Peer Review UML

Gruppo 17 (Pozzato, Puccia, Romano, Saso)

State Pattern

Il modello UML delle classi che compongono il progetto si basa sullo schema concettuale dello state pattern. Data la regolarità del susseguirsi delle fasi di gioco all'interno di una partita risulta possibile identificare per ogni turno un insieme di stati attraverso cui ogni giocatore deve procedere secondo le regole:

-INITIAL STATE: questo stato viene raggiunto nel momento in cui il primo player entra nella partita specificando il numero di giocatori di cui la partita è composta. Vengono creati e mescolati i mazzi, vengono distribuite tante mani quanti sono i giocatori indicati e vengono assegnati gli obiettivi comuni.

-WAIT PLAYER STATE: si attende che il numero di giocatori presenti nella lobby corrisponda al numero di giocatori indicati dal primo player. Ogni volta che un nuovo giocatore entra nella lobby viene assegnato a una mano e a un turno di gioco.

-PLACED CARD STATE: il giocatore aggiunge una carta alla struttura.

-DRAWN CARD STATE: il giocatore pesca una carta da uno dei deck o scegliendola tra le carte scoperte. Vengono ricalcolati i punti di ciascun giocatore.

(La macchina a stati finiti una volta stabilita la partita continua in loop tra gli stati PLACED CARD e DRAWN CARD. Un turno infatti è costituito dalle sole azioni di piazzamento e pesca).

-END STATE: questo stato viene raggiunto nel momento in cui uno dei giocatori supera i 20 punti. Permette ai giocatori di fare il giro finale e conta i punti per decretare il vincitore.

Relazione Model-Controller

Il progetto è stato pensato per rispondere al paradigma "Fat Model".
Lo scopo del controller è unicamente quello di controllare se l'azione dell'utente è legale o meno secondo le regole del gioco. Sarà poi compito suo comunicare la volontà dell'utente al model il quale, a seconda del comando ricevuto e dello stato in cui si trova, modifica di conseguenza lo stato del gioco inteso come il valore degli attributi delle classi elencate nella sezione "Rappresentazione dello stato".

Se tramite la macchina a stati vengono chiamati sequenzialmente una serie di metodi deputati alla modifica dello stato essa è realmente realizzabile tramite la classe Game che rappresenta l'anello di congiunzione tra gli stati della FSM e lo stato del gioco inteso come contenuto delle classi.
Game contiene la logica per cambiare opportunamente il valore degli attribuiti nelle classi che a Game associate (e.g. Deck, Board, Structure ecc).

Game

La classe Game viene stanziata una volta per partita. Si occupa della gestione dei dati relativa alla partita e gestisce anche una parte rilevante della logica di gioco. Questa scelta implementativa sacrifica in parte il disaccoppiamento e la divisone netta Model-Controller per consentire maggiore agilità nell'interrogazione e negli aggiornamenti dello stato di gioco. Il controller si occuperà quindi di controllare e comunicare gli ingressi, in parte direttamente sul client per permettere al programma un maggior grado di responisveness, ma senza avere accesso diretto alle modifiche del gioco.

Più in dettaglio, tra gli attributi della classe è univocamente identificato l'id del game che rappresenta e ritroviamo anche i riferimenti alle varie componenti sotto elencate (Board, Deck, ecc) al fine di centralizzarne per suo tramite l'accesso e la modifica.

Relazione Model-View

Il pattern utilizzato nel progettare le comunicazioni tra Model e View è l'Observer-Observable.

Viene pertanto data la possibilità a classi specifiche della View di registrarsi come ascoltatori di eventi nell'attributo ObserverList della classe Game.

Una volta inseriti nella lista essi si metteranno in ascolto di eventi lanciati dalla corrispettiva classe sul model (ad esempio inscrivendosi alla ObserverList la classe Deck della View viene avvisata nel momento in cui la classe Deck del model viene modificata).

Abbiamo pertanto elencato enumerandole tutte le modifiche dello stato del gioco divise per componenti il cui verificarsi rende necessario informare la View affinché possa cambiare di conseguenza. Esse sono:

- GAME HAND UPDATE EVENT
- GAME STRUCTURE UPDATE EVENT
- GAME BOARD UPDATE EVENT
- GAME DECK UPDATE EVENT
- NEW PLAYER EVENT
- NEW GAME EVENT

Rappresentazione dello stato

Board

La classe Board simula il piano di gioco, tenendo conto delle carte già uscite, dell'obiettivo comune e aggiornando il punteggio dei vari player.

Deck

La classe Deck gestisce sia il resourceDeck che il goldDeck, implementati come stack. Vengono creati nella fase iniziale del gioco, mischiati. Da essi si può solamente pescare fino ad esaurimento deck.

Player

Il player è identificato generalmente dal nickname e all'interno della partita dal colore. Nella creazione del player la scelta progettuale adottata e che traspare dall'UML è l'utilizzo dell'abstract factory method per gestire in maniera fluida e la creazione di un tipo specifico di player (la cui specificità è costituita dal tipo di connessione utilizzata e dall'interfaccia grafica o testuale di cui desidera fruire).

Structure

La classe Structure gestisce, per ogni player, la struttura delle carte posizionate. È attualmente sviluppata come un albero di StructureNode, classe che tiene conto dei vari fathers and child della carta con un riferimento alle carte connesse.

La classe restituisce tutte le informazioni relative alla struttura. Dunque risorse e oggetti attualmente disponibili, riconoscimento pattern, possibili piazzamenti delle carte, calcolo dei punteggi relativi al singolo giocatore in funzione delle carte obiettivo comuni e segrete.

Card

Card è una classe astratta che tiene conto dell'id univoco della carta e delle immagini relative alla carta. Distinguiamo quindi le quattro tipologie di carta ObjectiveCard, InitialCard, GoldCard, ResourceCard. Ognuno gestisce le informazioni da mantenere e le azioni specifica del tipo della carta.

A titolo di esempio, la sottoclasse ObjectiveCard si limiterà a salvare, settare e restituire il tipo di obiettivo (contenuto nell'enum ObjectiveCardType), mentre la costruzione e l'interrogazione di una GoldCard terrà in considerazione il tipo di gold card, le risorse necessarie, il punteggio ottenuto…

Hand

La classe hand è relativa ad un singolo player. Nella fase iniziale si occupa di tenere le alternative di obiettivo segreto e di salvare l'obiettivo definitivo una volta scelto. Nel resto del gioco tiene conto delle tre carte in mano al giocatore, rimuovendo la carta da piazzare e inserendo la carta pescata nella lista ad ogni turno.

Enum

Abbiamo fatto abbondante uso delle enum così da mantenere un quadro chiaro di tutte e sole le casistiche da affrontare nel corso del gioco. In ordine sparso troviamo le enum Objects, GoldCardType, ObjectiveCardType e tutte le enum relative agli eventi, necessarie a gestire la comunicazione Model-View ancora in fase di sviluppo.

Chat

La chat è attualmente implementata come una lista di ChatMessage di cui viene salvato id del messagigo, sender, lista di reciever (per definire a chi è visibile il messaggio), messaggio da mandare e timestamp .