

MODEL

Cards:

- Buildings: Tutti i Building ed enum di EffectType)
- TribeCards: Tutti i personaggi ed eventi + enum dei tipi + classi astratte)

GamePackage

- Board: Contiene ed inizializza tutti i mazzi e liste, esponendo metodi per pescare ed accadervi.
Gestisce mazzi e liste durante cambi di turno/era
- CurrentState: Contiene le informazioni dello stato attuale come player, era, ...
- GameStages: enum degli stadi di gioco (waiting, token-choiche, slot-choiche, res-actions, opt-card-e, res-event, ending, ended)

Orders: gestisce gli effetti, di order logic card in base al numero di player (bonus e malus).

Contiene OrderPlayer

DecksCreator: utility class che carica le carte dal file json e le crea

OrderLogicCard: classe astratta che fornisce metodi per fare push/pop/interogare con la order card

PlayableCard: classe astratta che definisce gli attributi base per ogni carta

Slot: definisce ogni singolo slot della Board e gli effetti associati (numero di carte da pescare, cibo, ...)

Player: definisce l'entità del giocatore con token associato, username, cibo, prestigio e carte

Game: classe model del paradigma MVC. Contiene tutte le informazioni del gioco (players, order, slot, liste) e le azioni per interagirvi (creare le sottoclassi, pescare, selezionare token, disconnettersi/riconnettersi, finire il gioco).

Contiene anche la logica interna per avanzare nel gioco come nextPlayerSetup (scambia le fasi di un turno), nextRound, endGame.

MiniModel: copia senza logica di Game nei client. Formata da liste di informazioni (liste di carte, player disconnessi, slotMap, order card, stato corrente...) e metodi per aggiornarle dopo che il server invia l'azione effettuata.

CONTROLLER

GameController: controller del paradigma MVC incaricato di mediare tra lo stack di rete del server ed il model.

Esegue controlli sulle azioni richieste e poi le inoltra a Game. Inoltra poi la sua risposta alla rete che decide se inviare l'azione (true) o l'errore (false)

ClientController: controller lato client che media tra MiniModel, View e stack di rete del client.

Ad esempio: (View) chiede di perorare una carta →
(ClientController) inoltra la richiesta alla rete →
(Network) inoltra la richiesta al server →
(Network) ottiene la risposta dal server →
(ClientController) applica l'azione sul MiniModel e aggiorna la View

GUI (VIEW)

GuiScene: contiene i file .FXML che rappresentano lo scheletro delle 4 scene (Login, token choice, main e leaderboards)

FireParticleSystem: classe ausiliaria che prende una scena e genera degli effetti, particellari su di essa

GUI: crea le quattro scene ed i relativi controller. Ha il metodo render che, in base allo stato del gioco, mostra la scena corretta (applicando un effetto di transizione).

Oltre a render, espone anche showError per ritornare alla schermata di login (se il server crasha) oppure mostrarlo nel gioco.

LoginFXMLController: controller FXML della prima scena di login. Offre dei form per inserire i dati e un pulsante per connettersi (alla pressione crea il client RMI/TCP e lo imposta nel ClientController).

TokenFXMLController: controller FXML della seconda scena per la scelta dei token.

Mostra i token da scegliere e, quando premo confirm, chiede al ClientController di scegliere il token selezionato.

MainFXMLController: controller FXML della scena principale del gioco.

Ha quattro HBox per rappresentare le due file di carte, la board e la mia mano.

Ha una VBox per i players, il round e i pulsanti per le informazioni, lo skip e il quit.

Ogni azione viene inoltrata al ClientController.

LeaderboardFXMLController: controller FXML dell'ultima scena contenente la classifica finale.

Ha un pulsante per tornare al login e iniziare una nuova partita.

ClientLauncherGUI: eseguibile che crea View e ClientController unendoli e fa partire l'interfaccia grafica.

TUI (VIEW)

AsciiTable: classe helper che offre metodi per creare tabelle in ascii art date delle liste di informazioni in ingresso

TUI: classe centrale che prende le informazioni dal MiniModel, le organizza in blocchi e tabelle e le stampa.
Ci sono metodi helper (ad esempio per centrare il tabellone) ed altre per costruire modularmente i blocchi come BoardStamp e BuildAllHandsContent.
Infine ci sono 3 metodi per unire tutti i blocchi (BuildTotalContent, BuildStandingContent, BuildBoardContent), uno per stampare il contenuto finale (display) e uno per stampare una linea informativa sui comandi disponibili in quel momento (printLine)

ClientLauncherTUI: classe eseguibile che accetta i comandi in input. Ci sono 3 tipi di input:

- Comandi per il login: dopo aver inserito le informazioni, viene creato il client RMI/TCP e passato al ClientController
- Comandi per le azioni: le azioni vengono inoltrate al ClientController
- Comandi grafici: usati per ridisegnare la pagina o mostrare informazioni

SERVER

ServerLauncher: classe eseguibile principale del server.

Crea le strutture necessarie come playlist, actionQueue e gameController, successivamente ripristina il game precedente (se esiste) tramite SaveManager, avvia il game processor ed i due server network TCP/PMI.

LimitedMap: mappa contenente i player e il booleano per connesso/disconnesso.

Quando raggiunge il numero desiderato di giocatori, invia il mini model ai client tramite Composite Client Broadcaster.

SaveManager: classe helper per salvare/caricare/eliminare un salvataggio del server.

GameEventProcessor: classe incaricata di eseguire gli eventi nella actionQueue.

Gestisce il caso del game attivo (finito), sospeso (un solo giocatore) e attivo:

applica l'evento sul game controller, se da errore invia l'evento al mittente con flag d'errore, altrimenti lo annida con le liste rilevanti da cambiare e lo invia a tutti i client con Composite Client Broadcaster.

RMI (NETWORK)

RMI Server: accetta le varie mosse inserendole in addRowQueue e gestisce il join nel game:

- Se il game è nullo oppure lo stato d'attesa si è svuotato, ricrea il model
- Se un player nuovo si connette durante una partita gli ritorna un errore
- Altrimenti, viene aggiunto al gioco (gestito sia il caso di riconnessione after crash del server che after disconnessione client)

RMI Heartbeat: ogni client ha un heartbeat sul server a cui manda un ping e il server si salva il timestamp dell'ultimo ping.

Se ora più di 1 secondo fa, il client viene disconnesso.

RMI Client: ha un metodo per ^(dis)connettersi, per partire l'heartbeat, e vari metodi per ogni tipo d'azione

ClientCallbackImpl: classe per ricevere risposte dal server: ha un metodo per inizializzare il minmodel e uno per applicare l'evento ricevuto sul controller client

CHI HA FATTO COSA

Aleandro:

Model - Buildings, 2 eventi, mini model (can test)

View - Gui (tutta), TUI (refactor)

Network - network event, RMI (no FA), Server launcher

Gabriele:

Model - Buildings, 2 eventi, Player,

Network - TCP, persistenza FA, Server launcher

Mattia:

Controller - test, JavaDoc

Model - characters, test, JavaDoc

Ruben:

Controller: tutto

Model: Game, Game Package, Decks Creator,

Network: resilienza FA, Server launcher

View: TUI (iniziale), GUI (postidmante)