GAMECONTROLLER

Il GameController è la classe responsabile dell'aggiornamento dello stato del Modello. Ogni chiamata a una funzione prevede un controllo per verificare se l'operazione è eseguibile; in caso affermativo, il controller modifica lo stato del Modello. Se il controllo fallisce, viene sollevata un'eccezione di tipo ControllerException, caratterizzata da un indice e un messaggio specifico.

Per garantire che i giocatori possano effettuare solo le azioni consentite dal proprio stato, è stato adottato il design pattern State. Ogni giocatore possiede un attributo actual\_state che rappresenta il suo stato corrente, e ogni chiamata a un metodo deve essere effettuata nel contesto player1.actual\_state.metodo(). Se un metodo non è consentito nello stato attuale, viene sollevata un'eccezione.

Il costruttore GameController accetta il numero massimo di giocatori (max\_num\_player) che definiscono la partita. Il gioco non inizia finché non sono presenti un numero di giocatori pari a max\_num\_player. Inizialmente, il gioco si trova nello stato NOT\_INITIALIZED e rimane in attesa finché non si raggiunge il numero massimo di giocatori.

L'aggiunta dei giocatori avviene tramite il metodo createPlayer, il quale richiede un nome (String) e un flag isFirst (boolean) per indicare se il giocatore è il primo. Dopo l'aggiunta di ogni giocatore, viene invocato il metodo checkNumPlayer per verificare se il numero massimo di giocatori è stato raggiunto e, in caso affermativo, avviare il gioco.  
  
Una volta raggiunto il numero massimo di giocatori:

1 il model passa dallo stato NOT\_INITIALIZED allo stato BEGIN

game.gameNextState();

2 Tutti i player passano dallo stato NOT\_INITIALIZED allo stato BEGIN

notifyObserver()

3 Vengono creati i mazzi da gioco e distribuite le carte con il metodo

game.actual\_state.initializedGame();

4 Sia i giocatori che la partita passano dallo stato BEGIN allo stato CHOOSE\_GOAL  
notifyObservers();

game.gameNextState();

Ogni player chiama in maniera autonoma il metod playerChooseGoal. Non è presente un turno di chiamata per questo metodo.

Ogni giocatore può selezionare una sola volta la propria carta obiettivo attraverso il metodo playerChooseGoal. Un contatore tiene traccia del numero di giocatori che hanno selezionato la carta obiettivo. Quando il numero raggiunge max\_num\_player, sia il gioco che tutti i giocatori cambiano stato da CHOOSE\_GOAL a CHOOSE\_STARTING\_CARD. Questo stato funziona in modo del tutto analogo al precedente. Per garantire che ogni giocatore possa selezionare la propria carta obiettivo una sola volta, è stata utilizzata una mappa choosed\_goal, che tiene traccia se ogni giocatore ha già selezionato la propria carta obiettivo.

Per garantire una distinzione tra i passaggi iniziali di stato dei giocatori, dove tutti passano contemporaneamente dallo stato NOT\_INITIALIZED allo stato BEGIN, e quelli relativi al loro effettivo turno di gioco, sono stati implementati due diversi metodi:

* notifyObservers(): Questo metodo notifica a tutti gli osservatori (i giocatori) che è il momento di cambiare stato. Viene utilizzato durante i passaggi iniziali di stato, quando tutti i giocatori devono passare allo stato successivo contemporaneamente.
* nextStatePlayer(): Questo metodo viene utilizzato nella gestione dei turni di gioco. Se un giocatore si trova nello stato PlaceCard, il suo stato verrà cambiato in DrawCard. Se lo stato è già DrawCard, il giocatore passerà allo stato WaitTurn, mentre il prossimo giocatore passerà allo stato PlaceCard. In particolare, se un giocatore raggiunge 21 punti, sarà necessario considerare attentamente il cambiamento di stato.

Il metodo placeCard() consente ai giocatori di posizionare le proprie carte sul campo di gioco. Viene effettuato un controllo sulla posizione della carta attraverso la funzione checkPlacing(), e se il posizionamento è valido, la carta viene effettivamente posizionata sul campo e lo stato del giocatore passa da PlaceCard a DrawCard.

I giocatori possono pescare carte dai vari mazzi attraverso i metodi playerPeachCardFromGoldDeck, playerPeachCardFromResourcesDeck e playerPeachFromCardsInCenter. È prevista una gestione particolare nel caso in cui uno o più mazzi siano esauriti durante il gioco.

* Il Controller implementa anche un meccanismo per la gestione degli eventi finali del gioco, come il conteggio dei punti e la determinazione della classifica dei giocatori.  
    
    
  Conteggio dei punti:

Il conteggio dei punti avviene nel metodo finalPointEndGame() del GameController. Per ogni giocatore, vengono eseguite le seguenti operazioni:

Si imposta lo stato del giocatore in "stato finale", in cui non è consentito effettuare ulteriori azioni.

Si calcolano i punti ottenuti dagli obiettivi completati dal giocatore e si aggiungono al suo punteggio totale.

Si calcolano i punti ottenuti dalle carte presenti sul campo del giocatore, aggiungendoli al suo punteggio totale.

Si ordina la lista dei giocatori in base al punteggio totale ottenuto e al numero di obiettivi raggiunti. Questo ordine determinerà la classifica finale dei giocatori.

* Caso dei mazzi esauriti durante il gioco:

Durante il gioco, potrebbe verificarsi la situazione in cui uno o entrambi i mazzi di carte (ad esempio, il mazzo delle risorse e il mazzo dell'oro) si esauriscono. In questo caso, vengono adottate le seguenti misure:

Pesca di carte dai mazzi esauriti: Se un giocatore cerca di pescare una carta da un mazzo esaurito, viene sollevata un'eccezione e il giocatore non può effettuare l'azione desiderata.

Fine del gioco: Se entrambi i mazzi si esauriscono contemporaneamente, il gioco entra nella sua fase finale. In questa fase, i giocatori completano le loro azioni finali e vengono conteggiati i punti. Se uno dei mazzi si esaurisce prima dell'altro, il gioco continua normalmente fino a quando non vengono esauriti entrambi i mazzi. Una volta che entrambi i mazzi sono esauriti, il gioco passa alla fase finale come descritto sopra.

CONNESSIONE E COMUNICAZIONE

L'architettura di comunicazione e connessione del nostro progetto è sviluppata nel modo seguente:

Un Server globale che una volta eseguito crea entrambi i server RMI e SOCKET al quale un Client generico può accedervi, il client in particolare tramite una scelta iniziale manifesta la sua preferenza e viene creata un'istanza di Client che può essere appunto o RMIClient o SOCKETClient, importante notare che il server a cui ci si connette inizialmente è un server Lobby, indipendente dalla gestione delle partite.

Finita la connessione la comunicazione avverrà nel modo seguente:

Un client, al quale non è ancora stato associato un player, sceglie il proprio nome (\*il come verrà spiegato meglio nella sezione riguardante la disconnessione/riconnessione) ed esprime la sua preferenza riguardo l’entrare in una partita esistente( se presenti e libere ) oppure crearne una nuova.  
A questo punto il server lobby, se si viene a creare una nuova partita, creerà un nuovo Game Server che si occuperà di comunicare direttamente con il controller di un game, da questo punto in poi il Client comunicherà quasi unicamente con il Game Server (\*unica eccezione saranno le funzioni per il controllo disconnessione ), similmente se si decide invece di entrare in un game già esistente, il server lobby ci restituirà le informazioni del server legato alla partita scelta e il client si connetterà al suo Game Server.

A questo punto introduciamo la desincronizzazione dei client: per permettere la comunicazione tra client con server lobby differenti, abbiamo fatto in modo che il Game Server desincronizzasse il flusso di thread generato dai possibili client rmi.  
Questa di basa sull’introduzione di una coda che riceve dei pacchetti ( wrap ) dai client e che verranno “spacchettati” ( unwrapped ) dal game server per capire il tipo di funzione da chiamare nel controller, quali dati inviare e altre informazioni utili, rendendo l’invio di informazioni indipendente dal tipo di Client e permettendo a tutti i client di essere sincronizzati allo stesso livello rompendo il ciclo di thread creato dai client rmi.

\*Gestione delle disconnessioni:  
A questo punto descriviamo l’identificazione di una disconnessione e la possibile riconnessione di un client. Tramite dei metodi chiamati HeartBeat che inviano dei segnali periodici tra client e server, quest’ultimo in caso di disconnessione del client non riceverà in tempo il suo heartbeat e lo considererà disconnesso, da questo punto in poi se il player associato al client si trovava in uno stato iniziale di selezione carte o obiettivi, questi vengono selezionati automaticamente dal game controller ( per permettere ai giocatori in partita di cominciarla senza aspettare l’arrivo di un nuovo giocatore ) e nel caso in cui sia il suo turno per piazzare una carta o pescare, questo viene saltato.   
Nel caso di riconnessione invece un client inserirà come prima cosa il nome, quest’azione può avere tre diversi esiti:   
Il nome non è presente in nessuna delle partite 🡪 il client si connette normalmente come nuovo player  
il nome è presente in una partita, ma il player associato risulta ancora connesso 🡪 ricevo un errore di inserimento e devo reinserire un nome univoco  
il nome è presente in una partita ed il player associato è disconnesso 🡪 il client viene riconnesso alla partita e al game server riprendendo da uno stato di attesa turno