel caso dei 6 giocatori, il giocatore

che sta abbandonando la partita si scopra possedere ⅛ delle 'ISTANZA DI TERRITORIO' sarà necessario per ognuna di esse un accesso in scrittura per aggiornarne il proprietario)

Concetto	Costrutto	Accessi	Tipo
PARTECIPA	R	1+5	S
PARTITA	E	1	L
TURNO	E	1	L
ISTANZA DI TERRITORIO	E	42 7	L S

COSTO TOTALE OPERAZIONE = 6*2 + 1 + 1 + 42 + 7*2 = 70

Operazione 6:

Consideriamo i 42 accessi in lettura su 'ISTANZA DI TERRITORIO' e 'POSSIEDE' necessari per restituire lo stato di gioco insieme ai 6 accessi in lettura su 'GIOCATORE' (assumendo 6 giocatori) per avere informazioni sui carri disponibili per ogni giocatore che sta partecipando alla partita.

Concetto	Costrutto	Accessi	Tipo
ISTANZA DI TERRITORIO	E	42	L
POSSIEDE	R	42	L
GIOCATORE	E	1	S

COSTO TOTALE OPERAZIONE = 42 + 42 + 1*2 = 86

Operazione 7:

Sarà necessario un accesso preliminare in lettura su 'GIOCATORE' per vedere il numero di carri che ha a disposizione e in lettura su 'POSSIEDE' per verificare che l'istanza di territorio sia di sua proprietà per poi andare a fare un accesso in scrittura su 'ISTANZA DI TERRITORIO' e aggiornare il numero di carri posizionati.

Concetto	Costrutto	Accessi	Tipo
GIOCATORE	E	1	L
POSSIEDE	R	1	L
ISTANZA DI TERRITORIO	E	1	S

COSTO TOTALE OPERAZIONE = 1 + 1 + 1*2 = 4

Operazione 8:

Simile al caso precedente tuttavia le letture preliminari andranno fatte come segue: 2 accessi in lettura su 'POSSIEDE' per vedere se il giocatore è proprietario di entrambi i territori, 1 accesso in lettura su 'ISTANZA DI TERRITORIO' per capire se sul territorio da cui origina lo spostamento ci

sia posizionato un numero sufficiente di carri armati da spostare, 81 accessi in lettura su 'ADIACENTE' per verificare l'adiacenza dei due territori. In seguito rimarranno i 2 accessi in scrittura per aggiornare il numero di carri armati posizionati su entrambe le istanze di territorio.

Concetto	Costrutto	Accessi	Tipo
POSSIEDE	R	2	L
ISTANZA DI TERRITORIO	E	1 2	L S
ADIACENTE	R	81	L

COSTO TOTALE OPERAZIONE = 2 + 1 + 2*2 + 81 = 88

Operazione 9:

Molto simile al caso precedente salvo il fatto che i 2 accessi in lettura su 'POSSIEDE' sono motivati dal fatto che un giocatore non può attaccare i suoi stessi territori.

Concetto	Costrutto	Accessi	Tipo
POSSIEDE	R	2	L
ISTANZA DI TERRITORIO	E	1 2	L S
ADIACENTE	R	81	L

COSTO TOTALE OPERAZIONE = 2 + 1 + 2*2 + 81 = 88

Operazione 10:

Per l'ultima operazione abbiamo bisogno di N accessi in lettura in 'PARTECIPA' (dove N è il numero di partecipazioni del giocatore che visualizza lo storico) e altrettanti accessi in lettura in 'PARTITA' per controllare per ogni partita che sia terminata e in tal caso (N-1 considerando che il giocatore al massimo sarà in 1 partita) fare 1 accesso in lettura su 'STANZA DI GIOCO' e uno 'VINCE' per leggere chi è il vincitore e di quale stanza faceva parte quella partita.

Concetto	Costrutto	Accessi	Tipo
PARTECIPA	R	N	L
PARTITA	E	N	L
STANZA DI GIOCO	E	N-1	L
VINCE	R	N-1	L

COSTO TOTALE OPERAZIONE = N + N + N-1 + N-1 = 4N-2

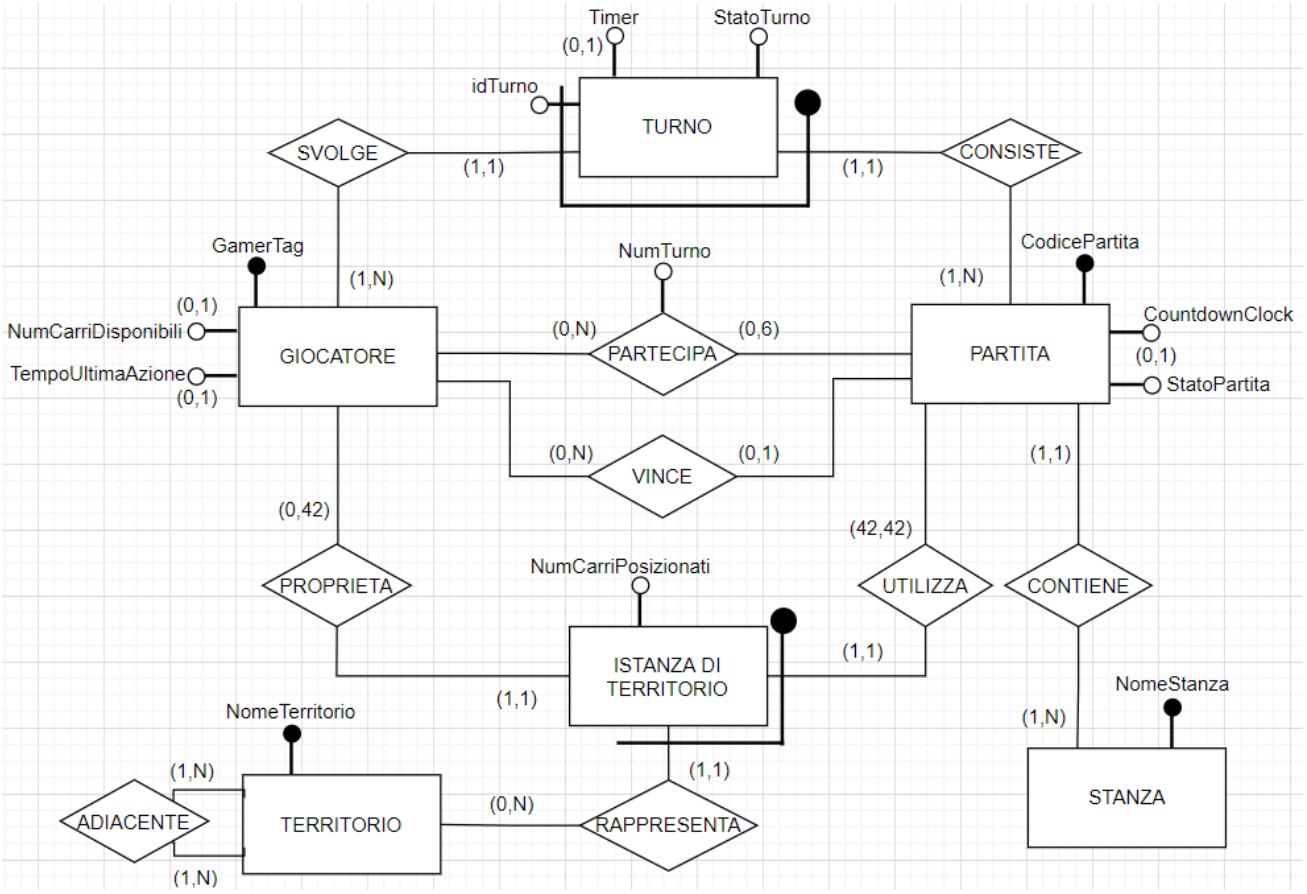
Ristrutturazione dello schema E-R

Non essendoci generalizzazioni o ridondanze non è stata apportata alcuna modifica allo schema proposto nella precedente "integrazione finale".

Trasformazione di attributi e identificatori

Nessuna trasformazione realizzata.

Traduzione di entità e associazioni



Partendo dal diagramma ER ristrutturato di sopra, costruiamo le seguenti relazioni.

STANZADIGIOCO(NomeStanza)

TERRITORIO(NomeTerritorio)

GIOCATORE(GamerTag, NumCarriDisponibili*, TempoUltimaAzione*)

PARTITA(CodicePartita, Stanza, TimerTurno*, CountdownClock*, FinePartita*, Vincitore*)

• PARTITA(Vincitore) ⊆ GIOCATORE(GamerTag)

• PARTITA(Stanza) ⊆ STANZADIGIOCO(NomeStanza)

PARTECIPA(Giocatore,Partita, NumTurno)

• PARTECIPA(Giocatore) ⊆ GIOCATORE(GamerTag)

• PARTECIPA(Partita) ⊆ PARTITA(CodicePartita)

TURNO(idTurno, Partita, Giocatore, StatoTurno, Timer*)

• TURNO(Partita) ⊆ Partita(CodicePartita)

• TURNO(Giocatore) ⊆ GIOCATORE(GamerTag)

ISTANZADITERRITORIO(Territorio,Partita,Proprietario, NumCarriPosizionati)

• ISTANZADITERRITORIO(Territorio) ⊆ TERRITORIO(NomeTerritorio)

• ISTANZADITERRITORIO(Partita) ⊆ Partita(CodicePartita)

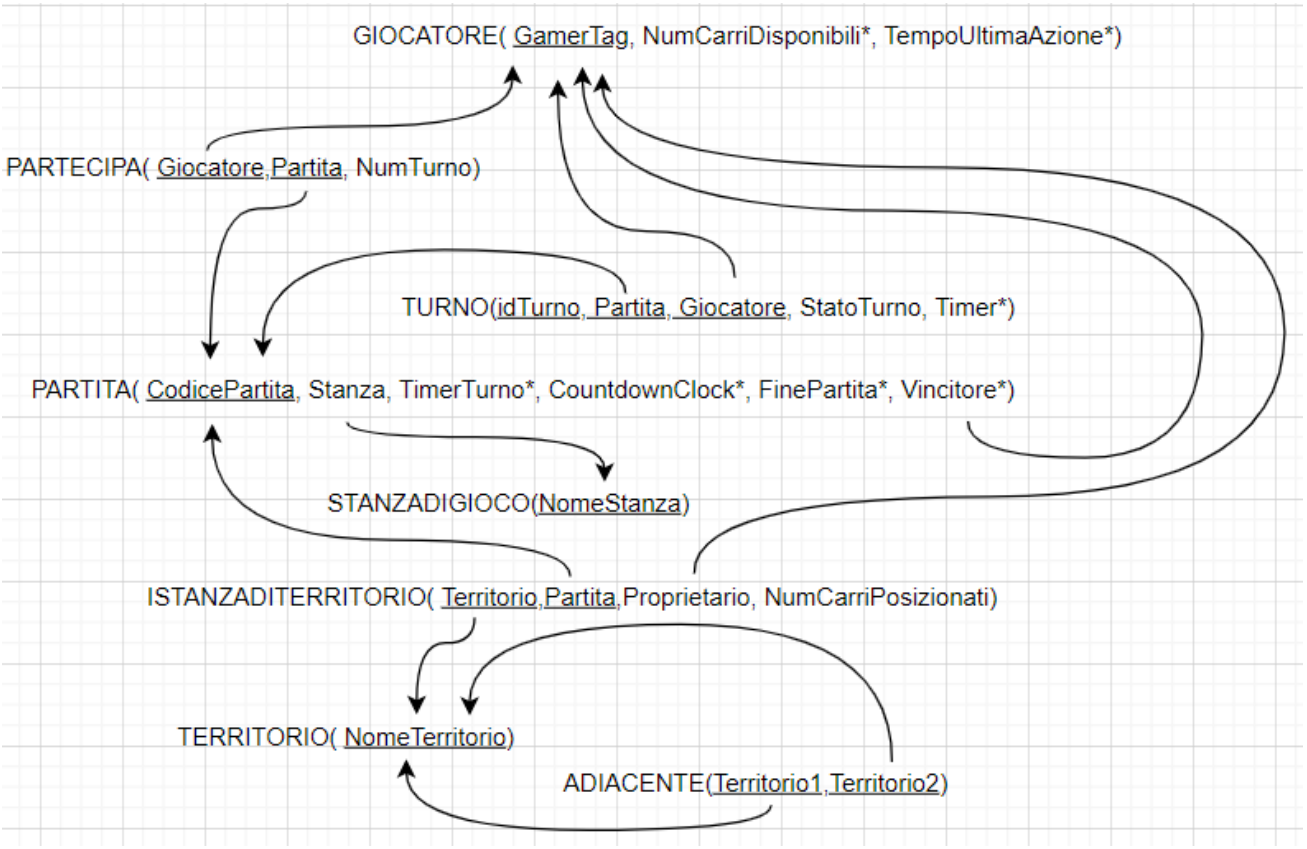
• ISTANZADITERRITORIO(Proprietario) ⊆ GIOCATORE(GamerTag)

ADIACENTE(Territorio1,Territorio2)

• ADIACENTE(Territorio1) ⊆ TERRITORIO(NomeTerritorio)

• ADIACENTE(Territorio2) ⊆ TERRITORIO(NomeTerritorio)

NB: Sono indicati con '*' gli attributi che possono essere NULL.



Normalizzazione del modello relazionale

1NF:

Non avendo attributi composti, essendo sempre presente una chiave primaria (garantendo l'assenza di tuple duplicate) e non essendoci gruppi di attributi che si ripetono si conclude che il precedente modello relazionale si trova già in 1NF.

2NF:

Non essendoci inoltre chiavi contenenti chiavi più piccole (e quindi non sono superchiavi bensì chiavi minimali) concludiamo che il modello si trova già anche in 2NF.

3NF:

Per tutte le relazioni del modello si verifica che per ogni dipendenza funzionale $X \rightarrow A$ vale che X è superchiave della relazione oppure A appartiene ad almeno una chiave. Viene quindi raggiunta anche la 3NF.

5) Progettazione fisica

Utenti e privilegi

Di seguito verranno elencati i diversi tipi di utente previsti per la base di dati. In particolare ognuno associato ad un omonimo ruolo nel dbms con privilegi di tipo 'execute' su (in elenco) determinate stored procedures ognuna delle quali opera su determinate tabelle (in sotto-elenco) per rendere il sistema il più possibile fruibile ma al contempo facendo sì che ogni tipo di utente è autorizzato esclusivamente ad eseguire lo stretto necessario all'interno della base di dati. Ad esempio nel caso del primo utente 'Login' la scelta progettuale è stata voluta con l'obiettivo di impedire ad un utente non autenticato di invocare una qualsiasi procedura che non sia quella di login e di registrazione impedendo quindi inizialmente l'accesso a tutte le risorse del sistema.

Login:

- 'login'
    - 'Utenti'
- 'registra_nuovo_giocatore'
    - 'Utenti'
    - 'Giocatore'

Moderatore:

- 'report_moderatori'
    - 'StanzaDiGioco'
    - 'Partita'
    - 'Partecipa'
    - 'Giocatore'
- 'crea_nuova_stanza'
    - 'StanzaDiGioco'
    - 'Partita' (dovendo creare la prima partita in quella stanza e permettendo così ai giocatori di entrare nella stanza e partecipare alla partita al suo interno.)
- 'registra_nuovo_moderatore'
    - 'Utenti'

Giocatore:

- ‘aggiungi_partecipante’
    - ‘Partecipa’
    - ‘Partita’ (per recuperare il codice della partita attualmente in esecuzione o in attesa associata a quella stanza)
- ‘attacca_territorio’
    - ‘Partita’
    - ‘Turno’
    - ‘IstanzaDiTerritorio’
    - ‘Adiacente’
    - ‘Partecipa’
- ‘ottieni_lista_stanze_disponibili’
    - ‘StanzaDiGioco’
    - ‘Partita’
    - ‘Partecipa’
- ‘posiziona_carriArmati_su_territorio’
    - ‘Partita’
    - ‘Turno’
    - ‘IstanzaDiTerritorio’
    - ‘Giocatore’
    - ‘Partecipa’
- ‘rimuovi_partecipante’
    - ‘Partita’
    - ‘Partecipa’
    - ‘Turno’
    - ‘IstanzaDiTerritorio’
    - ‘Giocatore’
- ‘sposta_carriArmati_tra_territori’
    - ‘Partita’
    - ‘Turno’
    - ‘IstanzaDiTerritorio’
    - ‘Adiacente’

    - ‘Partecipa’
- ‘vedi_stato_di_gioco’
    - ‘Partita’
    - ‘Turno’
    - ‘IstanzaDiTerritorio’
    - ‘Giocatore’
    - ‘Partecipa’
- ‘vedi_storico_partite’
    - ‘Partita’
    - ‘Partecipa’
- ‘salta_turno’
    - ‘Partita’
    - ‘Turno’
    - ‘Partecipa’
- ‘mostra_adiacenze_territorio’
    - ‘Adiacente’

Strutture di memorizzazione

Tabella <GIOCATORE>		
Colonna	**Tipo di dato**	**Attributi[2]**
GamerTag	VARCHAR(32)	PK, NN
NumCarriDisponibili	INT	
TempoUltimaAzione	TIMESTAMP	

Tabella <PARTITA>		
Colonna	**Tipo di dato**	**Attributi**
CodicePartita	INT	PK, NN, AI
Stanza	VARCHAR(45)	NN
StatoPartita	ENUM(‘wait’, ‘exec’, ‘end’)	NN
CountdownClock	TIME	
Vincitore	VARCHAR(32)	

[2] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

Tabella <STANZA DI GIOCO>		
Colonna	**Tipo di dato**	**Attributi**
NomeStanza	VARCHAR(45)	PK,NN

Tabella <TERRITORIO>		
Colonna	**Tipo di dato**	**Attributi**
NomeTerritorio	VARCHAR(45)	PK, NN

Tabella <ISTANZADITERRITORIO>		
Colonna	**Tipo di dato**	**Attributi**
Territorio	VARCHAR(45)	PK, NN
Partita	INT	PK, NN
Proprietario	VARCHAR(32)	NN
NumCarriPosizionati	INT	NN

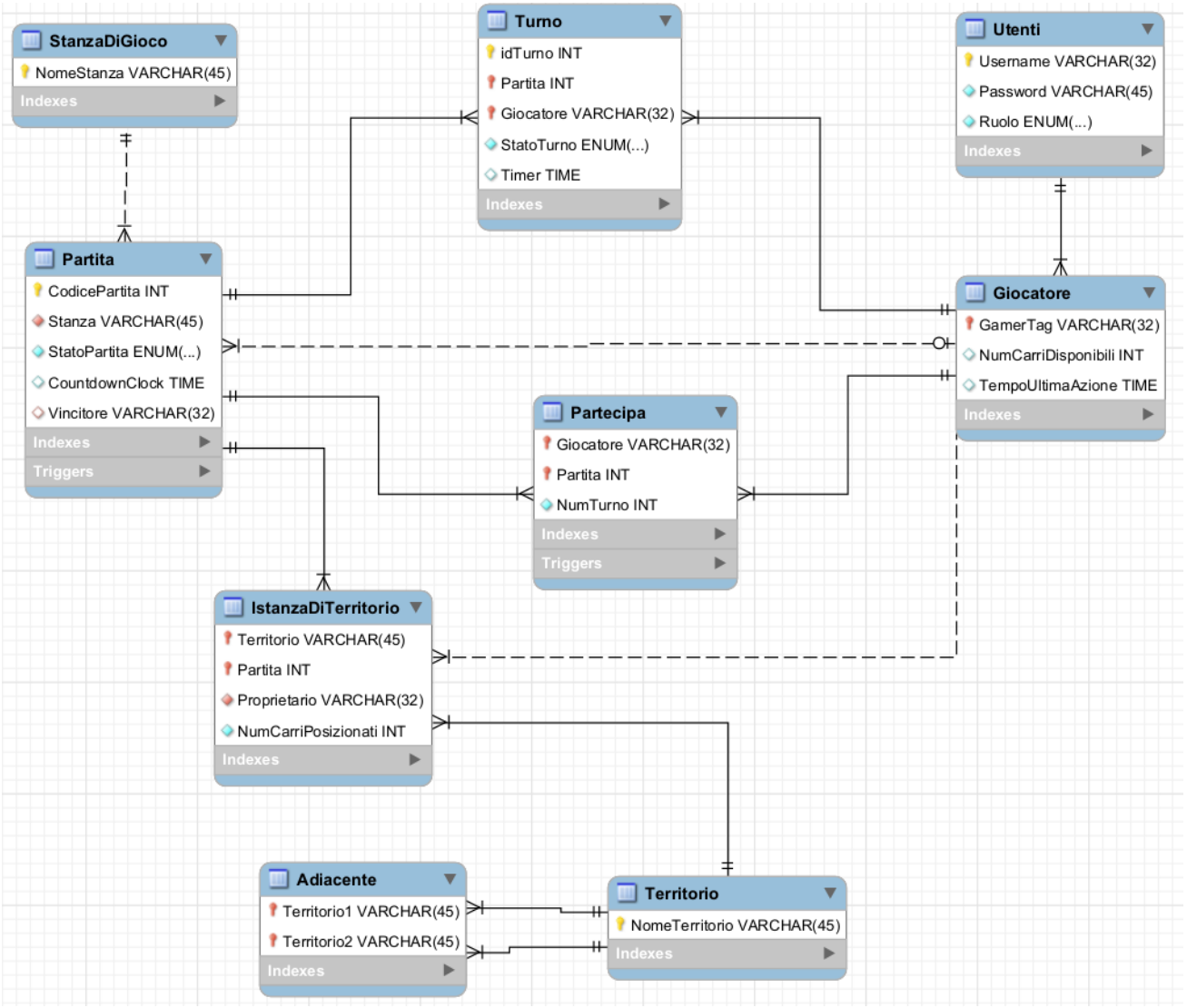
Tabella <PARTECIPA>		
Colonna	**Tipo di dato**	**Attributi**
Giocatore	VARCHAR(32)	PK, NN
Partita	INT	PK, NN
NumTurno	INT	NN

Tabella <TURNO>		
Colonna	**Tipo di dato**	**Attributi**
idTurno	INT	PK, NN, AI
Partita	INT	PK, NN
Giocatore	VARCHAR(32)	PK, NN
StatoTurno	ENUM(‘noaction’,’action’,’end’)	NN
Timer	TIME	

Tabella <ADIACENTE>		
Colonna	**Tipo di dato**	**Attributi**
Territorio1	VARCHAR(45)	PK, NN
Territorio2	VARCHAR(45)	PK, NN

Tabella <UTENTI>		
Colonna	**Tipo di dato**	**Attributi**
Username	VARCHAR(32)	PK, NN
Password	VARCHAR(45)	NN
Ruolo	ENUM(‘moderatore’,’giocatore’)	NN

Di seguito il diagramma delle tabelle estratto da mysql workbench.



Indici

All'interno di ciascuna tabella è stato creato un indice PRIMARY per identificare univocamente le tabelle mentre gli indici IDX permettono al dbms il controllo delle varie foreign keys.

Tabella <ADIACENTE>	
Indice <PRIMARY>	**Tipo[3]:**
Territorio1, Territorio2	PR
Indice <Territorio_Adiacente_2_idx>	**Tipo:**
Territorio2	IDX

Tabella <GIOCATORE>

[3] IDX = index, UQ = unique, FT = full text, PR = primary.

Indice <PRIMARY>	**Tipo:**
GamerTag	PR

Tabella <ISTANZADITERRITORIO>	
Indice <PRIMARY>	**Tipo:**
Territorio, Partita	PR
Indice <Partita_IstanzaDiTerritorio_idx>	**Tipo:**
Partita	IDX
Indice <Proprietario_IstanzaDiTerritorio_idx>	**Tipo:**
Proprietario	IDX

Tabella <PARTECIPA>	
Indice <PRIMARY>	**Tipo:**
Giocatore, Partita	PR
Indice <Partita_Partecipa_idx>	**Tipo:**
Partita	IDX
Indice <Partita_NumTurno_uq>	**Tipo:**
Partita, NumTurno	UQ

in questo caso l'indice 'UQ' viene inserito per evitare inserimenti di più partecipazioni per una data partita con stesso numero di turno.

Tabella <PARTITA>	
Indice <PRIMARY>	**Tipo:**
CodicePartita	PR
Indice <VINCITORE_PARTITA_idx>	**Tipo:**
Partita	IDX
Indice <Stanza_Partita_idx>	**Tipo:**
Proprietario	IDX

Tabella <STANZADIGIOCO>	
Indice <PRIMARY>	**Tipo:**
NomeStanza	PR

Tabella <TERRITORIO>	
Indice <PRIMARY>	**Tipo:**
NomeTerritorio	PR

Tabella <TURNO>	
Indice <PRIMARY>	**Tipo:**
idTurno, Partita, Giocatore	PR
Indice <Partita_Turno_idx>	**Tipo:**
Partita	IDX
Indice <Giocatore_Turno_idx>	**Tipo:**

Giocatore	IDX

Tabella <UTENTI>	
Indice <PRIMARY>	**Tipo:**
Username	PR

Trigger

1. Trigger after update su partita per far sì che, una volta che una partita passa in stato di 'exec' vengono invocate le procedure 'gioca_partita' e 'inizia_turno' tali da fare il setup della partita (creazione e assegnazione istanze di territori ai partecipanti, reimpostazione carri disponibili per i partecipanti) e da permettere al giocatore con numero di turno 1 di poter svolgere una delle azioni permesse durante il turno

```
CREATE DEFINER = CURRENT_USER TRIGGER `risikoDB`.`Check_PartitaIniziata` AFTER UPDATE ON
`Partita` FOR EACH ROW
BEGIN
	declare var_primoGiocatore VARCHAR(32);

	IF NEW.`StatoPartita` = 'exec' THEN
		CALL gioca_partita(OLD.`CodicePartita`);

		SELECT `Giocatore` FROM `Partecipa`
	    WHERE `Partita` = OLD.`CodicePartita`
	    AND `NumTurno` = 1
	    INTO var_primoGiocatore;

	    CALL inizia_turno(var_primoGiocatore, OLD.`CodicePartita`);
	END IF;
END
```

2. Trigger before insert su 'Partecipa' per assicurarsi che è stato raggiunto il numero massimo di giocatori e che il numero di turno assegnato sia compreso tra 1 e 6 (come da regola aziendale)

```
CREATE DEFINER = CURRENT_USER TRIGGER `risikoDB`.`Check_numeroTurnoValido` BEFORE INSERT ON
`Partecipa` FOR EACH ROW
BEGIN
	-- controllo se è già stato raggiunto il numero massimo di giocatori e se il numero di
turno rispetta la regola aziendale
    IF NEW.`NumTurno` > 6 OR NEW.`NumTurno` < 1 THEN
		SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Il Giocatore non può essere
ammesso.';
    END IF;
END
```

Eventi

All'interno del sistema sono stati implementati ben 3 eventi temporizzati attivati all'istanziazione del database. I primi due con l'obiettivo di implementare i timer citati nella specifica e il terzo per riavviare una nuova partita all'interno di una stanza dopo qualche secondo che l'ultima partita è terminata.

- 'timer_inizio_partita'
- 'timer_cambio_turno'
- 'riavvio_partita_automatico'

Di seguito il codice SQL per l'implementazione degli eventi appena citati.

```
set global event_scheduler = ON;
DELIMITER //
CREATE EVENT IF NOT EXISTS `risikoDB`.`timer_inizio_partita`
ON SCHEDULE EVERY 2 SECOND STARTS CURRENT_TIME ON COMPLETION PRESERVE
COMMENT 'implementazione countdown inizio partita'
DO
BEGIN
	UPDATE `Partita` SET `StatoPartita` = 'exec'
	WHERE `CountdownClock` < current_time() - INTERVAL 2 MINUTE -- sono passati 2 minuti
	AND `StatoPartita` = 'wait'; -- la partita era in stato di attesa
END//

DELIMITER //
CREATE EVENT IF NOT EXISTS `risikoDB`.`timer_cambio_turno`
ON SCHEDULE EVERY 10 SECOND STARTS CURRENT_TIME ON COMPLETION PRESERVE
COMMENT 'implementazione timer cambio turno'
DO
BEGIN
    declare var_gamerTag VARCHAR(32);
    declare var_codicePartita INT;
    declare var_statoTurno ENUM('noaction','action','end');

    SELECT `Partita`,`Giocatore`,`StatoTurno` FROM `Turno`
    WHERE `Timer` < current_time() - INTERVAL 3 MINUTE -- sono passati 3 minuti
    AND  `StatoTurno` <> 'end' -- il turno è attualmente in corso
    INTO var_codicePartita, var_gamerTag, var_statoTurno;

    IF var_statoTurno = 'noaction' THEN
              CALL cambio_turno(var_codicePartita, var_gamerTag);
    END IF;

END//

DELIMITER //
CREATE EVENT IF NOT EXISTS `risikoDB`.`riavvio_partita_automatico`
ON SCHEDULE EVERY 30 SECOND STARTS CURRENT_TIME ON COMPLETION PRESERVE
COMMENT 'implementazione riavvio partita in automatico'
```

```
DO
BEGIN
    declare done int default false;

    declare var_stato ENUM('wait','exec','end');
    declare var_stanza VARCHAR(45);

    declare cur_partite cursor for
    SELECT `StatoPartita`,`Stanza` FROM `Partita` AS `P1`
    WHERE `CodicePartita` = (SELECT max(`CodicePartita`)
                                FROM `Partita` AS `P2`
                            WHERE `P2`.`Stanza` = `P1`.`Stanza`);

    OPEN cur_partite;
    restart_loop: LOOP
         FETCH cur_partite INTO var_stato,var_stanza;
        IF var_stato = 'end' THEN
              INSERT INTO `Partita` (`Stanza`) -- ricrea una nuova partita in quella stanza
              VALUES(var_stanza) ;
         END IF;
    END LOOP;
    CLOSE cur_partite;
END//
```

Viste

Non è stata inserita alcuna vista.

Stored Procedures e transazioni

- 'login'
    - Procedura che permette l'autenticazione nella base di dati, in particolare legge nella tabella 'Utenti' e una volta trovato un riscontro restituisce il ruolo rispettivo e autentica l'utente, altrimenti rifiuta l'accesso.

```
CREATE PROCEDURE `login`(
       IN var_username VARCHAR(32),
       IN var_password VARCHAR(45),
    OUT var_ruolo INT)
BEGIN
    declare var_enumRuolo ENUM('moderatore','giocatore') ;

    SELECT `Ruolo` FROM `Utenti`
    WHERE `Username` = var_username AND `Password` = SHA1(var_password)
    INTO var_enumRuolo ;

    IF var_enumRuolo = 'moderatore' THEN
          SET var_ruolo = 0 ;
```

```
    ELSEIF var_enumRuolo = 'giocatore' THEN
          SET var_ruolo = 1 ;
    ELSE
          SET var_ruolo = 2 ;
    END IF ;
END
```

- 'registra_nuovo_giocatore'
    - Procedura che si occupa della registrazione di nuovi giocatori.

      NB:] viene inserita una transazione per far sì che i due inserimenti di utente e giocatore avvengano "o entrambi o nessuno dei due".

```
CREATE PROCEDURE `registra_nuovo_giocatore` (
IN var_gamerTag VARCHAR(32),
IN var_password VARCHAR(45))
BEGIN
    declare exit handler for sqlexception
    begin
        rollback; ## annullo la transazione
        resignal; ## segnalo al chiamante
    end;

    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    START TRANSACTION ;
          INSERT INTO `Utenti` (`Username`, `Password`, `Ruolo`)
          VALUES(var_gamerTag, SHA1(var_password), 'giocatore');

          INSERT INTO `Giocatore` (`GamerTag`, `NumCarriDisponibili`, `TempoUltimaAzione`)
                    VALUES (var_gamerTag, NULL, NULL) ;
    COMMIT ;
END
```

- 'report_moderatori'
    - Procedura che permette ad un moderatore di ricavare le informazioni necessarie per la visualizzazione del report dei moderatori come da specifica.

      NB:] viene impostato un comportamento transazionale con livello di isolamento REPEATABLE READ per avere le varie select consistenti tra loro (ricordiamo infatti che con questo livello di isolamento le letture vengono fatte su uno snapshot stabilito alla prima lettura).

```
CREATE PROCEDURE `report_moderatori` ()
BEGIN
    declare exit handler for sqlexception
    begin
        rollback; ## annullo la transazione
        resignal; ## segnalo al client
```

```
    end;

    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ ;
    SET TRANSACTION READ ONLY;
    START TRANSACTION ;

    -- quante stanze hanno attualmente partite in corso
    SELECT COUNT(*) AS `StanzeAttive` FROM `StanzaDiGioco`
    JOIN `Partita` ON `Partita`.`Stanza` = `StanzaDiGioco`.`NomeStanza`
    WHERE `StatoPartita` = 'exec';

    -- quanti sono i giocatori in ciascuna di queste partite (esiste solo una partecipazione per
un giocatore in quella partita dati i vincoli di chiave)
    SELECT `Stanza`,`CodicePartita`,COUNT(*) AS `Partecipanti` FROM `Partita`
    JOIN `Partecipa` ON `Partita`.`CodicePartita` = `Partecipa`.`Partita`
    WHERE `StatoPartita` = 'exec'
    GROUP BY `CodicePartita`;

    -- quanti sono i giocatori che hanno effettuato almeno un'azione negli ultimi 15 minuti
    -- che non sono all'interno di alcuna stanza di gioco.
    SELECT COUNT(*) AS `GiocatoriAttivi` FROM `Giocatore`
    WHERE `TempoUltimaAzione` >= current_timestamp() - INTERVAL 15 MINUTE
    AND controlla_giocatore_in_partita(`Giocatore`.`GamerTag`) = 0;

    COMMIT;
END
```

- ‘crea_nuova_stanza’
    - Procedura che permette ad un moderatore di creare una nuova stanza e una partita con nome di stanza corrispondente (permettendo così ai giocatori di entrare nella stanza e partecipare alla partita al suo interno.)

      NB:] viene inserita una transazione per far sì che i due inserimenti di stanza di gioco e partita avvengano “o entrambi o nessuno dei due”, non avrebbe senso che una stanza esistesse senza una partita al suo interno.

```
CREATE PROCEDURE `crea_nuova_stanza` (
IN var_nome_stanza VARCHAR(45))
BEGIN
    declare exit handler for sqlexception
    begin
        rollback; ## annullo la transazione
        resignal; ## segnalo al chiamante
    end;

    SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
    START TRANSACTION ;
          INSERT INTO `StanzaDiGioco` (`NomeStanza`)
              VALUES(var_nome_stanza) ;
```

```
          INSERT INTO `Partita` (`Stanza`)  -- creo fin da subito la prima partita contenuta in
tale stanza (lo stato di default è 'wait')
              VALUES(var_nome_stanza) ;
    COMMIT;
END
```

- 'registra_nuovo_moderatore'
    - Procedura che permette ad un moderatore, in quanto utente autorizzato, di creare nuove credenziali per un utenza con privilegi di moderatore.

```
CREATE PROCEDURE `registra_nuovo_moderatore` (
IN var_username VARCHAR(32),
IN var_password VARCHAR(45))
BEGIN
       INSERT INTO `Utenti` (`Username`, `Password`, `Ruolo`)
              VALUES(var_username, SHA1(var_password), 'moderatore') ;
END
```

- 'aggiungi_partecipante'
    - Procedura che permette ad un giocatore, che non sta partecipando ad un'altra partita, di entrare in una stanza (partecipare alla partita al suo interno).

      NB:] READ COMMITTED per evitare letture sporche. La presenza del trigger before insert in Partecipa e dell'indice unique sulla coppia Partita,numeroTurno evita di andare a mettere livelli di isolamento eccessivamente alti.

```
CREATE PROCEDURE `aggiungi_partecipante` (
IN var_gamerTag VARCHAR(32),
IN var_nomeStanza VARCHAR(45),
OUT var_codicePartita INT)
BEGIN
    declare var_numPartecipanti INT;

    declare exit handler for sqlexception
    begin
        rollback; ## annullo la transazione
        resignal; ## segnalo al chiamante
    end;

    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    START TRANSACTION ;
         -- controllo se il giocatore è già in un'altra partita
         IF controlla_giocatore_in_partita(var_gamerTag) > 0 THEN
              SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = 'Giocatore già in altra partita!
impossibile partecipare.';
         END IF;
```

```
        -- ricavo il codice della partita che è in quella stanza
        SELECT `CodicePartita` FROM `Partita`
        WHERE `StatoPartita` <> 'end'
        AND `Partita`.`Stanza`= var_nomeStanza
        INTO var_codicePartita;

        -- controllo se la stanza è stata trovata
        IF var_codicePartita IS NULL THEN
             SIGNAL SQLSTATE '45002' SET MESSAGE_TEXT = 'Stanza non disponibile! Riprovare o
consultare la lista delle stanze disponibili';
        END IF;

       -- controllo se la partita è già iniziata
        IF stato_partita(var_codicePartita) <> 'wait' THEN
             SIGNAL SQLSTATE '45003' SET MESSAGE_TEXT = 'Partita già iniziata! impossibile
partecipare.';
        END IF;

        SET var_numPartecipanti = conta_partecipanti(var_codicePartita);

        -- assegno il numero di turno nell'ordine di arrivo e inserisco la partecipazione
        INSERT INTO `Partecipa` (`Giocatore`,`Partita`,`NumTurno`)
      VALUES (var_gamerTag, var_codicePartita, var_numPartecipanti + 1);

        IF var_numPartecipanti + 1 = 3 THEN
             UPDATE `Partita` SET `CountdownClock`= current_time() -- segna il tempo corrente
per il countdown
             WHERE `CodicePartita` = var_codicePartita;
        END IF;
   COMMIT;
END
```

- 'attacca_territorio'
    - Procedura che permette di eseguire una delle tre azioni previste durante il turno di un giocatore in partita. In particolare l'azione di attacco viene effettuata come da specifica da un territorio del giocatore che attacca verso un territorio di proprietà del giocatore che difende.

```
CREATE PROCEDURE `attacca_territorio` (
IN var_giocatoreAtt VARCHAR(32),
IN var_numCarriAtt INT,
IN var_terrAtt VARCHAR(45),
IN var_terrDif VARCHAR(45),
OUT var_numCPersiAtt INT,
OUT var_numCPersiDif INT,
OUT var_conquistato INT,
OUT var_vittoriaPartita INT)
BEGIN
    declare var_codicePartita INT;
```

```
    declare var_flag INT;
    declare var_numCarriCheck INT;
    declare var_statoPartita ENUM('wait','exec','end');
    declare var_numCarriDif INT;
    declare var_numDadiAtt INT;
    declare var_numDadiDif INT;
    declare var_numCarriPersiAtt INT DEFAULT 0;
    declare var_numCarriPersiDif INT DEFAULT 0;
    SET var_numCPersiAtt = 0;
    SET var_numCPersiDif = 0;
    SET var_conquistato = 0;
    SET var_vittoriaPartita = 0;

    SET var_codicePartita = recupera_partita(var_giocatoreAtt);
    CALL controlli_prima_di_azione(var_giocatoreAtt, var_codicePartita);
    CALL controllo_esistenza_territorio(var_terrAtt);
    CALL controllo_esistenza_territorio(var_terrDif);

    -- controllo che lo stato di attacco sia di proprietà del giocatore attaccante e che quello
di difesa non lo sia
    SELECT COUNT(*) FROM `IstanzaDiTerritorio`
    WHERE `Partita` = var_codicePartita AND `Proprietario` = var_giocatoreAtt AND `Territorio` =
var_terrAtt
    INTO var_flag;

    IF var_flag <> 1 THEN
       SIGNAL SQLSTATE '45004' SET MESSAGE_TEXT = 'Il territorio da cui parte l\'attacco non è
di proprietà dell\'attaccante: Attacco non disponibile.';
    END IF;

    SELECT COUNT(*) FROM `IstanzaDiTerritorio`
    WHERE `Partita` = var_codicePartita AND `Proprietario` = var_giocatoreAtt AND `Territorio` =
var_terrDif
    INTO var_flag;

    IF var_flag <> 0 THEN
           SIGNAL SQLSTATE '45005' SET MESSAGE_TEXT = 'Non è possibile attaccare territori di
proprietà dell\'attaccante: Attacco non disponibile.';
    END IF;

    -- controllo che il numero di carri dichiarati sia presente (con scarto di un carro armato
che deve rimanere per forza nel territorio)
    IF var_numCarriAtt > 3 OR var_numCarriAtt < 1 THEN
           SIGNAL SQLSTATE '45006' SET MESSAGE_TEXT = 'Non è possibile attaccare con un numero
di carri armati che non sia compreso tra 1(verrà lanciato 1 dado) e 3(verranno lanciati tutti e
3 i dadi)';
    END IF;

    SELECT `NumCarriPosizionati` FROM `IstanzaDiTerritorio`
    WHERE `Partita` = var_codicePartita AND `Proprietario` = var_giocatoreAtt AND `Territorio` =
var_terrAtt
    INTO var_numCarriCheck;
```

```
    IF var_numCarriAtt >= var_numCarriCheck THEN
           SIGNAL SQLSTATE '45007' SET MESSAGE_TEXT = 'Sul territorio deve essere presente
almeno un carro armato in più di quelli schierati in attacco: Attacco non disponibile.';
    END IF;

    -- controllo adiacenza territori
    SELECT COUNT(*) FROM `Adiacente`
    WHERE (`Territorio1` = var_terrAtt AND `Territorio2` = var_terrDif) OR (`Territorio1` =
var_terrDif AND `Territorio2` = var_terrAtt)
    INTO var_flag;

    IF var_flag <> 1 THEN
           SIGNAL SQLSTATE '45008' SET MESSAGE_TEXT = 'Non è possibile attaccare un territorio
non adiacente: Attacco non disponibile.';
    END IF;

    -- calcolo numero dadi disponibili per attacco e difesa ed eseguo il lancio dei dadi
    SET var_numDadiAtt = var_numCarriAtt;

    SELECT `NumCarriPosizionati` FROM `IstanzaDiTerritorio`
    WHERE `Partita` = var_codicePartita AND `Territorio` = var_terrDif
    INTO var_numCarriCheck; -- numero di carri presenti sul territorio del giocatore che si
difende

    -- in un contesto reale il difensore dovrebbe dichiarare con quanti carri si vuole difendere
    -- per semplicità in questo caso qual'ora ne disponga si difende con lo stesso numero di
dadi dell'attaccante
    -- altrimenti si calcola il numero massimo di dadi possibile sulla base dei carri
disponibili
    IF var_numCarriCheck >= var_numCarriAtt THEN
          SET var_numDadiDif =  var_numDadiAtt;
    ELSE
           SET var_numDadiDif = var_numCarriCheck;
    END IF;

    CALL risultato_lancio_dadi(var_numDadiAtt, var_numDadiDif, var_numCarriPersiAtt,
var_numCarriPersiDif);
    SET var_numCPersiAtt = var_numCarriPersiAtt;
    SET var_numCPersiDif = var_numCarriPersiDif;

    IF var_numCarriPersiAtt <> 0 THEN
            UPDATE `IstanzaDiTerritorio` SET `NumCarriPosizionati` = `NumCarriPosizionati` -
var_numCarriPersiAtt
            WHERE `Partita` = var_codicePartita AND `Proprietario` = var_giocatoreAtt
            AND `Territorio` = var_terrAtt;
    END IF;

    IF var_numCarriPersiDif <> 0 THEN
           UPDATE `IstanzaDiTerritorio` SET `NumCarriPosizionati` = `NumCarriPosizionati` -
var_numCarriPersiDif
           WHERE `Partita` = var_codicePartita AND `Territorio` = var_terrDif;

           SELECT `NumCarriPosizionati` FROM `IstanzaDiTerritorio`
```

```
            WHERE `Partita` = var_codicePartita
            AND `Territorio` = var_terrDif
            INTO var_numCarriCheck;

            IF var_numCarriCheck = 0 THEN
                UPDATE `IstanzaDiTerritorio` SET `NumCarriPosizionati` = `NumCarriPosizionati` -
(var_numCarriAtt - var_numCarriPersiAtt) -- tolgo anche il resto dei carri schierati per
riposizionarli sul nuovo territorio
                WHERE `Partita` = var_codicePartita AND `Proprietario` = var_giocatoreAtt
                 AND `Territorio` = var_terrAtt;

                UPDATE `IstanzaDiTerritorio` SET `Proprietario` = var_giocatoreAtt,
`NumCarriPosizionati` = var_numCarriAtt - var_numCarriPersiAtt
                WHERE `Partita` = var_codicePartita AND `Territorio` = var_terrDif;

                 SET var_conquistato = 1;
               END IF;

        -- se l'attaccante ha tutti i territori diventa il vincitore
              IF calcola_possedimenti(var_giocatoreAtt, var_codicePartita) = 42 THEN
                  UPDATE `Partita` SET `Vincitore` = var_giocatoreAtt, `StatoPartita` = 'end'
                  WHERE `CodicePartita` = var_codicePartita;

                 SET var_vittoriaPartita = 1;
              END IF;
       END IF;

       CALL segna_ultimo_turno_azione(var_giocatoreAtt, var_codicePartita);
       CALL cambio_turno(var_codicePartita, var_giocatoreAtt);
END
```

- ‘ottieni_lista_stanze_disponibili’
    - Procedura che permette al giocatore, all’esterno di una stanza di gioco, di visualizzare quali sono le stanze che attualmente ammettono giocatori e in cui quindi, attraverso la procedura ‘aggiungi_partecipante’, il giocatore può entrare e partecipare alla partita al suo interno. In particolare attraverso la procedura si possono vedere il codice della partita attualmente in attesa e il numero di partecipanti attualmente nella stanza di gioco.

      NB:] READ COMMITTED dato che non voglio dirty reads.

```
CREATE PROCEDURE `ottieni_lista_stanze_disponibili` ()
BEGIN
    declare exit handler for sqlexception
    begin
        rollback; ## annullo la transazione
        resignal; ## segnalo al chiamante
    end;
```

```
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    SET TRANSACTION READ ONLY;
    START TRANSACTION;
         SELECT `NomeStanza`, `CodicePartita`, COUNT(`Giocatore`) AS `Partecipanti` FROM
`StanzaDiGioco`
         JOIN `Partita` ON `Partita`.`Stanza` = `StanzaDiGioco`.`NomeStanza`
        LEFT JOIN `Partecipa` ON `Partita`.`CodicePartita` = `Partecipa`.`Partita`
        WHERE `StatoPartita` = 'wait'
        GROUP BY `CodicePartita`
        HAVING COUNT(`Giocatore`) < 6;
    COMMIT;
END
```

- 'posiziona_carriArmati_su_territorio'
    - Insieme alle procedure di attacco e di spostamento questa procedura permette di eseguire una delle azioni possibili durante il turno di un giocatore. In particolare l'azione di posizionamento permette di aumentare il numero di carri armati posizionati su quella data istanza di territorio previa verifica della disponibilità di tale numero di carri armati per quel giocatore.

```
CREATE PROCEDURE `posiziona_carriArmati_su_territorio` (
IN var_gamerTag VARCHAR(32),
IN var_numCarriDaPosiz INT,
IN var_nomeTerr VARCHAR(45))
BEGIN
    declare var_codicePartita INT;
    declare var_giocatoreCheck VARCHAR(32);
    declare var_numCarriCheck INT;

    SET var_codicePartita = recupera_partita(var_gamerTag);
    CALL controlli_prima_di_azione(var_gamerTag, var_codicePartita);
    CALL controllo_esistenza_territorio(var_nomeTerr);

    -- controllo proprietà del territorio
    SELECT `Proprietario` FROM `IstanzaDiTerritorio`
    WHERE `Partita` = var_codicePartita
    AND `Territorio` = var_nomeTerr
    INTO var_giocatoreCheck;

    IF var_gamerTag <> var_giocatoreCheck THEN
          SIGNAL SQLSTATE '45014' SET MESSAGE_TEXT = 'Il territorio selezionato non è di
proprietà del giocatore! Posizionamento carri armati non disponibile.';
    END IF;

    -- controllo disponibilità dei carri armati indicati
    SELECT `NumCarriDisponibili` FROM `Giocatore`
    WHERE `GamerTag` = var_gamerTag
    INTO var_numCarriCheck;
```

```
    IF var_numCarriDaPosiz > var_numCarriCheck THEN
            SIGNAL SQLSTATE '45015' SET MESSAGE_TEXT = 'Carri armati disponibili inferiori a
quelli selezionati! Posizionamento carri armati non disponibile.';
    END IF;

    UPDATE `IstanzaDiTerritorio` SET `NumCarriPosizionati` = `NumCarriPosizionati` +
var_numCarriDaPosiz
    WHERE `Partita` = var_codicePartita
    AND `Territorio` = var_nomeTerr;

    UPDATE `Giocatore` SET `NumCarriDisponibili` = `NumCarriDisponibili` - var_numCarriDaPosiz
    WHERE `GamerTag` = var_gamerTag;

    CALL segna_ultimo_turno_azione(var_gamerTag, var_codicePartita);

    CALL cambio_turno(var_codicePartita, var_gamerTag);
END
```

- 'rimuovi_partecipante'
    - Procedura che permette ad un giocatore di abbandonare la partita e si occupa di redistribuire i numeri di turno ai giocatori rimanenti. In particolare se questo avviene quando la partita è in stato di 'exec' verrà anche fatta una redistribuzione dei territori del giocatore, se ancora ne possedeva .

      NB:] REPEATABLE READ ci permette di avere dati coerenti fra loro durante tutta la transazione

```
CREATE PROCEDURE `rimuovi_partecipante` (
IN var_gamerTag VARCHAR(32))
BEGIN
    declare var_codicePartita INT;
    declare var_lastPlayer VARCHAR(32);
    declare var_numPartecipanti INT;
    declare var_statoPartita ENUM('wait','exec','end');

    declare exit handler for sqlexception
    begin
        rollback; ## annullo la transazione
        resignal; ## segnalo al chiamante
    end;

    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
    START TRANSACTION ;
        SET var_codicePartita = recupera_partita(var_gamerTag);
        SET var_statoPartita = stato_partita(var_codicePartita);

        IF var_statoPartita <> 'end' THEN
                IF var_statoPartita = 'exec' THEN
```

```
                    CALL controlla_proprietario_turno(var_gamerTag, var_codicePartita);
                    CALL cambio_turno(var_codicePartita, var_gamerTag);
                END IF;

                DELETE FROM `Partecipa` WHERE  `Giocatore` = var_gamerTag AND `Partita`=
var_codicePartita;

                SET var_numPartecipanti = conta_partecipanti(var_codicePartita);

               IF var_numPartecipanti < 3 AND var_statoPartita = 'wait' THEN
                      UPDATE `Partita`
                      SET `CountdownClock` = NULL
                           -- ripristino il timer di countdown
                      WHERE `Partita`.`CodicePartita` = var_codicePartita;
                END IF;

                IF var_statoPartita = 'exec' AND var_numPartecipanti = 1 THEN -- termino
partita e decreto vincitore nel caso di ultimo giocatore rimasto in partita
                      SELECT `Giocatore` FROM `Partecipa`
                      WHERE `Partecipa`.`Partita` = var_codicePartita
                      INTO var_lastPlayer;

                      UPDATE `Partita` SET `Vincitore` = var_lastPlayer, `StatoPartita` =
'end'
                      WHERE `Partita`.`CodicePartita` = var_codicePartita;
                END IF;

                CALL riassegna_turni(var_codicePartita, var_gamerTag); -- si occuperà di
redistribuire anche i territori in caso la partita era in corso
                CALL reset_tempo_ultima_azione(var_gamerTag);
        END IF;
    COMMIT;
END
```

- 'sposta_carriArmati_tra_territori'
    - Insieme alle procedure di attacco e posizionamento questa procedura permette di eseguire una delle azioni possibili durante il turno di un giocatore. In particolare l'azione di spostamento permette di rimuovere un certo numero di carri armati posizionati su un certo territorio (che deve essere di proprietà del giocatore) e di aggiungerli su un territorio ad esso adiacente (anch'esso di proprietà del giocatore).

```
CREATE PROCEDURE `sposta_carriArmati_tra_territori` (
IN var_gamerTag VARCHAR(32),
IN var_numCarriDaSpost INT,
IN var_terrPartenza VARCHAR(45),
IN var_terrArrivo VARCHAR(45))
BEGIN
    declare var_codicePartita INT;
    declare var_flag INT;
```

```
    declare var_numCarriPosizionati INT;

    SET var_codicePartita = recupera_partita(var_gamerTag);
    CALL controlli_prima_di_azione(var_gamerTag, var_codicePartita);
    CALL controllo_esistenza_territorio(var_terrPartenza);
    CALL controllo_esistenza_territorio(var_terrArrivo);

    -- controllo che il giocatore sia proprietario di entrambi i territori
    SELECT COUNT(*) FROM `IstanzaDiTerritorio`
    WHERE `Partita` = var_codicePartita AND `Proprietario` = var_gamerTag
    AND (`Territorio` = var_terrPartenza OR `Territorio` = var_terrArrivo)
    INTO var_flag;

    IF var_flag <> 2 THEN
           SIGNAL SQLSTATE '45018' SET MESSAGE_TEXT = 'Si possono spostare carri armati solo
tra territori che siano entrambi di proprietà del giocatore! Spostamento non disponibile.';
    END IF;

    -- controllo che i territori siano adiacenti
    SELECT COUNT(*) FROM `Adiacente`
    WHERE (`Territorio1` = var_terrPartenza AND `Territorio2` = var_terrArrivo) OR
(`Territorio1` = var_terrArrivo AND `Territorio2` = var_terrPartenza)
    INTO var_flag;

    IF var_flag <> 1 THEN
           SIGNAL SQLSTATE '45019' SET MESSAGE_TEXT = 'Non è possibile spostare carri armati in
un territorio non adiacente! Spostamento non disponibile.';
    END IF;

    -- controllo se è stato inserito un numero valido di territori da spostare
    SELECT `NumCarriPosizionati` FROM `IstanzaDiTerritorio`
    WHERE `Partita` = var_codicePartita AND `Territorio` = var_terrPartenza
    INTO var_numCarriPosizionati;

    IF var_numCarriDaSpost > var_numCarriPosizionati - 1 THEN
           SIGNAL SQLSTATE '45020' SET MESSAGE_TEXT = 'Non è possibile spostare più del numero
di carri armati presenti nel territorio di partenza -1 : Spostamento non disponibile.';
    END IF;

    UPDATE `IstanzaDiTerritorio` SET `NumCarriPosizionati` = `NumCarriPosizionati` -
var_numCarriDaSpost
    WHERE `Partita` = var_codicePartita AND `Territorio` = var_terrPartenza;

    UPDATE `IstanzaDiTerritorio` SET `NumCarriPosizionati` = `NumCarriPosizionati` +
var_numCarriDaSpost
    WHERE `Partita` = var_codicePartita AND `Territorio` = var_terrArrivo;

    CALL segna_ultimo_turno_azione(var_gamerTag, var_codicePartita);

    CALL cambio_turno(var_codicePartita, var_gamerTag);
END
```

- 'vedi_stato_di_gioco'
    - Procedura che permette al giocatore di vedere, durante il suo turno, per ciascuna istanza di territorio il nome del territorio, del proprietario attuale e il numero di carri armati che vi sono posizionati attualmente. Inoltre la procedura restituisce l'elenco dei giocatori partecipanti con i rispettivi numeri di carri armati a disposizione.
      NB:] READ COMMITTED per evitare dirty reads.

```
CREATE PROCEDURE `vedi_stato_di_gioco` (
IN var_gamerTag VARCHAR(32))
BEGIN
    declare var_codicePartita INT;

    declare exit handler for sqlexception
    begin
        rollback; ## annullo la transazione
        resignal; ## segnalo al chiamante
    end;

    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    SET TRANSACTION READ ONLY;
    START TRANSACTION ;
          SET var_codicePartita = recupera_partita(var_gamerTag);
          CALL controlli_prima_di_azione(var_gamerTag, var_codicePartita);

          -- TERRITORIO / PROPRIETARIO / ARMATE DISPIEGATE
          SELECT `Territorio`, `Proprietario`, `NumCarriPosizionati`
          FROM `IstanzaDiTerritorio`
          WHERE `Partita` = var_codicePartita ;

          -- GIOCATORE PARTECIPANTE / CARRI DISPONIBILI
          SELECT `GamerTag` AS `Giocatore`,`NumCarriDisponibili` AS `Carri disponibili` FROM
`Giocatore`
          JOIN `Partecipa` ON `Partecipa`.`Giocatore` = `Giocatore`.`GamerTag`
          WHERE `Partita` = var_codicePartita;
    COMMIT;
END
```

- 'vedi_storico_partite'
    - Procedura che permette al giocatore. in qualsiasi momento, di visualizzare lo storico delle partite giocate (si intendono tutte quelle partite, vinte o perse, che sono terminate quando il giocatore ancora non aveva abbandonato).
      NB:] READ COMMITTED per evitare dirty reads.

```
CREATE PROCEDURE `vedi_storico_partite` (
IN var_gamerTag VARCHAR(32))
BEGIN
    declare exit handler for sqlexception
```

```
    begin
        rollback; ## annullo la transazione
        resignal; ## segnalo al chiamante
    end;

    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    START TRANSACTION ;
          SELECT `Stanza`, `CodicePartita`, `Vincitore` FROM `Partita`
          JOIN `Partecipa` ON `Partecipa`.`Partita` = `Partita`.`CodicePartita`
          WHERE `Partecipa`.`Giocatore` = var_gamerTag AND `Partita`.`StatoPartita` = 'end';

          CALL reset_tempo_ultima_azione(var_gamerTag);
    COMMIT;
END
```

- 'salta_turno'
    - Procedura che permette al giocatore, durante il suo turno, di passarlo al prossimo giocatore senza svolgere alcuna azione.

```
CREATE PROCEDURE `salta_turno` (
IN var_gamerTag VARCHAR(32))
BEGIN
    declare var_codicePartita INT;
    declare var_statoPartita ENUM('wait','exec','end');

    SET var_codicePartita = recupera_partita(var_gamerTag);
    SET var_statoPartita = stato_partita(var_codicePartita);

    IF var_statoPartita <> 'exec' THEN
          IF var_statoPartita = 'wait' THEN
                SIGNAL SQLSTATE '45016' SET MESSAGE_TEXT = 'La partita non è ancora iniziata:
Azione non disponibile';
          ELSE
                SIGNAL SQLSTATE '45017' SET MESSAGE_TEXT = 'La partita è terminata.';
          END IF;
    END IF;

    CALL controlla_proprietario_turno(var_gamerTag, var_codicePartita);

    CALL cambio_turno(var_codicePartita, var_gamerTag);
END
```

- 'mostra_adiacenze_territorio'
    - Procedura che permette al giocatore, quando si trova in una stanza di gioco, di mostrare i territori adiacenti ad un dato territorio.
    - NB:] alleggerisco con READ UNCOMMITTED non essendoci anomalie possibili

```
CREATE PROCEDURE `mostra_adiacenze_territorio` (
```

```
IN var_nomeTerritorio VARCHAR(45))
BEGIN
    declare exit handler for sqlexception
    begin
        rollback; ## annullo la transazione
        resignal; ## segnalo al chiamante
    end;

    SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
    START TRANSACTION ;

        CALL controllo_esistenza_territorio(var_nomeTerritorio);

         SELECT `Territorio1` AS `Territori Adiacenti` FROM `Adiacente`
              WHERE `Territorio2` = var_nomeTerritorio
              UNION
              SELECT `Territorio2` FROM `Adiacente`
              WHERE `Territorio1` = var_nomeTerritorio;
    COMMIT;
END
```

- 'assegna_carriArmati_a_giocatore'
    - Procedura che permette l'assegnamento, ad un giocatore che ha compiuto un'azione durante il suo turno, di un numero di carri armati pari al numero di territori posseduti diviso tre, arrotondato per eccesso.
    - NB:] READ COMMITTED per evitare dirty reads.

```
CREATE PROCEDURE `assegna_carriArmati_a_giocatore` (
IN var_gamerTag VARCHAR(32))
BEGIN
    declare var_codicePartita INT;
    declare var_numCarriDaAss INT;

    declare exit handler for sqlexception
    begin
        rollback; ## annullo la transazione
        resignal; ## segnalo al chiamante
    end;

    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
    START TRANSACTION ;
          SET var_codicePartita = recupera_partita(var_gamerTag);

          SET var_numCarriDaAss = CEILING((calcola_possedimenti(var_gamerTag,
var_codicePartita))/3); -- numero di territori posseduti diviso 3 arrotondato per eccesso

          UPDATE `Giocatore` SET `NumCarriDisponibili` = `NumCarriDisponibili` +
var_numCarriDaAss
          WHERE `GamerTag` = var_gamerTag;
    COMMIT;
```

```
END
```

- ‘cambio_turno’
    - Procedura che permette il meccanismo di cambio di turno.

```
CREATE PROCEDURE `cambio_turno` (
IN var_codicePartita INT,
IN var_giocatorePrecedente VARCHAR(32))
BEGIN
    declare var_numPartecipanti INT;
    declare var_numTurnoPrecedente INT;
    declare var_idTurnoPrecedente INT;
    declare var_statoTurnoPrecedente ENUM('noaction','action','end');
    declare var_giocatoreSuccessivo VARCHAR(32);
    declare var_numTurnoSuccessivo INT;

    SET var_numPartecipanti = conta_partecipanti(var_codicePartita);

    -- mi ricavo il numero di turno del giocatore precedente
    SELECT `NumTurno` FROM `Partecipa`
    WHERE `Partita` = var_codicePartita AND `Giocatore` = var_giocatorePrecedente
    INTO var_numTurnoPrecedente;

    SET var_numTurnoSuccessivo = (var_numTurnoPrecedente % var_numPartecipanti) + 1;

    -- seleziono il gamerTag del prossimo giocatore
    SELECT `Giocatore` FROM `Partecipa`
    WHERE `Partita` = var_codicePartita
    AND `NumTurno` = var_numTurnoSuccessivo
    INTO var_giocatoreSuccessivo;

    SELECT `idTurno`,`StatoTurno` FROM `Turno`
    WHERE `Partita` = var_codicePartita AND `Giocatore` = var_giocatorePrecedente
    ORDER BY `idTurno` DESC LIMIT 1
    INTO var_idTurnoPrecedente, var_statoTurnoPrecedente;

    IF var_statoTurnoPrecedente = 'action' THEN
           CALL assegna_carriArmati_a_giocatore(var_giocatorePrecedente);
    END IF;

    UPDATE `Turno` SET `StatoTurno` = 'end'
    WHERE `idTurno` = var_idTurnoPrecedente;

    CALL inizia_turno(var_giocatoreSuccessivo, var_codicePartita);
END
```

- ‘controlla_proprietario_turno’
    - Procedura che permette di controllare che il giocatore passato in input corrisponda col giocatore proprietario del turno attualmente in corso nella partita passata in input.

```
CREATE PROCEDURE `controlla_proprietario_turno` (
IN var_gamerTag VARCHAR(32),
IN var_codicePartita INT)
BEGIN
    declare var_gamerCheck VARCHAR(32);

    SELECT `Giocatore` FROM `Turno`
    WHERE `Partita` = var_codicePartita
    AND `Turno`.`StatoTurno` <> 'end'
    INTO var_gamerCheck;

    IF var_gamerTag <> var_gamerCheck THEN
           SIGNAL SQLSTATE '45009' SET MESSAGE_TEXT = 'Non è il turno del giocatore! Azione non
disponibile.';
    END IF;
END
```

- 'controlli_prima_di_azione'
    - Procedura che permette, prima di un'azione (come attacco, posizionamento o spostamento) di controllare che la partita non sia ancora in attesa o terminata e che sia il turno del giocatore che richiede l'azione, controlla inoltre che il giocatore abbia ancora almeno un territorio (altrimenti significherebbe che ha perso salvo casi in cui giocatori con ancora possedimenti abbandonino la partita).

```
CREATE PROCEDURE `controlli_prima_di_azione` (
IN var_gamerTag VARCHAR(32),
IN var_codicePartita INT)
BEGIN
    declare var_statoPartita ENUM('wait','exec','end');

    SET var_statoPartita = stato_partita(var_codicePartita);

    IF var_statoPartita <> 'exec' THEN
           IF var_statoPartita = 'wait' THEN
                  SIGNAL SQLSTATE '45010' SET MESSAGE_TEXT = 'La partita non è ancora iniziata:
Azione non disponibile';
           ELSE
                  SIGNAL SQLSTATE '45011' SET MESSAGE_TEXT = 'La partita è terminata.';
           END IF;
    END IF;

    CALL controlla_proprietario_turno(var_gamerTag, var_codicePartita);

    IF calcola_possedimenti(var_gamerTag, var_codicePartita) = 0 THEN
        SIGNAL SQLSTATE '45012' SET MESSAGE_TEXT = 'Non hai più possedimenti, attendi che un
altro giocatore abbandoni o che la partita finisca';
    END IF;
END
```

- 'controllo_esistenza_territorio'
  - Procedura che permette di effettuare un controllo (senza la procedura avrebbe comportato l'inserimento dello stesso controllo in tutte le procedure che la chiamano) sull'esistenza effettiva del/dei territorio/i passati in input alle altre procedure.

```
CREATE PROCEDURE `controllo_esistenza_territorio` (
IN var_terr VARCHAR(45))
BEGIN
	IF var_terr NOT IN (SELECT `NomeTerritorio` FROM `Territorio`) THEN
		SIGNAL SQLSTATE '45013' SET MESSAGE_TEXT = 'Non esiste alcun territorio con
questo nome.';
	END IF;
END
```

- 'gioca_partita'
  - Procedura azionata dal trigger after update in partita che permette di "iniziare" la partita andando a creare le istanze di territorio che comporranno il tabellone e assegnandole ai partecipanti in maniera equa e casuale oltre a reimpostare la dotazione di carri armati disponibili per ciascun giocatore a seconda del numero di partecipanti.

```
CREATE PROCEDURE `gioca_partita` (
IN var_codicePartita INT)
BEGIN
    declare var_nomeTerritorio VARCHAR(45);
    declare var_numPartecipanti INT;
    declare var_carriArmatiDisp INT;
    declare var_gamerTag VARCHAR(32);
    declare var_whileIndex INT;

    SELECT conta_partecipanti(var_codicePartita) INTO var_numPartecipanti;

    IF var_numPartecipanti = 3 THEN SET var_carriArmatiDisp = 35;
    ELSEIF var_numPartecipanti = 4 THEN SET var_carriArmatiDisp = 30;
    ELSEIF var_numPartecipanti = 5 THEN SET var_carriArmatiDisp = 25;
    ELSE SET var_carriArmatiDisp = 20;
    END IF;

    -- imposto il numero di carri armati disponibili ai giocatori
    UPDATE `Giocatore` SET `NumCarriDisponibili` = var_carriArmatiDisp
    WHERE `GamerTag` IN (SELECT `Giocatore` FROM `Partecipa`
                                        WHERE `Partita` = var_codicePartita);

    -- creo e distribuisco le istanze dei territori casualmente
    SET var_whileIndex = 42;
    WHILE var_whileIndex > 0 DO
        -- seleziono casualmente un nome di territorio per cui l'istanza non è già stata
```

```
creata
            SELECT `NomeTerritorio` FROM `Territorio`
            WHERE `NomeTerritorio` NOT IN (SELECT `Territorio` FROM `IstanzaDiTerritorio`
                                           WHERE `Partita` = var_codicePartita)
            ORDER BY RAND() LIMIT 1
            INTO var_nomeTerritorio;

            -- assegnazione a carosello tra i partecipanti
            SELECT `Giocatore` FROM `Partecipa`
            WHERE `Partita` = var_codicePartita
            AND `NumTurno` = (ABS(var_whileIndex-42) % var_numPartecipanti) + 1
            INTO var_gamerTag;

            -- creo l'istanza e la assegno al giocatore selezionato
            INSERT INTO `IstanzaDiTerritorio` (`Territorio`, `Partita`, `Proprietario`)
            VALUES (var_nomeTerritorio, var_codicePartita, var_gamerTag) ;
            -- di default viene impostato il numero di carri armati sul territorio ad 1
            SET var_whileIndex = var_whileIndex - 1;
    END WHILE;
END
```

- 'inizia_turno'
    - Procedura che permette di creare il nuovo turno per il giocatore successivo impostando il timer del turno al tempo di creazione.

```
CREATE PROCEDURE `inizia_turno` (
IN var_gamerTag VARCHAR(32),
IN var_codicePartita INT)
BEGIN
    INSERT INTO `Turno` (`Partita`,`Giocatore`,`Timer`)
    VALUES (var_codicePartita,var_gamerTag,current_time());
END
```

- 'reset_tempo_ultima_azione'
    - Procedura che permette di aggiornare il tempo dell'ultima azione per un certo giocatore quando vengono fatte azioni specifiche quali visualizzazione dello storico e uscita da una partita con l'obiettivo di tenere traccia di tale informazione per il report dei moderatori.

```
CREATE PROCEDURE `reset_tempo_ultima_azione` (
IN var_gamerTag VARCHAR(32))
BEGIN
    UPDATE `Giocatore`
    SET `TempoUltimaAzione` = current_timestamp()
    WHERE `Giocatore`.`GamerTag` = var_gamerTag;
END
```

- ‘riassegna_turni’
    - Procedura che permette, una volta che un giocatore è uscito, di riassegnare i turni ai giocatori rimasti in partita mantenendoli in ordine di entrata nella stanza.

      NB:] questa procedura è invocata dalla procedura rimuovi_partecipante e pertanto vive nel contesto di isolamento di tale procedura, non vengono quindi specificati nuovi livelli di isolamento

```
CREATE PROCEDURE `riassegna_turni` (
IN var_codicePartita INT,
IN var_giocatoreUscito VARCHAR(32))
BEGIN
    declare done int default false;

    declare var_gamerTag VARCHAR(32);
    declare var_nomeTerrDaRiass VARCHAR(45);
    declare var_numTerrDaRiass INT;
    declare var_numPartecipanti INT;
    declare var_i INT;

    declare cur_gioc cursor for
    select `Giocatore` from `Partecipa`
    where `Partita` = var_codicePartita
    order by `NumTurno` asc ;

    declare continue handler for not found set done = true;

    SET var_i = 1;

    OPEN cur_gioc;
    update_loop: LOOP
              FETCH cur_gioc INTO var_gamerTag;
              IF done THEN
                     LEAVE update_loop;
              END IF;
        UPDATE `Partecipa` SET `NumTurno` = var_i
        WHERE `Giocatore`= var_gamerTag AND `Partita` = var_codicePartita;
        SET var_i = var_i + 1 ;
    END LOOP;
    CLOSE cur_gioc;

    SET var_numPartecipanti = conta_partecipanti(var_codicePartita);

    -- GESTIONE CASO PARTITA IN CORSO
    IF stato_partita(var_codicePartita) = 'exec' THEN -- redistribuzione territori di chi ha
abbandonato nel caso di partita in corso
              SET var_i = 0;

              SELECT COUNT(*) FROM `IstanzaDiTerritorio`
              WHERE `Partita` = var_codicePartita
              AND `Proprietario` = var_giocatoreUscito
```

```
                INTO var_numTerrDaRiass;

                WHILE var_numTerrDaRiass > 0 DO
                        -- seleziono casualmente un nome di territorio tra quelli rimasti di
proprietà del giocatore uscito
                        SELECT `Territorio` FROM `IstanzaDiTerritorio`
                        WHERE `Partita` = var_codicePartita
                        AND `Proprietario` = var_giocatoreUscito
                        ORDER BY RAND() LIMIT 1
                        INTO var_nomeTerrDaRiass;

                         -- assegnazione a carosello tra i partecipanti
                        SELECT `Giocatore` FROM `Partecipa`
                        WHERE `Partita` = var_codicePartita
                        AND `NumTurno` = (var_i % var_numPartecipanti) + 1
                        INTO var_gamerTag;

                        -- aggiorno l'istanza e la assegno al giocatore selezionato e reimposto il
numero di carri a 1 per non creare disparità tra le assegnazioni
                        UPDATE `IstanzaDiTerritorio`
                        SET `Proprietario` = var_gamerTag, `NumCarriPosizionati` = 1
                        WHERE `Partita` = var_codicePartita
                        AND `Territorio` = var_nomeTerrDaRiass;
                        SET var_i = var_i + 1;
                        SET var_numTerrDaRiass = var_numTerrDaRiass - 1;
                END WHILE;
    END IF;
END
```

- ‘risultato_lancio_dadi’
    - Procedura che permette di calcolare, dati il numero di dadi del giocatore che attacca e del giocatore che difende, il numero di carri armati eventualmente persi da ciascuno schieramento.

      NB:] alleggerisco con READ UNCOMMITTED dato che non ci possono essere anomalie.

```
CREATE PROCEDURE `risultato_lancio_dadi` (
IN var_numDadiAtt INT,
IN var_numDadiDif INT,
OUT var_numCarriPersiAtt INT,
OUT var_numCarriPersiDif INT)
BEGIN

    declare var_vald1Att INT DEFAULT NULL;
    declare var_vald2Att INT DEFAULT NULL;
    declare var_vald3Att INT DEFAULT NULL;
    declare var_vald1Dif INT DEFAULT NULL;
    declare var_vald2Dif INT DEFAULT NULL;
    declare var_vald3Dif INT DEFAULT NULL;
```

```
    declare var_vald1Tmp INT DEFAULT NULL;
    declare var_vald2Tmp INT DEFAULT NULL;
    declare var_vald3Tmp INT DEFAULT NULL;

    declare exit handler for sqlexception
    begin
        rollback; ## annullo la transazione
        resignal; ## segnalo al chiamante
    end;

    SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
    START TRANSACTION ;
            IF var_numDadiAtt = 1 THEN
                    SET var_vald1Att = FLOOR(RAND()*(6)) + 1;
            ELSEIF var_numDadiAtt = 2 THEN
                    SET var_vald1Tmp = FLOOR(RAND()*(6)) + 1;
                    SET var_vald2Tmp = FLOOR(RAND()*(6)) + 1;
                    SET var_vald1Att = GREATEST(var_vald1Tmp, var_vald2Tmp);
                    SET var_vald2Att = LEAST(var_vald1Tmp, var_vald2Tmp);
            ELSE
                    SET var_vald1Tmp = FLOOR(RAND()*(6)) + 1;
                    SET var_vald2Tmp = FLOOR(RAND()*(6)) + 1;
                    SET var_vald3Tmp = FLOOR(RAND()*(6)) + 1;
                    SET var_vald1Att = GREATEST(var_vald1Tmp, var_vald2Tmp, var_vald3Tmp);
                    SET var_vald3Att = LEAST(var_vald1Tmp, var_vald2Tmp, var_vald3Tmp);
                    SET var_vald2Att = var_vald1Tmp + var_vald2Tmp + var_vald3Tmp -
var_vald1Att - var_vald3Att;
            END IF;

            IF var_numDadiDif = 1 THEN
                    SET var_vald1Dif = FLOOR(RAND()*(6)) + 1;
            ELSEIF var_numDadiDif = 2 THEN
                    SET var_vald1Tmp = FLOOR(RAND()*(6)) + 1;
                    SET var_vald2Tmp = FLOOR(RAND()*(6)) + 1;
                    SET var_vald1Dif = GREATEST(var_vald1Tmp, var_vald2Tmp);
                    SET var_vald2Dif = LEAST(var_vald1Tmp, var_vald2Tmp);
            ELSE
                    SET var_vald1Tmp = FLOOR(RAND()*(6)) + 1;
                    SET var_vald2Tmp = FLOOR(RAND()*(6)) + 1;
                    SET var_vald3Tmp = FLOOR(RAND()*(6)) + 1;
                    SET var_vald1Dif = GREATEST(var_vald1Tmp, var_vald2Tmp, var_vald3Tmp);
                    SET var_vald3Dif = LEAST(var_vald1Tmp, var_vald2Tmp, var_vald3Tmp);
                    SET var_vald2Dif = var_vald1Tmp + var_vald2Tmp + var_vald3Tmp -
var_vald1Att - var_vald3Att;
            END IF;

            SET var_numCarriPersiAtt = 0;
            SET var_numCarriPersiDif = 0;
            IF ((var_vald1Att IS NOT NULL) AND (var_vald1Dif IS NOT NULL)) THEN
                    IF var_vald1Att > var_vald1Dif THEN
                            SET var_numCarriPersiDif = 1;
                    ELSE
                            SET var_numCarriPersiAtt = 1;
```

```
                END IF;
            END IF;
            IF ((var_vald2Att IS NOT NULL) AND (var_vald2Dif IS NOT NULL)) THEN
                IF var_vald2Att > var_vald2Dif THEN
                    SET var_numCarriPersiDif = var_numCarriPersiDif + 1;
                ELSE
                    SET var_numCarriPersiAtt = var_numCarriPersiAtt + 1;
                END IF;
            END IF;
            IF ((var_vald3Att IS NOT NULL) AND (var_vald3Dif IS NOT NULL)) THEN
                IF var_vald3Att > var_vald3Dif THEN
                    SET var_numCarriPersiDif = var_numCarriPersiDif + 1;
                ELSE
                    SET var_numCarriPersiAtt = var_numCarriPersiAtt + 1;
                END IF;
            END IF;
        COMMIT;
END
```

- ‘segna_ultimo_turno_azione’
    - Procedura che permette di aggiornare lo stato di un turno in ‘action’ così da poterne tenere traccia in fase di eventuale assegnazione di nuovi carri armati al cambio del turno.

```
CREATE PROCEDURE `segna_ultimo_turno_azione`(
IN var_gamerTag VARCHAR(32),
IN var_codicePartita INT)
BEGIN
        declare var_ultimoTurnoGiocatore INT;

        SELECT max(`idTurno`) FROM `Turno`
        WHERE `Partita` = var_codicePartita
        AND `Giocatore` = var_gamerTag
        INTO var_ultimoTurnoGiocatore;

        UPDATE `Turno` SET `StatoTurno` = 'action'
        WHERE `idTurno` = var_ultimoTurnoGiocatore;
END
```

Funzioni

- ‘calcola_possedimenti’
    - Funzione che, dato in input un gamerTag e un codicePartita, calcola i possedimenti di quel giocatore in quella partita.

```
CREATE FUNCTION `calcola_possedimenti` (var_gamerTag VARCHAR(32),var_codicePartita INT) RETURNS
INT READS SQL DATA
```

```
BEGIN
    declare var_numPossedimenti INT;

    SELECT COUNT(*) FROM `IstanzaDiTerritorio`
    WHERE `Partita` = var_codicePartita AND `Proprietario` = var_gamerTag
    INTO var_numPossedimenti;

    RETURN var_numPossedimenti;

END
```

- 'conta_partecipanti'
    - Funzione che, dato in input un codicePartita, conta i partecipanti attuali a quella partita.

```
CREATE FUNCTION `conta_partecipanti` (var_codicePartita INT) RETURNS INT READS SQL DATA
BEGIN
    declare var_numPartecipanti INT;

    SELECT COUNT(*) FROM `Partecipa`
    WHERE `Partecipa`.`Partita` = var_codicePartita
    INTO var_numPartecipanti;

    RETURN var_numPartecipanti;
END
```

- 'controlla_giocatore_in_partita'
    - Funzione che, dato in input un gamerTag, conta le partecipazioni di quel giocatore a partite con stato diverso da 'end'

```
CREATE FUNCTION `controlla_giocatore_in_partita`(var_gamertag VARCHAR(32)) RETURNS INT READS SQL
DATA
BEGIN
       declare var_partecipazioni INT;

    SELECT COUNT(*) FROM `Partecipa`
    JOIN `Partita` ON `Partecipa`.`Partita` = `Partita`.`CodicePartita`
    WHERE `Partecipa`.`Giocatore` = var_gamerTag AND `Partita`.`StatoPartita` <> 'end'
    INTO var_partecipazioni;

    RETURN var_partecipazioni;
END
```

- 'recupera_partita'
    - Funzione che, dato in input un gamerTag, restituisce il codice dell'ultima partita a cui quel giocatore ha partecipato

```
CREATE FUNCTION `recupera_partita` (var_gamerTag VARCHAR(32)) RETURNS INT READS SQL DATA
BEGIN
    declare var_codicePartita INT;

    SELECT `CodicePartita` FROM `Partita`
    JOIN `Partecipa` ON `Partecipa`.`Partita` = `Partita`.`CodicePartita`
    WHERE `Partecipa`.`Giocatore` = var_gamerTag
    ORDER BY `CodicePartita` DESC LIMIT 1
    INTO var_codicePartita;

    RETURN var_codicePartita;
END
```

- 'stato_partita'
    - Funzione che, dato in input un codicePartita, restituisce lo stato di quella partita.

```
CREATE FUNCTION `risikoDB`.`stato_partita` (var_codicePartita INT) RETURNS
ENUM('wait','exec','end') READS SQL DATA
BEGIN
    declare var_statoPartita ENUM('wait','exec','end');

    SELECT `StatoPartita` FROM `Partita`
    WHERE `CodicePartita` = var_codicePartita
    INTO var_statoPartita;

    RETURN var_statoPartita;
END
```

Appendice: Implementazione

Codice SQL per istanziare il database

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION
_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema risikoDB
-- -----------------------------------------------------
DROP SCHEMA IF EXISTS `risikoDB` ;

-- -----------------------------------------------------
-- Schema risikoDB
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `risikoDB` ;
USE `risikoDB` ;

-- -----------------------------------------------------
-- Table `risikoDB`.`Utenti`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `risikoDB`.`Utenti` ;

CREATE TABLE IF NOT EXISTS `risikoDB`.`Utenti` (
  `Username` VARCHAR(32) NOT NULL,
  `Password` VARCHAR(45) NOT NULL,
  `Ruolo` ENUM('moderatore', 'giocatore') NOT NULL,
  PRIMARY KEY (`Username`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `risikoDB`.`Giocatore`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `risikoDB`.`Giocatore` ;

CREATE TABLE IF NOT EXISTS `risikoDB`.`Giocatore` (
  `GamerTag` VARCHAR(32) NOT NULL,
  `NumCarriDisponibili` INT NULL,
  `TempoUltimaAzione` TIME NULL,
  PRIMARY KEY (`GamerTag`),
  CONSTRAINT `GamerTag_Giocatore`
    FOREIGN KEY (`GamerTag`)
    REFERENCES `risikoDB`.`Utenti` (`Username`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `risikoDB`.`StanzaDiGioco`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `risikoDB`.`StanzaDiGioco` ;

CREATE TABLE IF NOT EXISTS `risikoDB`.`StanzaDiGioco` (
  `NomeStanza` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`NomeStanza`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `risikoDB`.`Partita`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `risikoDB`.`Partita` ;

CREATE TABLE IF NOT EXISTS `risikoDB`.`Partita` (
  `CodicePartita` INT NOT NULL AUTO_INCREMENT,
  `Stanza` VARCHAR(45) NOT NULL,
  `StatoPartita` ENUM('wait', 'exec', 'end') NOT NULL DEFAULT 'wait',
  `CountdownClock` TIME NULL,
  `Vincitore` VARCHAR(32) NULL,
  PRIMARY KEY (`CodicePartita`),
  CONSTRAINT `Vincitore_Partita`
    FOREIGN KEY (`Vincitore`)
    REFERENCES `risikoDB`.`Giocatore` (`GamerTag`)
    ON DELETE SET NULL
    ON UPDATE NO ACTION,
  CONSTRAINT `Stanza_Partita`
    FOREIGN KEY (`Stanza`)
    REFERENCES `risikoDB`.`StanzaDiGioco` (`NomeStanza`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `VINCITORE_PARTITA_idx` ON `risikoDB`.`Partita` (`Vincitore` ASC) VISIBLE;

CREATE INDEX `Stanza_Partita_idx` ON `risikoDB`.`Partita` (`Stanza` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `risikoDB`.`Territorio`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `risikoDB`.`Territorio` ;

CREATE TABLE IF NOT EXISTS `risikoDB`.`Territorio` (
  `NomeTerritorio` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`NomeTerritorio`))
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `risikoDB`.`IstanzaDiTerritorio`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `risikoDB`.`IstanzaDiTerritorio` ;

CREATE TABLE IF NOT EXISTS `risikoDB`.`IstanzaDiTerritorio` (
  `Territorio` VARCHAR(45) NOT NULL,
  `Partita` INT NOT NULL,
  `Proprietario` VARCHAR(32) NOT NULL,
  `NumCarriPosizionati` INT NOT NULL DEFAULT 1,
  PRIMARY KEY (`Territorio`, `Partita`),
  CONSTRAINT `Territorio_IstanzaDiTerritorio`
    FOREIGN KEY (`Territorio`)
    REFERENCES `risikoDB`.`Territorio` (`NomeTerritorio`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT `Partita_IstanzaDiTerritorio`
    FOREIGN KEY (`Partita`)
    REFERENCES `risikoDB`.`Partita` (`CodicePartita`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT `Proprietario_IstanzaDiTerritorio`
    FOREIGN KEY (`Proprietario`)
    REFERENCES `risikoDB`.`Giocatore` (`GamerTag`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `Partita_IstanzaDiTerritorio_idx` ON `risikoDB`.`IstanzaDiTerritorio` (`Partita`
ASC) VISIBLE;

CREATE INDEX `Proprietario_IstanzaDiTerritorio_idx` ON `risikoDB`.`IstanzaDiTerritorio`
(`Proprietario` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `risikoDB`.`Partecipa`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `risikoDB`.`Partecipa` ;

CREATE TABLE IF NOT EXISTS `risikoDB`.`Partecipa` (
  `Giocatore` VARCHAR(32) NOT NULL,
  `Partita` INT NOT NULL,
  `NumTurno` INT NOT NULL,
  PRIMARY KEY (`Giocatore`, `Partita`),
  CONSTRAINT `Giocatore_Partecipa`
    FOREIGN KEY (`Giocatore`)
    REFERENCES `risikoDB`.`Giocatore` (`GamerTag`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT `Partita_Partecipa`
    FOREIGN KEY (`Partita`)
    REFERENCES `risikoDB`.`Partita` (`CodicePartita`)
```

```
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `Partita_Partecipa_idx` ON `risikoDB`.`Partecipa` (`Partita` ASC) INVISIBLE;

CREATE UNIQUE INDEX `Partita_NumTurno_uq` ON `risikoDB`.`Partecipa` (`Partita` ASC, `NumTurno`
ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `risikoDB`.`Adiacente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `risikoDB`.`Adiacente` ;

CREATE TABLE IF NOT EXISTS `risikoDB`.`Adiacente` (
  `Territorio1` VARCHAR(45) NOT NULL,
  `Territorio2` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Territorio1`, `Territorio2`),
  CONSTRAINT `Territorio_Adiacente_1`
    FOREIGN KEY (`Territorio1`)
    REFERENCES `risikoDB`.`Territorio` (`NomeTerritorio`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT `Territorio_Adiacente_2`
    FOREIGN KEY (`Territorio2`)
    REFERENCES `risikoDB`.`Territorio` (`NomeTerritorio`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `Territorio_Adiacente_2_idx` ON `risikoDB`.`Adiacente` (`Territorio2` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `risikoDB`.`Turno`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `risikoDB`.`Turno` ;

CREATE TABLE IF NOT EXISTS `risikoDB`.`Turno` (
  `idTurno` INT NOT NULL AUTO_INCREMENT,
  `Partita` INT NOT NULL,
  `Giocatore` VARCHAR(32) NOT NULL,
  `StatoTurno` ENUM('noaction', 'action', 'end') NOT NULL DEFAULT 'noaction',
  `Timer` TIME NULL,
  PRIMARY KEY (`idTurno`, `Partita`, `Giocatore`),
  CONSTRAINT `Partita_Turno`
    FOREIGN KEY (`Partita`)
    REFERENCES `risikoDB`.`Partita` (`CodicePartita`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `Giocatore_Turno`
    FOREIGN KEY (`Giocatore`)
```

```
    REFERENCES `risikoDB`.`Giocatore` (`GamerTag`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `Partita_Turno_idx` ON `risikoDB`.`Turno` (`Partita` ASC) VISIBLE;

CREATE INDEX `Giocatore_Turno_idx` ON `risikoDB`.`Turno` (`Giocatore` ASC) VISIBLE;

USE `risikoDB` ;

DELIMITER ;
SET SQL_MODE = '';
DROP USER IF EXISTS login;
SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION
_BY_ZERO,NO_ENGINE_SUBSTITUTION';
CREATE USER 'login' IDENTIFIED BY 'Login@000';

GRANT EXECUTE ON procedure `risikoDB`.`login` TO 'login';
GRANT EXECUTE ON procedure `risikoDB`.`registra_nuovo_giocatore` TO 'login';
SET SQL_MODE = '';
DROP USER IF EXISTS moderatore;
SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION
_BY_ZERO,NO_ENGINE_SUBSTITUTION';
CREATE USER 'moderatore' IDENTIFIED BY 'Moderatore@000';

GRANT EXECUTE ON procedure `risikoDB`.`report_moderatori` TO 'moderatore';
GRANT EXECUTE ON procedure `risikoDB`.`crea_nuova_stanza` TO 'moderatore';
GRANT EXECUTE ON procedure `risikoDB`.`registra_nuovo_moderatore` TO 'moderatore';
SET SQL_MODE = '';
DROP USER IF EXISTS giocatore;
SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION
_BY_ZERO,NO_ENGINE_SUBSTITUTION';
CREATE USER 'giocatore' IDENTIFIED BY 'Giocatore@000';

GRANT EXECUTE ON procedure `risikoDB`.`aggiungi_partecipante` TO 'giocatore';
GRANT EXECUTE ON procedure `risikoDB`.`attacca_territorio` TO 'giocatore';
GRANT EXECUTE ON procedure `risikoDB`.`ottieni_lista_stanze_disponibili` TO 'giocatore';
GRANT EXECUTE ON procedure `risikoDB`.`posiziona_carriArmati_su_territorio` TO 'giocatore';
GRANT EXECUTE ON procedure `risikoDB`.`rimuovi_partecipante` TO 'giocatore';
GRANT EXECUTE ON procedure `risikoDB`.`sposta_carriArmati_tra_territori` TO 'giocatore';
GRANT EXECUTE ON procedure `risikoDB`.`vedi_stato_di_gioco` TO 'giocatore';
GRANT EXECUTE ON procedure `risikoDB`.`vedi_storico_partite` TO 'giocatore';
GRANT EXECUTE ON procedure `risikoDB`.`salta_turno` TO 'giocatore';
GRANT EXECUTE ON procedure `risikoDB`.`mostra_adiacenze_territorio` TO 'giocatore';

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

```
-- -----------------------------------------------------
-- Data for table `risikoDB`.`Utenti`
-- -----------------------------------------------------
START TRANSACTION;
USE `risikoDB`;
INSERT INTO `risikoDB`.`Utenti` (`Username`, `Password`, `Ruolo`) VALUES ('Moderatore',
'88fc268ab1dee02107c47cc0353ddf881c1a85d8', 'moderatore');
INSERT INTO `risikoDB`.`Utenti` (`Username`, `Password`, `Ruolo`) VALUES ('EdoMan000',
'9025171ebffa5cbb4ce816a02e257aaa6a80e2e1', 'giocatore');
INSERT INTO `risikoDB`.`Utenti` (`Username`, `Password`, `Ruolo`) VALUES ('EdoMan001',
'9025171ebffa5cbb4ce816a02e257aaa6a80e2e1', 'giocatore');
INSERT INTO `risikoDB`.`Utenti` (`Username`, `Password`, `Ruolo`) VALUES ('EdoMan002',
'9025171ebffa5cbb4ce816a02e257aaa6a80e2e1', 'giocatore');
INSERT INTO `risikoDB`.`Utenti` (`Username`, `Password`, `Ruolo`) VALUES ('EdoMan003',
'9025171ebffa5cbb4ce816a02e257aaa6a80e2e1', 'giocatore');

COMMIT;


-- -----------------------------------------------------
-- Data for table `risikoDB`.`Giocatore`
-- -----------------------------------------------------
START TRANSACTION;
USE `risikoDB`;
INSERT INTO `risikoDB`.`Giocatore` (`GamerTag`, `NumCarriDisponibili`, `TempoUltimaAzione`)
VALUES ('EdoMan000', NULL, NULL);
INSERT INTO `risikoDB`.`Giocatore` (`GamerTag`, `NumCarriDisponibili`, `TempoUltimaAzione`)
VALUES ('EdoMan001', NULL, NULL);
INSERT INTO `risikoDB`.`Giocatore` (`GamerTag`, `NumCarriDisponibili`, `TempoUltimaAzione`)
VALUES ('EdoMan002', NULL, NULL);
INSERT INTO `risikoDB`.`Giocatore` (`GamerTag`, `NumCarriDisponibili`, `TempoUltimaAzione`)
VALUES ('EdoMan003', NULL, NULL);

COMMIT;


-- -----------------------------------------------------
-- Data for table `risikoDB`.`StanzaDiGioco`
-- -----------------------------------------------------
START TRANSACTION;
USE `risikoDB`;
INSERT INTO `risikoDB`.`StanzaDiGioco` (`NomeStanza`) VALUES ('StanzaDiProva1');
INSERT INTO `risikoDB`.`StanzaDiGioco` (`NomeStanza`) VALUES ('StanzaDiProva2');

COMMIT;


-- -----------------------------------------------------
-- Data for table `risikoDB`.`Partita`
-- -----------------------------------------------------
START TRANSACTION;
USE `risikoDB`;
INSERT INTO `risikoDB`.`Partita` (`CodicePartita`, `Stanza`, `StatoPartita`, `CountdownClock`,
```

```
`Vincitore`) VALUES (1, 'StanzaDiProva1', 'wait', NULL, NULL);
INSERT INTO `risikoDB`.`Partita` (`CodicePartita`, `Stanza`, `StatoPartita`, `CountdownClock`,
`Vincitore`) VALUES (2, 'StanzaDiProva2', 'wait', NULL, NULL);

COMMIT;


-- -----------------------------------------------------
-- Data for table `risikoDB`.`Territorio`
-- -----------------------------------------------------
START TRANSACTION;
USE `risikoDB`;
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('ALASKA');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('ALBERTA');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('AMERICA CENTRALE');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('STATI UNITI ORIENTALI');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('STATI UNITI OCCIDENTALI');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('GROENLANDIA');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('TERRITORI DEL NORD OVEST');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('ONTARIO');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('QUEBEC');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('ARGENTINA');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('BRASILE');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('PERU');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('VENEZUELA');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('GRAN BRETAGNA');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('ISLANDA');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('EUROPA SETTENTRIONALE');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('EUROPA MERIDIONALE');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('EUROPA OCCIDENTALE');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('SCANDINAVIA');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('UCRAINA');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('CONGO');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('EGITTO');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('MADAGASCAR');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('AFRICA DEL NORD');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('AFRICA ORIENTALE');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('AFRICA DEL SUD');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('AFGHANISTAN');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('INDIA');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('CITA');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('CINA');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('GIAPPONE');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('KAMCHATKA');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('MEDIO ORIENTE');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('MONGOLIA');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('SIAM');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('SIBERIA');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('URALI');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('JACUZIA');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('INDONESIA');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('NUOVA GUINEA');
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('AUSTRALIA OCCIDENTALE');
```

```
INSERT INTO `risikoDB`.`Territorio` (`NomeTerritorio`) VALUES ('AUSTRALIA ORIENTALE');

COMMIT;


-- -----------------------------------------------------
-- Data for table `risikoDB`.`Adiacente`
-- -----------------------------------------------------
START TRANSACTION;
USE `risikoDB`;
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('ARGENTINA', 'PERU');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('ARGENTINA',
'BRASILE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('PERU', 'BRASILE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('PERU', 'VENEZUELA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('BRASILE',
'VENEZUELA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('BRASILE', 'AFRICA DEL
NORD');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('VENEZUELA', 'AMERICA
CENTRALE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('AMERICA CENTRALE',
'STATI UNITI OCCIDENTALI');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('AMERICA CENTRALE',
'STATI UNITI ORIENTALI');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('STATI UNITI
OCCIDENTALI', 'STATI UNITI ORIENTALI');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('STATI UNITI
OCCIDENTALI', 'ALBERTA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('STATI UNITI
OCCIDENTALI', 'ONTARIO');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('STATI UNITI
ORIENTALI', 'ONTARIO');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('STATI UNITI
ORIENTALI', 'QUEBEC');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('ALBERTA', 'ONTARIO');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('ALBERTA', 'TERRITORI
DEL NORD OVEST');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('ALBERTA', 'ALASKA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('ONTARIO', 'QUEBEC');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('ONTARIO', 'TERRITORI
DEL NORD OVEST');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('ONTARIO',
'GROENLANDIA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('QUEBEC',
'GROENLANDIA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('TERRITORI DEL NORD
OVEST', 'ALASKA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('TERRITORI DEL NORD
OVEST', 'GROENLANDIA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('ALASKA',
'KAMCHATKA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('GROENLANDIA',
```

```
'ISLANDA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('ISLANDA',
'SCANDINAVIA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('ISLANDA', 'GRAN
BRETAGNA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('SCANDINAVIA',
'UCRAINA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('SCANDINAVIA', 'GRAN
BRETAGNA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('SCANDINAVIA', 'EUROPA
SETTENTRIONALE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('GRAN BRETAGNA',
'EUROPA SETTENTRIONALE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('GRAN BRETAGNA',
'EUROPA OCCIDENTALE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('UCRAINA', 'EUROPA
SETTENTRIONALE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('UCRAINA', 'EUROPA
MERIDIONALE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('UCRAINA',
'AFGHANISTAN');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('UCRAINA', 'URALI');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('UCRAINA', 'MEDIO
ORIENTE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('EUROPA
SETTENTRIONALE', 'EUROPA OCCIDENTALE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('EUROPA
SETTENTRIONALE', 'EUROPA MERIDIONALE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('EUROPA OCCIDENTALE',
'EUROPA MERIDIONALE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('EUROPA OCCIDENTALE',
'AFRICA DEL NORD');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('EUROPA MERIDIONALE',
'EGITTO');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('EUROPA MERIDIONALE',
'MEDIO ORIENTE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('EUROPA MERIDIONALE',
'AFRICA DEL NORD');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('MEDIO ORIENTE',
'CINA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('MEDIO ORIENTE',
'EGITTO');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('MEDIO ORIENTE',
'INDIA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('MEDIO ORIENTE',
'AFGHANISTAN');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('AFGHANISTAN',
'URALI');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('AFGHANISTAN',
'CINA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('URALI', 'CINA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('URALI', 'SIBERIA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('AFRICA DEL NORD',
```

```
'EGITTO');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('AFRICA DEL NORD',
'CONGO');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('AFRICA DEL NORD',
'AFRICA ORIENTALE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('EGITTO', 'AFRICA
ORIENTALE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('CINA', 'INDIA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('CINA', 'MONGOLIA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('CINA', 'SIBERIA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('CINA', 'SIAM');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('INDIA', 'SIAM');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('SIBERIA',
'MONGOLIA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('SIBERIA', 'CITA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('SIBERIA', 'JACUZIA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('CONGO', 'AFRICA
ORIENTALE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('CONGO', 'AFRICA DEL
SUD');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('AFRICA ORIENTALE',
'AFRICA DEL SUD');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('AFRICA ORIENTALE',
'MADAGASCAR');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('AFRICA DEL SUD',
'MADAGASCAR');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('MONGOLIA', 'CITA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('MONGOLIA',
'GIAPPONE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('MONGOLIA',
'KAMCHATKA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('KAMCHATKA',
'GIAPPONE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('KAMCHATKA', 'CITA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('SIAM', 'INDONESIA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('CITA', 'JACUZIA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('INDONESIA', 'NUOVA
GUINEA');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('INDONESIA',
'AUSTRALIA OCCIDENTALE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('NUOVA GUINEA',
'AUSTRALIA OCCIDENTALE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('NUOVA GUINEA',
'AUSTRALIA ORIENTALE');
INSERT INTO `risikoDB`.`Adiacente` (`Territorio1`, `Territorio2`) VALUES ('AUSTRALIA
OCCIDENTALE', 'AUSTRALIA ORIENTALE');

COMMIT;
```

Codice del Front-End

./code/

main.c

```c
#include "main.h"

int main() {
    clearScreen();
    if (!loadConfiguration()) {
        printError("Errore: Configurazione Ambiente Non Riuscita!") ;
        exit(-1);
    }
    if (connectToDatabase()) {
        loginController();
    }
}
```

main.h

```c
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <stdlib.h>

#include "config/environment.h"
#include "engineering/connector.h"
#include "engineering/inout.h"
#include "controller/loginController.h"
#include "view/viewUtils.h"
```

./code/config

environment.c

```c
#include "environment.h"

const char *CONFIGURATION_FILE_NAME = "code/config/configuration.properties" ;

bool setEnvironmentVariable(char *environmentToken) ;

bool loadConfiguration() {
    // Open configuration file to prepare env
    FILE *configurationFile = fopen(CONFIGURATION_FILE_NAME, "r") ;
    if (configurationFile == NULL) {
        fprintf(stderr, "Errore apertura file\n") ;
        return false ;
    }

    char *environmentLine = NULL ;
    size_t len ;
```

```
    ssize_t nread = getline(&environmentLine, &len, configurationFile) ;
    while (nread > 0) {
        char *environmentToken = strtok(environmentLine, "\n") ;

        if (setEnvironmentVariable(environmentToken) == false) {
            free(environmentLine) ;
            fclose(configurationFile) ;
            return false ;
        }

        nread = getline(&environmentLine, &len, configurationFile) ;
    }
    //getline alloca memoria quando viene passata un NULL come puntatore --> libero memoria
allocata
    free(environmentLine) ;

    fclose(configurationFile) ;
    return true ;
}


bool setEnvironmentVariable(char *environmentToken) {
    //Function to parse Environment Variable from configuration file
    char *tokenName = strtok(environmentToken, "=") ;
    char *tokenValue = strtok(NULL, "=") ;

    if (setenv(tokenName, tokenValue, 0)) {
        printf("Errore Impostazione Variabile d'Ambiente\n") ;
        return false ;
    }
    return true ;
}
```

environment.h

```
#pragma once

#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <stdlib.h>

bool loadConfiguration();
```

configuration.properties

```
DB.HOST=localhost
DB.PORT=3306
DB.NAME=risikoDB
LOGIN.USER=login
LOGIN.PASSWD=Login@000
MODERATORE.USER=moderatore
```

```
MODERATORE.PASSWD=Moderatore@000
GIOCATORE.USER=giocatore
GIOCATORE.PASSWD=Giocatore@000
```

./code/controller

loginController.c

```c
#include "loginController.h"

static bool switchRole(Role role) {
    char *databaseName = getenv("DB.NAME") ;
    char *username ;
    char *password ;

    switch(role) {
        case MODERATORE :
            username = getenv(DB_MODERATORE_USER) ;
            password = getenv(DB_MODERATORE_PASSWD) ;
            break ;
        case GIOCATORE :
            username = getenv(DB_GIOCATORE_USER) ;
            password = getenv(DB_GIOCATORE_PASSWD) ;
            break ;
        case LOGIN :
            username = getenv(DB_LOGIN_USER) ;
            password = getenv(DB_LOGIN_PASSWD) ;
            break ;
    }

    if (username == NULL || password == NULL || databaseName == NULL) {
        printError("Errore : Variabili d'Ambiente Non Trovate") ;
        return false ;
    }

    if (mysql_change_user(conn, username, password, databaseName) != 0) {
        print_sql_error(conn, "Errore SQL: Impossibile Cambiare Privilegi Utente") ;
        return false ;
    }

    return true ;
}

static bool successLogin(Role myRole, char* username) {
    clearScreen() ;
    showAppTitle() ;
    bool flag = false;
    if (switchRole(myRole) == false) {
        return false;
    }

    switch (myRole) {
        case MODERATORE :
```

```
            moderatoreController(username) ;
            flag = true ;
            break;
        case GIOCATORE :
            giocatoreController(username) ;
            flag = true ;
            break;
        case LOGIN :
            printError("login fallito");
            break;
    }

    switchRole(LOGIN) ;
    return flag;
}

static void login(){
    clearScreen();
    showAppTitle();
    puts("\t\t\t\t|_____________________|");
    puts("\t\t\t\t|             LOGIN               |");
    puts("\t\t\t\t|_________________________________|\n");
    Credentials creds ;
    int failed_attempts = 0;

    do {
        if(failed_attempts == 3){
            return;
        }
        memset(&creds, 0, sizeof(Credentials)) ;
        Role myRole = LOGIN ;

        if (promptLoginAndRegistration(&creds)) {
            myRole = logAsUser(creds) ;
            if (myRole == LOGIN) {
                printError("LOGIN FALLITO: Username e/o Password non corrispondono ad alcun
utente") ;
                failed_attempts ++;
            }
            else {
                if(successLogin(myRole, creds.username)){
                    return;
                }
            }
        }else{
            failed_attempts ++;
        }
    } while (true) ;
}

static bool registration(){
    clearScreen();
    showAppTitle();
```

```
    puts("\t\t\t\t|_____________________|");
    puts("\t\t\t\t|          REGISTRAZIONE          |");
    puts("\t\t\t\t|_________________________________|\n");
    Credentials creds;
    int failed_attempts = 0;
    do {
        if(failed_attempts == 3){
            return false;
        }
        memset(&creds, 0, sizeof(Credentials));
        if(promptLoginAndRegistration(&creds)){
            if(registerNewPlayer(creds)){
                printSuccess("Giocatore registrato correttamente");
                return true;
            }
        }
        failed_attempts ++;
    } while (true) ;

}

void loginController() {
main_menu:
    clearScreen();
    showAppTitle();
    puts("\t\t\t\t|_____________________|");
    puts("\t\t\t\t|         MENU PRINCIPALE         |");
    puts("\t\t\t\t|_________________________________|\n");
    int input;
    int failed_attempts = 0;
    while (true) {
        if(failed_attempts == 3){
            goto main_menu;
        }
        input = promptInitialMenu();
              switch (input)
        {
            case 1:
                login();
                goto main_menu;
                break;
            case 2:
                if(!registration()){
                    goto main_menu;
                }
                break;
            case 3:
                puts("");
                printf("\033[41m%s\033[0m",
"-----------------------ARRIVEDERCI-----------------------");
                puts("");
                puts("");
                mysql_close(conn);
```

```
                exit(0);
            default:
                printError("Scegli tra le opzioni proposte!");
                failed_attempts ++;
                break;
        }
    }
```

loginController.h

```
#pragma once

#include <stdbool.h>
#include <stdlib.h>
#include <string.h>

#include "../view/loginAndRegistrationView.h"
#include "../database/db.h"
#include "../database/dbUtils.h"
#include "../engineering/inout.h"
#include "../view/viewUtils.h"
#include "../model/credentials.h"
#include "giocatoreController.h"
#include "moderatoreController.h"
#include <stdbool.h>

void loginController();
```

giocatoreController.c

```
#include "giocatoreController.h"

char *g_username;

//[1] VEDI STATO DI GIOCO
static bool vediStatoDiGioco(){
    printSuccess("-- GENERAZIONE STATO DI GIOCO INIZIATA --");
    if(showMatchStatus(g_username)){
        printSuccess("-- GENERAZIONE STATO DI GIOCO TERMINATA --");
        return true;
    }else{
        return false;
    }
}

//[2] SALTA TURNO
static bool saltaTurno(){
    if(skipTurn(g_username)){
        puts("");
        printf("\033[43m%s\033[0m","-- TURNO SALTATO --");
        puts("");
        puts("");
```

```
        return true;
    }else{
        return false;
    }
}

//[3] ATTACCA TERRITORIO
static bool attaccaTerritorio(){
    char nomeTerritorio1[VARCHAR45];
    char nomeTerritorio2[VARCHAR45];
    int num_army;
    int num_army_loss_attack;
    int num_army_loss_defense;
    int is_conquered;
    int i_won;
    do{
        memset(&nomeTerritorio1, 0, sizeof(nomeTerritorio1));
        memset(&nomeTerritorio2, 0, sizeof(nomeTerritorio2));
        memset(&num_army, 0, sizeof(num_army));
        if(getAttackInfo(nomeTerritorio1, nomeTerritorio2, &num_army)){
            //conversione nome territori in uppercase
            toUpperCase(nomeTerritorio1);
            toUpperCase(nomeTerritorio2);
            if(attackTerritory(g_username, &num_army, nomeTerritorio1, nomeTerritorio2,
&num_army_loss_attack, &num_army_loss_defense, &is_conquered, &i_won)){

printAttackResults(nomeTerritorio1,nomeTerritorio2,num_army_loss_attack,num_army_loss_defense);
                puts("");
                puts("");
                if(is_conquered == 1){
                    printConqueredTerritory();
                    puts("");
                    puts("");
                    if(i_won == 1){
                        printWinningTrophy();
                        puts("");
                        puts("");
                    }
                }
                return true;
            }
        }
        return false;
    }while(true);
}

//[4] POSIZIONA CARRI ARMATI SU TERRITORIO
static bool posizionaCarriArmatiSuTerritorio(){
    char nomeTerritorio[VARCHAR45];
    int num_army;
    do{
        memset(&nomeTerritorio, 0, sizeof(nomeTerritorio));
        memset(&num_army, 0, sizeof(num_army));
```

```
        if(getPositionArmyInfo(nomeTerritorio, &num_army)){
            //conversione nome territorio in uppercase
            toUpperCase(nomeTerritorio);
            if(placeArmyOnTerritory(g_username, &num_army, nomeTerritorio)){
                printSuccess("-- POSIZIONAMENTO CARRI ARMATI EFFETTUATO CON SUCCESSO --");
                return true;
            }
        }
        return false;
    }while(true);
}

//[5] SPOSTA CARRI ARMATI TRA TERRITORI
static bool spostaCarriArmatiTraTerritori(){
    char nomeTerritorio1[VARCHAR45];
    char nomeTerritorio2[VARCHAR45];
    int num_army;
    do{
        memset(&nomeTerritorio1, 0, sizeof(nomeTerritorio1));
        memset(&nomeTerritorio2, 0, sizeof(nomeTerritorio2));
        memset(&num_army, 0, sizeof(num_army));
        if(getMoveArmyInfo(&num_army, nomeTerritorio1, nomeTerritorio2)){
            //conversione nome territorio in uppercase
            toUpperCase(nomeTerritorio1);
            toUpperCase(nomeTerritorio2);
            if(moveArmyBetweenTerritories(g_username, &num_army, nomeTerritorio1,
nomeTerritorio2)){
                printSuccess("-- SPOSTAMENTO CARRI ARMATI EFFETTUATO CON SUCCESSO --");
                return true;
            }
        }
        return false;
    }while(true);
}

//[6] MOSTRA ADIACENZE TERRITORIO
static bool mostraAdiacenzeTerritorio(){
    char nomeTerritorio[VARCHAR45];
    do{
        memset(&nomeTerritorio, 0, sizeof(nomeTerritorio));
        if (getInput("Nome territorio di cui mostrare le adiacenze:\n>> ", nomeTerritorio, 46))
{
            //conversione nome territorio in uppercase
            toUpperCase(nomeTerritorio);
            printSuccess("-- GENERAZIONE LISTA TERRITORI ADIACENTI INIZIATA --");
            if(showAdjacentTerritories(nomeTerritorio)){
                printSuccess("-- GENERAZIONE LISTA TERRITORI ADIACENTI TERMINATA --");
                return true;
            }else{
                return false;
            }
        }
        printError("Errore Lettura Territorio Di Cui Mostrare Le Adiacenze") ;
```

```
        return false;
    }while(true);
}

//[7-3] VISUALIZZA STORICO PARTITE
static bool visualizzaStoricoPartite(){
    printSuccess("-- GENERAZIONE STORICO PARTITE INIZIATA --");
    if(showMatchHistory(g_username)){
        printSuccess("-- GENERAZIONE STORICO PARTITE TERMINATA --");
        return true;
    }else{
        return false;
    }
}

//[8] ABBANDONA PARTITA (ESCI DALLA STANZA)
static bool abbandonaPartita(){
    return leaveMatch(g_username);
}

void giocatoreInStanzaController(char *nomeStanza, int idPartita){
    bool prev_error = false;
clean_up:
    clearScreen();
    showAppTitle();
    printf("\t\t\t\t|_____________________|\n");
    printf("\t\t\t\t|   SEI IN UNA STANZA DI GIOCO    |   NomeStanza: %s\n",nomeStanza);
    printf("\t\t\t\t|_________________________________|   CodicePartita: %d\n",idPartita);
    int input;
    int failed_attempts = 0;
    while(true)
    {
        if(prev_error){
            printError("\t Ripristino menu dopo troppi tentativi errati\n\t Stai più attento!");
            prev_error = false;
        }
        if(failed_attempts == 5){
            prev_error = true;
            goto clean_up;
        }
        input = promptMenuStanza();
        switch (input)
        {
        case 1:
            if(!vediStatoDiGioco()){
                failed_attempts ++;
            }
            break;
        case 2:
            if(!saltaTurno()){
                failed_attempts ++;
            }
            break;
```

```
        case 3:
            if(!attaccaTerritorio()){
                failed_attempts ++;
            }
            break;
        case 4:
            if(!posizionaCarriArmatiSuTerritorio()){
                failed_attempts ++;
            }
            break;
        case 5:
            if(!spostaCarriArmatiTraTerritori()){
                failed_attempts ++;
            }
            break;
        case 6:
            if(!mostraAdiacenzeTerritorio()){
                failed_attempts ++;
            }
            break;
        case 7:
            if(!visualizzaStoricoPartite()){
                failed_attempts ++;
            }
            break;
        case 8:
            if(abbandonaPartita()){
                return;
            }else{
                failed_attempts ++;
            }
            break;
        case 9:
            goto clean_up;
        default:
            printError("Scegli tra le opzioni proposte!");
            failed_attempts ++;
            break;
        }
    }

}

//[2] MOSTRA STANZE DISPONIBILI
static bool mostraStanzeDisponibili(){
    printSuccess("-- GENERAZIONE LISTA STANZE DISPONIBILI INIZIATA --");
    if(showAvailableGameRooms()){
        printSuccess("-- GENERAZIONE LISTA STANZE DISPONIBILI TERMINATA --");
        return true;
    }else{
        return false;
    }
}
```

```
//[1] ENTRA IN UNA STANZA DI GIOCO
static bool entraInStanza(){
    char nomeStanza[VARCHAR45];
    char input;
    int idPartita;
    do {
        memset(&nomeStanza, 0, sizeof(nomeStanza));
        if(getInput("Inserire il nome della stanza di gioco in cui si vuole entrare:\n>> ",
nomeStanza, VARCHAR45)){
            if(enterInGameRoom(g_username, nomeStanza, &idPartita)){
                giocatoreInStanzaController(nomeStanza, idPartita);
                return true;
            }else{
                int fail = 0;
                do{
                    if(fail == 3){
                        break;
                    }
                    if(getInput("Vuoi vedere la lista delle stanze disponibili? (s/n): ",
&input, 2)){
                        if(input == 's'){
                            mostraStanzeDisponibili();
                            break;
                        }else if(input == 'n'){
                            break;
                        }else{
                            printError("Selezione errata, scegliere tra 's' e 'n'");
                            fail ++;
                        }
                    }else{
                        fail ++;
                    }
                }while(true);
            }
        }
        return false;
    } while (true) ;
}

void giocatoreController(char* username){
    g_username = username;
    bool prev_error = false;
clean_up:
    clearScreen();
    showAppTitle();
    puts("\t\t\t\t|‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|");
    puts("\t\t\t\t|         AREA GIOCATORI          |");
    puts("\t\t\t\t|_________________________________|");
    printf("\t\t\t\t     Bentornat* %s.\n\n",username);
    int input;
    int failed_attempts = 0;
```

```
    while(true)
    {
        if(prev_error){
            printError("\t Ripristino menu dopo troppi tentativi errati\n\t Stai più attento!");
            prev_error = false;
        }
        if(failed_attempts == 5){
            prev_error = true;
            goto clean_up;
        }
        input = promptMenuGiocatore();
        switch (input)
        {
        case 1:
            if(!entraInStanza()){
                failed_attempts ++;
                break;
            }
            goto clean_up;
        case 2:
            if(!mostraStanzeDisponibili()){
                failed_attempts ++;
            }
            break;
        case 3:
            if(!visualizzaStoricoPartite()){
                failed_attempts ++;
            }
            break;
        case 4:
            return;
        case 5:
            goto clean_up;
        default:
            printError("Scegli tra le opzioni proposte!");
            failed_attempts ++;
            break;
        }
    }

}
```

giocatoreController.h

```
#include "../engineering/inout.h"
#include "../database/dbUtils.h"
#include "../database/db.h"
#include "../view/viewUtils.h"
#include "../view/menuGiocatore.h"

void giocatoreController(char* username);
```

moderatoreController.c

```
#include "moderatoreController.h"

static bool creaStanza(){
    char nomeStanza[VARCHAR45];
    do {
        memset(&nomeStanza, 0, sizeof(nomeStanza));
        if(getInput("Inserire il nome per la nuova stanza di gioco:\n>> ", nomeStanza,
VARCHAR45)){
            if(createNewGameRoom(nomeStanza)){
                printSuccess("Nuova Stanza Creata con successo!");
                return true;
            }
        }else{
            printError("Nome stanza di gioco non valido");
        }
        return false;
    } while (true) ;
}

bool registraNuovoModeratore(){
    Credentials creds;
    do {
        memset(&creds, 0, sizeof(Credentials));
        if(promptLoginAndRegistration(&creds)){
            if(registerNewModder(creds)){
                printSuccess("Nuovo moderatore registrato correttamente");
                return true;
            }
        }
        return false;
    } while (true) ;
}

static bool vediReport(){
    do {
        printSuccess("-- GENERAZIONE REPORT INIZIATA --");
        if(retrieveReport()){
            printSuccess("-- GENERAZIONE REPORT TERMINATA --");
            return true;
        }
        return false;
    } while (true) ;
}



void moderatoreController(char *username){
    bool prev_error = false;
clean_up:
    clearScreen();
    showAppTitle();
    puts("\t\t\t\t|──────────────────────|");
```

```
    puts("\t\t\t\t|         AREA MODERATORI         |");
    puts("\t\t\t\t|_________________________________|");
    printf("\t\t\t\t     Bentornat* %s.\n\n",username);
    int input;
    int failed_attempts = 0;
    while(true)
    {
        if(prev_error){
            printError("\t Ripristino menu dopo troppi tentativi errati\n\t Stai più attento!");
            prev_error = false;
        }
        if(failed_attempts == 5){
            prev_error = true;
            goto clean_up;
        }
        input = promptMenuModeratore();
        switch (input)
        {
        case 1:
            if(!creaStanza()){
                failed_attempts ++;
            }
            break;
        case 2:
            if(!vediReport()){
                failed_attempts ++;
            }
            break;
        case 3:
            if(!registraNuovoModeratore()){
                failed_attempts ++;
            }
            break;
        case 4:
            return;
        case 5:
            goto clean_up;
        default:
            printError("Scegli tra le opzioni proposte!");
            failed_attempts ++;
            break;
        }
    }

}
```

moderatoreController.h

```
#include "../engineering/inout.h"
#include "../database/dbUtils.h"
#include "../database/db.h"
#include "../view/viewUtils.h"
#include "../view/menuModeratore.h"
```

```
void moderatoreController(char *username);
```

./code/database

db.c

```
#include "db.h"

bool moveArmyBetweenTerritories(char *username, int *num_carri, char *terr1, char *terr2){
	MYSQL_STMT* prepared_stmt;
	MYSQL_BIND param[4];

	// Prepare stored procedure call
	if (!setup_prepared_stmt(&prepared_stmt, "call sposta_carriArmati_tra_territori(?, ?, ?,
?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize prepared
statement for procedure: sposta_carriArmati_tra_territori", false);
		goto err1;
	}

	// Prepare parameters
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[0].buffer = username;
	param[0].buffer_length = strlen(username);

	param[1].buffer_type = MYSQL_TYPE_LONG; // IN
	param[1].buffer = num_carri;
	param[1].buffer_length = sizeof(num_carri);

	param[2].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[2].buffer = terr1;
	param[2].buffer_length = strlen(terr1);

	param[3].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[3].buffer = terr2;
	param[3].buffer_length = strlen(terr2);

	// Binding
	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters in
procedure: sposta_carriArmati_tra_territori", true);
		goto err;
	}

	// Execution
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "Error in execution for procedure:
sposta_carriArmati_tra_territori");
		goto err;
	}
```

```
        mysql_stmt_close(prepared_stmt);
        return true;
err:
        mysql_stmt_close(prepared_stmt);
err1:
        return false;
}


bool showAdjacentTerritories(char *terr){
    MYSQL_STMT* prepared_stmt;
        MYSQL_BIND param[1];

        // Prepare stored procedure call
        if (!setup_prepared_stmt(&prepared_stmt, "call mostra_adiacenze_territorio(?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize prepared
statement for procedure: mostra_adiacenze_territorio", false);
                goto err1;
        }

        // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
        param[0].buffer = terr;
        param[0].buffer_length = strlen(terr);

        // Binding
        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters in
procedure: mostra_adiacenze_territorio", true);
                goto err;
        }

        // Execution
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "Error in execution for procedure:
mostra_adiacenze_territorio");
                goto err;
        }

        // Dump of the result set
        dump_result_set(conn, prepared_stmt, "\n\tTERRITORI ADIACENTI\n");
        mysql_stmt_next_result(prepared_stmt);
        mysql_stmt_close(prepared_stmt);
        return true;
err:
        mysql_stmt_close(prepared_stmt);
err1:
        return false;
}
```

```
bool placeArmyOnTerritory(char *username, int *num_carri, char *terr){
	MYSQL_STMT* prepared_stmt;
	MYSQL_BIND param[3];

	// Prepare stored procedure call
	if (!setup_prepared_stmt(&prepared_stmt, "call posiziona_carriArmati_su_territorio(?, ?,
?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize prepared
statement for procedure: posiziona_carriArmati_su_territorio", false);
		goto err1;
	}

	// Prepare parameters
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[0].buffer = username;
	param[0].buffer_length = strlen(username);

	param[1].buffer_type = MYSQL_TYPE_LONG; // IN
	param[1].buffer = num_carri;
	param[1].buffer_length = sizeof(num_carri);

	param[2].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[2].buffer = terr;
	param[2].buffer_length = strlen(terr);

	// Binding
	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters in
procedure: posiziona_carriArmati_su_territorio", true);
		goto err;
	}

	// Execution
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "Error in execution for procedure:
posiziona_carriArmati_su_territorio");
		goto err;
	}

	mysql_stmt_close(prepared_stmt);
	return true;
err:
	mysql_stmt_close(prepared_stmt);
err1:
	return false;
}

bool attackTerritory(char *username, int *num_carri, char *terr1, char *terr2, int
*num_army_loss_attack, int *num_army_loss_defense,int *is_conquered, int *i_won){
	MYSQL_STMT* prepared_stmt;
	MYSQL_BIND param[8];
```

```
	// Prepare stored procedure call
	if (!setup_prepared_stmt(&prepared_stmt, "call attacca_territorio(?, ?, ?, ?, ?, ?, ?, 
?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize prepared 
statement for procedure: attacca_territorio", false);
		goto err1;
	}

	// Prepare parameters
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[0].buffer = username;
	param[0].buffer_length = strlen(username);

	param[1].buffer_type = MYSQL_TYPE_LONG; // IN
	param[1].buffer = num_carri;
	param[1].buffer_length = sizeof(num_carri);

	param[2].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[2].buffer = terr1;
	param[2].buffer_length = strlen(terr1);

	param[3].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[3].buffer = terr2;
	param[3].buffer_length = strlen(terr2);

	param[4].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[4].buffer = num_army_loss_attack;
	param[4].buffer_length = sizeof(num_army_loss_attack);

	param[5].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[5].buffer = num_army_loss_defense;
	param[5].buffer_length = sizeof(num_army_loss_defense);

	param[6].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[6].buffer = is_conquered;
	param[6].buffer_length = sizeof(is_conquered);

	param[7].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[7].buffer = i_won;
	param[7].buffer_length = sizeof(i_won);

	// Binding
	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters in 
procedure: attacca_territorio", true);
		goto err;
	}

	// Execution
	if (mysql_stmt_execute(prepared_stmt) != 0) {
```

```
		print_stmt_error(prepared_stmt, "Error in execution for procedure:
attacca_territorio");
		goto err;
	}

	// Prepare output params
	memset(param, 0, sizeof(param));
	param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[0].buffer = num_army_loss_attack;
	param[0].buffer_length = sizeof(num_army_loss_attack);

	param[1].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[1].buffer = num_army_loss_defense;
	param[1].buffer_length = sizeof(num_army_loss_defense);

	param[2].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[2].buffer = is_conquered;
	param[2].buffer_length = sizeof(is_conquered);

	param[3].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[3].buffer = i_won;
	param[3].buffer_length = sizeof(i_won);

	// Binding res
	if (mysql_stmt_bind_result(prepared_stmt,param)){
		print_stmt_error(prepared_stmt, "Could not retrieve output in procedure:
attacca_territorio");
		goto err;
	}

	// Retrieve output parameter
	if (mysql_stmt_fetch(prepared_stmt)){
		print_stmt_error(prepared_stmt, "Could not buffer result in procedure:
attacca_territorio");
		goto err;
	}

	mysql_stmt_close(prepared_stmt);
	return true;
err:
	mysql_stmt_close(prepared_stmt);
err1:
	return false;
}

bool skipTurn(char *username){
	MYSQL_STMT* prepared_stmt;
	MYSQL_BIND param[1];

	// Prepare stored procedure call
	if (!setup_prepared_stmt(&prepared_stmt, "call salta_turno(?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize prepared
statement for procedure: salta_turno", false);
```

```
                goto err1;
        }

        // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
        param[0].buffer = username;
        param[0].buffer_length = strlen(username);

        // Binding
        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters in
procedure: salta_turno", true);
                goto err;
        }

        // Execution
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "Error in execution for procedure: salta_turno");
                goto err;
        }

        mysql_stmt_close(prepared_stmt);
        return true;
err:
        mysql_stmt_close(prepared_stmt);
err1:
        return false;
}

bool showMatchStatus(char *username){
        MYSQL_STMT* prepared_stmt;
        MYSQL_BIND param[1];

        // Prepare stored procedure call
        if (!setup_prepared_stmt(&prepared_stmt, "call vedi_stato_di_gioco(?)", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize prepared
statement for procedure: vedi_stato_di_gioco", false);
                goto err1;
        }

        // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
        param[0].buffer = username;
        param[0].buffer_length = strlen(username);

        // Binding
        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters in
procedure: vedi_stato_di_gioco", true);
```

```
                goto err;
        }

        // Execution
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "Error in execution for procedure:
vedi_stato_di_gioco");
                goto err;
        }

        // Dump of the result set
        dump_result_set(conn, prepared_stmt, "\n\tSTATO DI GIOCO\n");
        mysql_stmt_next_result(prepared_stmt);
        dump_result_set(conn, prepared_stmt, "\n\tARMATE GIOCATORI\n");
        mysql_stmt_next_result(prepared_stmt);
        mysql_stmt_close(prepared_stmt);
        return true;
err:
        mysql_stmt_close(prepared_stmt);
err1:
        return false;
}

bool showAvailableGameRooms(){
        MYSQL_STMT* prepared_stmt;

        // Prepare stored procedure call
        if (!setup_prepared_stmt(&prepared_stmt, "call ottieni_lista_stanze_disponibili()",
conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize prepared
statement for procedure: ottieni_lista_stanze_disponibili", false);
                goto err1;
        }

        // Execution
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "Error in execution for procedure:
ottieni_lista_stanze_disponibili");
                goto err;
        }

        // Dump of the result set
        dump_result_set(conn, prepared_stmt, "\n\tPARTITE DISPONIBILI\n");
        mysql_stmt_next_result(prepared_stmt);
        mysql_stmt_close(prepared_stmt);
        return true;
err:
        mysql_stmt_close(prepared_stmt);
err1:
        return false;
}

bool leaveMatch(char *username){
```

```
	MYSQL_STMT* prepared_stmt;
	MYSQL_BIND param[1];

	// Prepare stored procedure call
	if (!setup_prepared_stmt(&prepared_stmt, "call rimuovi_partecipante(?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize prepared
statement for procedure: rimuovi_partecipante", false);
		goto err1;
	}

	// Prepare parameters
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[0].buffer = username;
	param[0].buffer_length = strlen(username);

	// Binding
	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters in
procedure: rimuovi_partecipante", true);
		goto err;
	}

	// Execution
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "Error in execution for procedure:
rimuovi_partecipante");
		goto err;
	}

	mysql_stmt_close(prepared_stmt);
	return true;
err:
	mysql_stmt_close(prepared_stmt);
err1:
	return false;
}

bool showMatchHistory(char *username){
	MYSQL_STMT* prepared_stmt;
	MYSQL_BIND param[1];

	// Prepare stored procedure call
	if (!setup_prepared_stmt(&prepared_stmt, "call vedi_storico_partite(?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize prepared
statement for procedure: vedi_storico_partite", false);
		goto err1;
	}

	// Prepare parameters
	memset(param, 0, sizeof(param));
```

```
	param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[0].buffer = username;
	param[0].buffer_length = strlen(username);

	// Binding
	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters in
procedure: vedi_storico_partite", true);
		goto err;
	}

	// Execution
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "Error in execution for procedure:
vedi_storico_partite");
		goto err;
	}

	// Dump of the result set
	dump_result_set(conn, prepared_stmt, "\n\tSTORICO PARTITE\n");
	mysql_stmt_next_result(prepared_stmt);
	mysql_stmt_close(prepared_stmt);
	return true;
err:
	mysql_stmt_close(prepared_stmt);
err1:
	return false;
}

bool enterInGameRoom(char *username, char *nomeStanza, int *idPartita){
	MYSQL_STMT* prepared_stmt;
	MYSQL_BIND param[3];

	// Prepare stored procedure call
	if (!setup_prepared_stmt(&prepared_stmt, "call aggiungi_partecipante(?, ?, ?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize prepared
statement for procedure: aggiungi_partecipante", false);
		goto err1;
	}

	// Prepare parameters
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[0].buffer = username;
	param[0].buffer_length = strlen(username);

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[1].buffer = nomeStanza;
	param[1].buffer_length = strlen(nomeStanza);

	param[2].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[2].buffer = idPartita;
```

```
	param[2].buffer_length = sizeof(idPartita);

	// Binding
	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters in
procedure: aggiungi_partecipante", true);
		goto err;
	}

	// Execution
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "Error in execution for procedure:
aggiungi_partecipante");
		goto err;
	}

	// Prepare output params
	memset(param, 0, sizeof(param));
	param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
	param[0].buffer = idPartita;
	param[0].buffer_length = sizeof(idPartita);

	// Binding res
	if (mysql_stmt_bind_result(prepared_stmt,param)){
		print_stmt_error(prepared_stmt, "Could not retrieve output in procedure:
aggiungi_partecipante");
		goto err;
	}

	// Retrieve output parameter
	if (mysql_stmt_fetch(prepared_stmt)){
		print_stmt_error(prepared_stmt, "Could not buffer result in procedure:
aggiungi_partecipante");
		goto err;
	}
	mysql_stmt_close(prepared_stmt);
	return true;
err:
	mysql_stmt_close(prepared_stmt);
err1:
	return false;
}

bool createNewGameRoom(char *nomeStanza){
	MYSQL_STMT* prepared_stmt;
	MYSQL_BIND param[1];

	// Prepare stored procedure call
	if (!setup_prepared_stmt(&prepared_stmt, "call crea_nuova_stanza(?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize prepared
statement for procedure: crea_nuova_stanza", false);
		goto err1;
	}
```

```
        // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
        param[0].buffer = nomeStanza;
        param[0].buffer_length = strlen(nomeStanza);

        // Binding
        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters in
procedure: crea_nuova_stanza", true);
                goto err;
        }

        // Execution
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "Error in execution for procedure:
crea_nuova_stanza");
                goto err;
        }

        mysql_stmt_close(prepared_stmt);
        return true;
err:
        mysql_stmt_close(prepared_stmt);
err1:
        return false;

}

bool retrieveReport(){
        MYSQL_STMT* prepared_stmt;

        // Prepare stored procedure call
        if (!setup_prepared_stmt(&prepared_stmt, "call report_moderatori()", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize prepared
statement for procedure: report_moderatori", false);
                goto err1;
        }

        // Execution
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "Error in execution for procedure:
report_moderatori");
                goto err;
        }

        // Dump of the result set
        dump_result_set(conn, prepared_stmt, "\n\tREPORT MODERATORI\n");
        mysql_stmt_next_result(prepared_stmt);
        dump_result_set(conn, prepared_stmt, "");
        mysql_stmt_next_result(prepared_stmt);
```

```
	dump_result_set(conn, prepared_stmt, "");
	mysql_stmt_next_result(prepared_stmt);
	mysql_stmt_close(prepared_stmt);
	return true;
err:
	mysql_stmt_close(prepared_stmt);
err1:
	return false;
}

bool registerNewModder(Credentials creds){
	MYSQL_STMT* prepared_stmt;
	MYSQL_BIND param[2];

	// Prepare stored procedure call
	if (!setup_prepared_stmt(&prepared_stmt, "call registra_nuovo_moderatore(?, ?)", conn))
{
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize prepared
statement for procedure: registra_nuovo_moderatore", false);
		goto err1;
	}
	// Prepare parameters
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[0].buffer = creds.username;
	param[0].buffer_length = strlen(creds.username);

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[1].buffer = creds.password;
	param[1].buffer_length = strlen(creds.password);

	// Binding
	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters in
procedure: registra_nuovo_moderatore", true);
		goto err;
	}

	// Execution
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "Error in execution for procedure:
registra_nuovo_moderatore");
		goto err;
	}

	mysql_stmt_close(prepared_stmt);
	return true;
err:
	mysql_stmt_close(prepared_stmt);
err1:
	return false;
}
```

```
bool registerNewPlayer(Credentials creds){
	MYSQL_STMT* prepared_stmt;
	MYSQL_BIND param[2];

	// Prepare stored procedure call
	if (!setup_prepared_stmt(&prepared_stmt, "call registra_nuovo_giocatore(?, ?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize prepared
statement for procedure: registra_nuovo_giocatore", false);
		goto err1;
	}

	// Prepare parameters
	memset(param, 0, sizeof(param));

	param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[0].buffer = creds.username;
	param[0].buffer_length = strlen(creds.username);

	param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
	param[1].buffer = creds.password;
	param[1].buffer_length = strlen(creds.password);

	// Binding
	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters in
procedure: registra_nuovo_giocatore", true);
		goto err;
	}

	// Execution
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "Error in execution for procedure:
registra_nuovo_giocatore");
		goto err;
	}

	mysql_stmt_close(prepared_stmt);
	return true;
err:
	mysql_stmt_close(prepared_stmt);
err1:
	return false;
}

Role logAsUser(Credentials creds) {
	MYSQL_STMT* login_procedure;

	MYSQL_BIND param[3]; // Used both for input and output
	int role = 0;

	if (!setup_prepared_stmt(&login_procedure, "call login(?, ?, ?)", conn)) {
		print_stmt_error(login_procedure, "Unable to initialize prepared statement for
```

```
procedure: login");
                goto err1;
        }

        // Prepare parameters
        memset(param, 0, sizeof(param));

        param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
        param[0].buffer = creds.username;
        param[0].buffer_length = strlen(creds.username);

        param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
        param[1].buffer = creds.password;
        param[1].buffer_length = strlen(creds.password);

        param[2].buffer_type = MYSQL_TYPE_LONG; // OUT
        param[2].buffer = &role;
        param[2].buffer_length = sizeof(role);

        // Binding
        if (mysql_stmt_bind_param(login_procedure, param) != 0) { // Note _param
                print_stmt_error(login_procedure, "Could not bind parameters in procedure:
login");
                goto err;
        }

        // Execution
        if (mysql_stmt_execute(login_procedure) != 0) {
                print_stmt_error(login_procedure, "Error in execution for procedure: login");
                goto err;
        }

        // Prepare output parameters
        memset(param, 0, sizeof(param));
        param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
        param[0].buffer = &role;
        param[0].buffer_length = sizeof(role);

        if (mysql_stmt_bind_result(login_procedure, param)) {
                print_stmt_error(login_procedure, "Could not retrieve output in procedure:
aggiungi_partecipante");
                goto err;
        }

        // Retrieve output parameter
        if (mysql_stmt_fetch(login_procedure)) {
                print_stmt_error(login_procedure, "Could not buffer results in procedure:
aggiungi_partecipante");
                goto err;
        }

        mysql_stmt_close(login_procedure);
        return role;
```

```
err:
	mysql_stmt_close(login_procedure);
err1:
	return LOGIN;
}
```

db.h

```
#include <mysql/mysql.h>
#include "../controller/loginController.h"
#include "../engineering/connector.h"
#include <string.h>
#include <stdio.h>
#include "../engineering/inout.h"
#include "../model/credentials.h"
#include "dbUtils.h"
#include "../view/viewUtils.h"

#define VARCHAR45 45 + 1

Role logAsUser(Credentials creds);

bool registerNewPlayer(Credentials creds);

bool registerNewModder(Credentials creds);

bool createNewGameRoom(char *nomeStanza);

bool showMatchHistory(char *username);

bool showAvailableGameRooms();

bool retrieveReport();

bool enterInGameRoom(char *username, char *nomeStanza,int *idPartita);

bool skipTurn(char *username);

bool showMatchStatus(char *username);

bool leaveMatch(char *username);

bool attackTerritory(char *username, int *num_carri, char *terr1, char *terr2, int
*num_army_loss_attack, int *num_army_loss_defense,int *is_conquered, int *i_won);

bool placeArmyOnTerritory(char *username, int *num_carri, char *terr);

bool moveArmyBetweenTerritories(char *username, int *num_carri, char *terr1, char *terr2);

bool showAdjacentTerritories(char *terr);
```

dbUtils.c

```c
#include "dbUtils.h"

void print_stmt_error(MYSQL_STMT* stmt, char* message)
{
	puts("");
    printf("\033[41m");
	printf("%s\n", message);
	if (stmt != NULL) {
	    printf("Error %u (%s): %s",
			mysql_stmt_errno(stmt),
			mysql_stmt_sqlstate(stmt),
			mysql_stmt_error(stmt));
	}
    printf("\033[0m");
	puts("");
	puts("");
}


void print_sql_error(MYSQL* conn, char* message)
{
	puts("");
    printf("\033[41m");
	printf("%s\n", message);
	if (conn != NULL) {
#if MYSQL_VERSION_ID >= 40101
		printf("Error %u (%s): %s",
			mysql_errno(conn), mysql_sqlstate(conn), mysql_error(conn));
#else
		printf("Error %u: %s",
			mysql_errno(conn), mysql_error(conn));
#endif
	}
    printf("\033[0m");
	puts("");
	puts("");
}

bool setup_prepared_stmt(MYSQL_STMT** stmt, char* statement, MYSQL* conn)
{
	bool update_length = true;

	*stmt = mysql_stmt_init(conn);
	if (*stmt == NULL)
	{
		print_sql_error(conn, "Could not initialize statement handler");
		return false;
	}

	if (mysql_stmt_prepare(*stmt, statement, strlen(statement)) != 0) {
		print_stmt_error(*stmt, "Could not prepare statement");
		return false;
```

```
        }

        mysql_stmt_attr_set(*stmt, STMT_ATTR_UPDATE_MAX_LENGTH, &update_length);

        return true;
}

void finish_with_error(MYSQL* conn, char* message)
{
        print_sql_error(conn, message);
        mysql_close(conn);
        exit(EXIT_FAILURE);
}

void finish_with_stmt_error(MYSQL* conn, MYSQL_STMT* stmt, char* message, bool close_stmt)
{
        print_stmt_error(stmt, message);
        if (close_stmt)       mysql_stmt_close(stmt);
        mysql_close(conn);
        exit(EXIT_FAILURE);
}

static void print_dashes(MYSQL_RES* res_set)
{
        MYSQL_FIELD* field;
        unsigned int i, j;

        mysql_field_seek(res_set, 0);
        putchar('+');
        for (i = 0; i < mysql_num_fields(res_set); i++) {
                field = mysql_fetch_field(res_set);
                for (j = 0; j < field->max_length + 2; j++)
                        putchar('-');
                putchar('+');
        }
        putchar('\n');
}

static void dump_result_set_header(MYSQL_RES* res_set)
{
        MYSQL_FIELD* field;
        unsigned long col_len;
        unsigned int i;

        /* determine column display widths -- requires result set to be */
        /* generated with mysql_store_result(), not mysql_use_result() */

        mysql_field_seek(res_set, 0);

        for (i = 0; i < mysql_num_fields(res_set); i++) {
                field = mysql_fetch_field(res_set);
                col_len = strlen(field->name);
```

```
                if (col_len < field->max_length)
                        col_len = field->max_length;
                if (col_len < 4 && !IS_NOT_NULL(field->flags))
                        col_len = 4; /* 4 = length of the word "NULL" */
                field->max_length = col_len; /* reset column info */
        }

        print_dashes(res_set);
        putchar('|');
        mysql_field_seek(res_set, 0);
        for (i = 0; i < mysql_num_fields(res_set); i++) {
                field = mysql_fetch_field(res_set);
                printf(" %-*s |", (int)field->max_length, field->name);
        }
        putchar('\n');

        print_dashes(res_set);
}

void dump_result_set(MYSQL* conn, MYSQL_STMT* stmt, char* title)
{
        int i;
        int status;
        int num_fields;       /* number of columns in result */
        MYSQL_FIELD* fields;  /* for result set metadata */
        MYSQL_BIND* rs_bind;  /* for output buffers */
        MYSQL_RES* rs_metadata;
        MYSQL_TIME* date;
        size_t attr_size;

        /* Prefetch the whole result set. This in conjunction with
         * STMT_ATTR_UPDATE_MAX_LENGTH set in `setup_prepared_stmt`
         * updates the result set metadata which are fetched in this
         * function, to allow to compute the actual max length of
         * the columns.
         */
        if (mysql_stmt_store_result(stmt)) {
                fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
                fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
                exit(0);
        }

        /* the column count is > 0 if there is a result set */
        /* 0 if the result is only the final status packet */
        num_fields = mysql_stmt_field_count(stmt);

        if (num_fields > 0) {
                /* there is a result set to fetch */
                printf("%s\n", title);

                if ((rs_metadata = mysql_stmt_result_metadata(stmt)) == NULL) {
                        finish_with_stmt_error(conn, stmt, "Unable to retrieve result metadata\n",
true);
```

```
		}

		dump_result_set_header(rs_metadata);

		fields = mysql_fetch_fields(rs_metadata);

		rs_bind = (MYSQL_BIND*)malloc(sizeof(MYSQL_BIND) * num_fields);
		if (!rs_bind) {
			finish_with_stmt_error(conn, stmt, "Cannot allocate output buffers\n",
true);
		}
		memset(rs_bind, 0, sizeof(MYSQL_BIND) * num_fields);

		/* set up and bind result set output buffers */
		for (i = 0; i < num_fields; ++i) {

			// Properly size the parameter buffer
			switch (fields[i].type) {
			case MYSQL_TYPE_DATE:
			case MYSQL_TYPE_TIMESTAMP:
			case MYSQL_TYPE_DATETIME:
			case MYSQL_TYPE_TIME:
				attr_size = sizeof(MYSQL_TIME);
				break;
			case MYSQL_TYPE_FLOAT:
				attr_size = sizeof(float);
				break;
			case MYSQL_TYPE_DOUBLE:
				attr_size = sizeof(double);
				break;
			case MYSQL_TYPE_TINY:
				attr_size = sizeof(signed char);
				break;
			case MYSQL_TYPE_SHORT:
			case MYSQL_TYPE_YEAR:
				attr_size = sizeof(short int);
				break;
			case MYSQL_TYPE_LONG:
			case MYSQL_TYPE_INT24:
				attr_size = sizeof(int);
				break;
			case MYSQL_TYPE_LONGLONG:
				attr_size = sizeof(long long int);
				break;
			default:
				attr_size = fields[i].max_length;
				break;
			}

			// Setup the binding for the current parameter
			rs_bind[i].buffer_type = fields[i].type;
			rs_bind[i].buffer = malloc(attr_size + 1);
			rs_bind[i].buffer_length = attr_size + 1;
```

```
                        if (rs_bind[i].buffer == NULL) {
                                finish_with_stmt_error(conn, stmt, "Cannot allocate output
buffers\n", true);
                        }
                }

                if (mysql_stmt_bind_result(stmt, rs_bind)) {
                        finish_with_stmt_error(conn, stmt, "Unable to bind output parameters\n",
true);
                }

                /* fetch and display result set rows */
                while (true) {
                        status = mysql_stmt_fetch(stmt);

                        if (status == 1 || status == MYSQL_NO_DATA)
                                break;

                        putchar('|');

                        for (i = 0; i < num_fields; i++) {

                                if (rs_bind[i].is_null_value) {
                                        printf(" %-*s |", (int)fields[i].max_length, "NULL");
                                        continue;
                                }

                                switch (rs_bind[i].buffer_type) {

                                case MYSQL_TYPE_VAR_STRING:
                                case MYSQL_TYPE_DATETIME:
                                        printf(" %-*s |", (int)fields[i].max_length,
(char*)rs_bind[i].buffer);
                                        break;

                                case MYSQL_TYPE_DATE:
                                case MYSQL_TYPE_TIMESTAMP:
                                        date = (MYSQL_TIME*)rs_bind[i].buffer;
                                        printf(" %d-%02d-%02d |", date->year, date->month,
date->day);
                                        break;

                                case MYSQL_TYPE_STRING:
                                        printf(" %-*s |", (int)fields[i].max_length,
(char*)rs_bind[i].buffer);
                                        break;

                                case MYSQL_TYPE_FLOAT:
                                case MYSQL_TYPE_DOUBLE:
                                        printf(" %.02f |", *(float*)rs_bind[i].buffer);
                                        break;
```

```
                                case MYSQL_TYPE_LONG:
                                case MYSQL_TYPE_SHORT:
                                case MYSQL_TYPE_TINY:
                                        printf(" %-*d |", (int)fields[i].max_length,
*(int*)rs_bind[i].buffer);
                                        break;

                                case MYSQL_TYPE_LONGLONG:
                                        printf(" %-*lld |",(int)fields[i].max_length, *(long long
int*)rs_bind[i].buffer);
                                        break;

                                case MYSQL_TYPE_NEWDECIMAL:
                                        printf(" %-*.02lf |", (int)fields[i].max_length,
*(float*)rs_bind[i].buffer);
                                        break;

                                default:
                                        printf("ERROR: Unhandled type (%d)\n",
rs_bind[i].buffer_type);
                                        abort();
                                }
                        }
                        putchar('\n');
                        print_dashes(rs_metadata);
                }

                mysql_free_result(rs_metadata); /* free metadata */

                /* free output buffers */
                for (i = 0; i < num_fields; i++) {
                        free(rs_bind[i].buffer);
                }
                free(rs_bind);
                puts("");
        }
}
```

dbUtils.h

```
#pragma once

#include <mysql/mysql.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "../engineering/inout.h"
#include "../view/viewUtils.h"

extern void print_sql_error(MYSQL* conn, char* message);

extern void print_stmt_error(MYSQL_STMT* stmt, char* message);
```

```
extern void finish_with_error(MYSQL* conn, char* message);

extern void finish_with_stmt_error(MYSQL* conn, MYSQL_STMT* stmt, char* message, bool
close_stmt);

extern bool setup_prepared_stmt(MYSQL_STMT** stmt, char* statement, MYSQL* conn);

extern void dump_result_set(MYSQL* conn, MYSQL_STMT* stmt, char* title);
```

./code/engineering

connector.c

```
#include "connector.h"

MYSQL *conn ;


bool connectToDatabase() {
    char *host = getenv(DB_HOST) ;
    char *port = getenv(DB_PORT) ;
    char *databaseName = getenv(DB_NAME) ;
    char *loginUser = getenv(DB_LOGIN_USER) ;
    char *loginPassword = getenv(DB_LOGIN_PASSWD) ;
    unsigned int timeout = 300;
       bool reconnect = true;

    if (host == NULL || port == NULL || loginUser == NULL || loginPassword == NULL ||
databaseName == NULL) {
        printError("Errore: Variabili d'ambiente non trovate") ;
        return false ;
    }

    conn = mysql_init(NULL) ;
    if (conn == NULL) {
        printError("Errore: Inizializzazione librerie non riuscita");
        return false ;
    }

    if (mysql_real_connect(
        conn,
        host,
        loginUser,
        loginPassword,
        databaseName,
        atoi(port),
        NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS | CLIENT_COMPRESS |
CLIENT_INTERACTIVE | CLIENT_REMEMBER_OPTIONS) == NULL) {
            print_sql_error(conn, "Errore SQL: Connessione al Database non riuscita\n") ;
            return false ;
    }
```

```
        if (mysql_options(conn, MYSQL_OPT_CONNECT_TIMEOUT, &timeout)) {
                print_sql_error(conn, "Errore SQL: [mysql_options] failed.");
        }

        if(mysql_options(conn, MYSQL_OPT_RECONNECT, &reconnect)) {
                print_sql_error(conn, "Errore SQL: [mysql_options] failed.");
        }

    return true ;
}
```

connector.h

```
#pragma once

#include <stdbool.h>
#include <stdio.h>
#include <mysql/mysql.h>
#include "../database/dbUtils.h"
#include "../view/viewUtils.h"
#include "inout.h"

#define DB_HOST "DB.HOST"
#define DB_PORT "DB.PORT"
#define DB_NAME "DB.NAME"

#define DB_LOGIN_USER "LOGIN.USER"
#define DB_LOGIN_PASSWD "LOGIN.PASSWD"

#define DB_MODERATORE_USER "MODERATORE.USER"
#define DB_MODERATORE_PASSWD "MODERATORE.PASSWD"

#define DB_GIOCATORE_USER "GIOCATORE.USER"
#define DB_GIOCATORE_PASSWD "GIOCATORE.PASSWD"

extern MYSQL *conn ;

bool connectToDatabase() ;
```

inout.c

```
#include "inout.h"

bool getInput(char *requestString, char *resultBuffer, int bufferSize) {

    printBoldRed(requestString) ;

    // +1 is for \n special char
    char inputBuffer[bufferSize + 1] ;

    //reading inputMaxSize - 1 chars \n included, if any
    if (fgets(inputBuffer, bufferSize + 1, stdin) == NULL) {
```

```
        printError("couldn't scan input") ;
        return false ;
    }

    //removing \n, if any
    for (int i = 0 ; i < (int) strlen(inputBuffer) ; i++) {
        if (inputBuffer[i] == '\n') {
            inputBuffer[i] = '\0' ;
        }
    }

    if ((int) strlen(inputBuffer) == bufferSize) {
        while(getchar() != '\n') ;
        printError("input exceeds max buffer size") ;
        return false ;
    }

    strcpy(resultBuffer, inputBuffer) ;

    //check for empty input
    if (strlen(resultBuffer) == 0) {
        return false ;
    }
    return true ;
}

bool getPositionArmyInfo(char *terr, int *num_carri){
    if (!getInput("Nome territorio su cui posizionare i carri armati:\n>> ", terr, 46)) {
        printError("Errore Lettura Territorio Su Cui Posizionare Carri Armati") ;
        return false ;
    }
    if (!getInteger("Numero di carri da posizionare:\n>> ", num_carri)) {
        printError("Errore Lettura Numero Carri Armati Da Posizionare") ;
        return false ;
    }
    return true ;
}

bool getAttackInfo(char *terr1, char *terr2, int *num_carri){
    if (!getInput("Nome territorio da cui far partire l'attacco:\n>> ", terr1, 46)) {
        printError("Errore Lettura Territorio Partenza Attacco") ;
        return false ;
    }
    if (!getInput("Nome territorio che si vuole attaccare:\n(NB: deve essere adiacente)\n>> ",
terr2, 46)) {
        printError("Errore Lettura Territorio Destinazione Attacco") ;
        return false ;
    }
    if (!getInteger("Numero di carri da schierare in attacco(1-3):\n>> ", num_carri)) {
        printError("Errore Lettura Numero Carri Armati In Attacco") ;
        return false ;
    }
    return true ;
```

```
}

bool getMoveArmyInfo(int *num_carri, char *terr1, char *terr2){
    if (!getInput("Nome territorio da cui prendere i carri armati per lo spostamento:\n>> ",
terr1, 46)) {
        printError("Errore Lettura Territorio Da Cui Spostare Carri Armati") ;
        return false ;
    }
    if (!getInput("Nome territorio su cui spostare i carri armati:\n(NB: deve essere
adiacente)\n>> ", terr2, 46)) {
        printError("Errore Lettura Territorio Su Cui Spostare Carri Armati") ;
        return false ;
    }
    if (!getInteger("Numero di carri da spostare:\n>> ", num_carri)) {
        printError("Errore Lettura Numero Carri Armati Da Spostare") ;
        return false ;
    }
    return true ;
}

bool getCredentials(Credentials *creds){
    if (!getInput("Username: ", creds->username, USERNAME_MAX_SIZE)) {
        printError("Errore Lettura Username") ;
        return false ;
    }
    if (!getInput("Password: ", creds->password, PASSWORD_MAX_SIZE)) {
        printError("Errore Lettura Password") ;
        return false ;
    }
    return true ;
}

bool getInteger(char* domanda, int *integerPtr) {
    char integerStringBuff[11 + 1] ;
    if (!getInput(domanda, integerStringBuff, 11 + 1)) {
        printError("Errore Inserimento Codice Numerico") ;
        return false ;
    }
    char *checkString = "\0" ;
    errno = 0;
    long longInput = strtol(integerStringBuff, &checkString, 10) ;
    if (errno != 0) {
        printError("Numero Non Valido") ;
        errno = 0 ;
        return false ;
    }
    if (*checkString != '\0') {
        printError("Numero Non Valido") ;
        errno = 0 ;
        return false ;
    } ;
    if (longInput > INT_MAX || longInput < INT_MIN) return false ;
    *integerPtr = (int) longInput ;
```

```
    return true ;
}

void toUpperCase(char *str){
    int i;
    for(i=0; str[i]!='\0'; i++)
    {
        if(str[i]>='a' && str[i]<='z')
        {
            str[i] = str[i] - 32;
        }
    }
}
```

inout.h

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <errno.h>
#include <ctype.h>
#include <stdbool.h>
#include "../model/credentials.h"
#include "../view/viewUtils.h"

void toUpperCase(char *str);

char multiChoice(char* domanda, char choices[], int num);

bool getInput(char *requestString, char *resultBuffer, int bufferSize);

bool getInteger(char* domanda, int *integerPtr);

bool getCredentials(Credentials *creds);

bool getAttackInfo(char *terr1, char *terr2, int *num_carri);

bool getMoveArmyInfo(int *num_carri, char *terr1, char *terr2);

bool getPositionArmyInfo(char *terr, int *num_carri);
```

./code/model

credentials.h

```
#pragma once

#define USERNAME_MAX_SIZE 32 + 1
#define PASSWORD_MAX_SIZE 45 + 1

typedef struct {
```

```
    char username[USERNAME_MAX_SIZE] ;
    char password[PASSWORD_MAX_SIZE] ;
} Credentials ;

typedef enum {
    MODERATORE = 0,
    GIOCATORE = 1,
    LOGIN = 2
} Role ;
```

./code/view

loginAndRegistrationView.c

```
#include "loginAndRegistrationView.h"

bool promptLoginAndRegistration(Credentials *creds) {
    printBoldRed("Inserire le credenziali\n");

    return getCredentials(creds);
}

int promptInitialMenu(){
    int input;
    printBoldRed("Cosa vuoi fare? \n");
    printBoldRed("[1] ");
    printf("LOGIN\n");
    printBoldRed("[2] ");
    printf("REGISTRAZIONE\n");
    printBoldRed("[3] ");
    printf("USCIRE\n");
    printBoldRed(">> ");
    if(!getInteger("", &input)){
            return -1;
    }
    return input;
}
```

loginAndRegistrationView.h

```
#pragma once

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../engineering/inout.h"
#include "viewUtils.h"

bool promptLoginAndRegistration(Credentials *creds);

int promptInitialMenu();
```

menuGiocatore.c

```c
#include "menuGiocatore.h"

int promptMenuGiocatore(){
    int input;
    printBoldRed("Cosa vuoi fare? \n");
    printBoldRed("[1] ");
    printf("ENTRA IN UNA STANZA DI GIOCO\n");
    printBoldRed("[2] ");
    printf("MOSTRA STANZE DISPONIBILI\n");
    printBoldRed("[3] ");
    printf("VISUALIZZA STORICO PARTITE\n");
    printBoldRed("[4] ");
    printf("TORNA AL MENU PRINCIPALE\n");
    printBoldRed("[5] ");
    printf("CLEAN SCREEN\n");
    printBoldRed(">> ");
    if(!getInteger("", &input)){
            return -1;
    }
    return input;
}

int promptMenuStanza(){
    int input;
    printBoldRed("Cosa vuoi fare? \n");
    printBoldRed("[1] ");
    printf("VEDI STATO DI GIOCO\n");
    printBoldRed("[2] ");
    printf("SALTA TURNO\n");
    printBoldRed("[3] ");
    printf("ATTACCA TERRITORIO\n");
    printBoldRed("[4] ");
    printf("POSIZIONA CARRI ARMATI SU TERRITORIO\n");
    printBoldRed("[5] ");
    printf("SPOSTA CARRI ARMATI TRA TERRITORI\n");
    printBoldRed("[6] ");
    printf("MOSTRA ADIACENZE TERRITORIO\n");
    printBoldRed("[7] ");
    printf("VISUALIZZA STORICO PARTITE\n");
    printBoldRed("[8] ");
    printf("ABBANDONA PARTITA (ESCI DALLA STANZA)\n");
    printBoldRed("[9] ");
    printf("CLEAN SCREEN\n");
    printBoldRed(">> ");
    if(!getInteger("", &input)){
            return -1;
    }
    return input;
}
```

menuGiocatore.h

```c
#include "../engineering/inout.h"
#include "viewUtils.h"

int promptMenuGiocatore();

int promptMenuStanza();
```

menuModeratore.c

```c
#include "menuModeratore.h"

int promptMenuModeratore(){
    int input;
    printBoldRed("Cosa vuoi fare? \n");
    printBoldRed("[1] ");
    printf("CREA NUOVA STANZA DI GIOCO\n");
    printBoldRed("[2] ");
    printf("VISUALIZZA REPORT\n");
    printBoldRed("[3] ");
    printf("REGISTRA NUOVO MODERATORE\n");
    printBoldRed("[4] ");
    printf("TORNA AL MENU PRINCIPALE\n");
    printBoldRed("[5] ");
    printf("CLEAN SCREEN\n");
    printBoldRed(">> ");
    if(!getInteger("", &input)){
            return -1;
    }
    return input;
}
```

menuModeratore.h

```c
#include "../engineering/inout.h"
#include "viewUtils.h"

int promptMenuModeratore();
```

viewUtils.c

```c
#include "viewUtils.h"

void clearScreen() {
    printf("\033[2J\033[H");
}

void printBoldRed(char *str){
    printf("\033[1;31m%s\033[0m",str);
}
```

```c
void printError (char *errorMessage){
    puts("");
    printf("\033[41m%s\033[0m",errorMessage);
    puts("");
    puts("");
}

void printSuccess (char *successMessage){
    puts("");
    printf("\033[42m%s\033[0m",successMessage);
    puts("");
    puts("");
}

void showAppTitle() {
    printf("\033[1;31m");
    puts("");
    puts("  $$$$$$$\\  $$\\           $$\\ $$\\                         ");
    puts("  $$  __$$\\ \\__|           \\__|$$ |                        ");
    puts("  $$ |  $$ |$$\\  $$$$$$$\\ $$\\ $$ |  $$\\  $$$$$$\\         ");
    puts("  $$$$$$$  |$$ |$$  _____|$$ |$$ | $$  |$$  __$$\\       ");
    puts("  $$  __$$< $$ |\\$$$$$$\\  $$ |$$$$$$  / $$ /  $$ |         ████████ ]▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▃
...☻   ...☻   ...☻");
    puts("  $$ |  $$ |$$ | \\____$$\\ $$ |$$  _$$<  $$ |  $$ |      ▂▃▄█████████▄▂▂▁");
    puts("  $$ |  $$ |$$ |$$$$$$$  |$$ |$$ | \\$$\\ \\$$$$$$  |
I█████████████████████].");
    puts("  \\__|  \\__|\\__|\\_______/ \\__|\\__|  \\__| \\______/
...◥⊙▲⊙▲⊙▲⊙▲⊙▲⊙▲⊙▲⊙▲⊙▲⊙◤...          NO WAR :)");
    puts("\tBy Edoardo Manenti [m:0278821]");
    puts("");
    printf("\033[0m");
}

void printWinningTrophy() {
    puts("\n");
    printf("\033[43m%s\033[0m","\t\t\t\t        CONGRATULAZIONI!        ");puts("");
    printf("\033[43m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⡟⠛⢻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡟⠛⢻⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[43m%s\033[0m","\t\t\t\t ⣿⣿⣿⡏ ⣠⡞ ⢸⣧            ⣼⡇ ⢳⡄⠘⣿⣿⣿ ");puts("");
    printf("\033[43m%s\033[0m","\t\t\t\t ⣿⣿⣿⡇⠘⣿⣦⣿⣷          ⣴⣿⣶⣿⡏⢸⣿⣿⣿ ");puts("");
    printf("\033[43m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⡀⠹⣿⣿⣿⣧        ⣼⣿⣿⣿⠟⢀⣿⣿⣿⣿ ");puts("");
    printf("\033[43m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⣿⣆⠉⠻⣿⣿⣦      ⣴⣿⡿⠋⣠⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[43m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⣿⣿⣷⣤⣀⡛⠿⣷⡀  ⢀⣾⡟⠛⣀⣤⣾⣿⣿⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[43m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣶⣶⣶⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[43m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣷⣤⣤⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[43m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡟   ⢻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[43m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠃    ⠘⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[43m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡟⠛⠛⠛⠛⠛⠛⢻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[43m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠛⠛⠛⠛⠛⠛⠛⠛⠛⠛⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[43m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿ ⣴⣶⣶⣶⣶⣶⣶⣦ ⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[43m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿ ⢿⣿⣿⣿⣿⣿⣿⡿ ⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[43m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣤⣤⣤⣤⣤⣤⣤⣤⣤⣤⣤⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[43m%s\033[0m","\t\t\t\t          HAI VINTO!          ");puts("");
    puts("\n");
```

```
}

void printConqueredTerritory() {
    puts("\n");
    printf("\033[44m%s\033[0m","\t\t\t\t        CONGRATULAZIONI!        ");puts("");
    printf("\033[44m%s\033[0m","\t\t\t\t ⣿⡏⢹⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡟⢹⣿ ");puts("");
    printf("\033[44m%s\033[0m","\t\t\t\t ⣿⣿⣦⡀⠈⠙⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⠋⠁⢀⣴⣿⣿ ");puts("");
    printf("\033[44m%s\033[0m","\t\t\t\t ⣿⣿⣿⣦    ⠙⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⠋    ⣴⣿⣿⣿ ");puts("");
    printf("\033[44m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⣿⣦⡀    ⠈⢻⣿⣿⣿⣿⣿⣿⣿⡟⠉⠁    ⢀⣼⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[44m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⣿⣿⣿⣦  ⢻⣦⡈⠙⢿⣿⣿⡿⠋⢁⣴⡟  ⣠⣿⣿⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[44m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⣿⣿⣿⣿⣷⣄ ⠙⢷⣄ ⠙⣿⣄⣠⡾⠋ ⣠⣾⣿⣿⣿⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[44m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣷⣄ ⠙⢷⣄ ⠈⠻⣦⣀⣴⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[44m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⡟⢻⣿⣿⣿⣿⣿⡿⣦⡀ ⠙⢷⣄ ⠈⢻⣿⣿⣿⣿⡿⢻⣿⣿⣿⣿ ");puts("");
    printf("\033[44m%s\033[0m","\t\t\t\t ⣿⣿⣿⣦⡀⠈⢻⣿⡟⠁ ⣰⣿⣦⡀ ⠙⢷⣄ ⠈⢻⣿⡟⠁⢀⣴⣿⣿⣿ ");puts("");
    printf("\033[44m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⣿⣦⡀⠁ ⣠⡾⠋ ⢸⣿⣦⡀ ⠙⢷⣄ ⠈⢀⣴⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[44m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⣿⣿⣿⡟  ⠉  ⢀⣴⣿⣿⣿⣿⣦⡀  ⠉  ⢻⣿⣿⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[44m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⣿⣿⠏ ⣠⣾⣦⡀ ⠈⢻⣿⣿⣿⡿⠋ ⢀⣴⣿⣦ ⠙⣿⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[44m%s\033[0m","\t\t\t\t ⣿⣿⡟⠋⠁⣠⣾⣿⣿⣿⣿⣦⡀ ⢀⣿⡇ ⢀⣴⣿⣿⣿⣿⣿⣦⠈⠙⢿⣿⣿ ");puts("");
    printf("\033[44m%s\033[0m","\t\t\t\t ⣿⣧ ⣠⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣦⡀ ⣸⣿ ");puts("");
    printf("\033[44m%s\033[0m","\t\t\t\t ⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿ ");puts("");
    printf("\033[44m%s\033[0m","\t\t\t\t     TERRITORIO CONQUISTATO     ");puts("");
    puts("\n");
}

void printAttackResults(char *terr1, char *terr2, int carriPersiAtt, int carrPersiDif){
    printf("\033[1;31m");
    puts("\n");
    printf("\t\t $$\\    $$\\  $$$$$$\\      \033[0m\033[41m'%s' HA ATTACCATO
'%s'\033[0m\033[1;31m",terr1,terr2);puts("");
    printf("\t\t $$ |   $$ |$$  __$$\\                                  ");puts("");
    printf("\t\t $$ |   $$ |$$ /  \\__|     RISULTATO ATTACCO:         ");puts("");
    printf("\t\t \\$$\\  $$  |\\$$$$$$\\                                  ");puts("");
    if(carriPersiAtt == 1){
        printf("\t\t  \\$$\\$$  /  \\____$$\\      ⊙ %d carro armato perso in %s
",carriPersiAtt,terr1);puts("");
    }else{
        printf("\t\t  \\$$\\$$  /  \\____$$\\      ⊙ %d carri armati persi in %s
",carriPersiAtt,terr1);puts("");
    }
    printf("\t\t   \\$$$  /  $$\\   $$ |                                 ");puts("");
    if(carrPersiDif == 1){
        printf("\t\t    \\$  /   \\$$$$$$  |     ⊙ %d carro armato perso in %s
",carrPersiDif,terr2);puts("");
    }else{
        printf("\t\t    \\$  /   \\$$$$$$  |     ⊙ %d carri armati persi in %s
",carrPersiDif,terr2);puts("");
    }
    printf("\t\t     \\_/     \\______/                                  ");puts("");
    puts("");
    printf("\033[0m");
}
```

viewUtils.h

```c
#include <stdbool.h>
#include <stdio.h>
#include <string.h>

void clearScreen() ;

void printBoldRed(char *str);

void printError(char *errorMessage);

void printSuccess (char *successMessage);

void showAppTitle() ;

void printWinningTrophy();

void printConqueredTerritory();

void printAttackResults(char *terr1, char *terr2, int carriPersiAtt, int carrPersiDif);
```