Peer-Review 1: UML

Niccolò Brembilla,
Luigi Bruzzese,
Eleonora Cabai,
Milagros A. Casaperalta Garcia

Gruppo AM31
28 marzo 2023

Spiegazione del diagramma UML
delle classi del gruppo AM31

A.A. 2022/2023
Prof. Alessandro Margara

1 Server

All'interno del server, per facilitare la visualizzazione del diagramma UML, abbiamo definito i packages *model*, *controller* e

- *gameExceptions*, che contiene alcune particolari eccezioni
- *ListenerPattern*, che è una libreria Java che supporta lo scambio di messaggi tra il model ed il controller

1.1 Package Model

All'interno del model sono contenute tutte le classi utilizzate come base per l'implementazione del gioco, in particolare

- nel package *comGoals* è presente la classe astratta *ComGoal*. Questa viene estesa da 8 classi che rappresentano i 12 obiettivi comuni del gioco, che in alcuni casi sono stati raggruppati grazie a dei pattern di controllo comuni
- la classe *PersGoal* contiene 12 attributi *enum* che rappresentano tutti i possibili obiettivi personali. La configurazione di ciascuno di essi, cioè la posizione in cui si deve trovare uno specifico *HouseItem*, è contenuta in un file .json che viene letto in fase di costruzione
- la classe *Board* è stata implementata come una matrice di *ItemCards* supportata da una matrice di interi che contiene in ogni cella il numero minimo di giocatori supportati da essa (0, 2, 3 oppure 4)

In particolare, i check sulla *Board* e sulla *Bookshelf* vengono effettuati tramite i rispettivi metodi privati di selezione delle caselle (*checkSelection()*) e di inserimento (*checkSpace()*), invocati dai metodi *selectCard()* ed *insertCard()*, che, in caso di fallimento, lanciano un'eccezione da inviare al controller. I metodi *getAsArrayList()* di entrambe le classi restituiscono una loro rappresentazione utilizzata per creare i messaggi da inviare al client, in modo da non esporre il rep.

1.2 Package Controller

All'interno del controller abbiamo inserito

- il package *Messages* che contiene tutti i possibili messaggi che possono essere scambiati tra il model ed il controller. Attualmente questo package permette solamente di identificare le tipologie di informazioni scambiate tra client e server, infatti il modo in cui avverrà lo scambio (serialmente oppure tramite stringhe) è ancora in fase di decisione

- la classe astratta *ConnectionControl*, della quale viene creata un'istanza per ogni utente connesso in base alla tecnologia utilizzata (socket o RMI). Questa classe contiene il metodo *run()*, che la avvia come thread e le permette di rimanere in ascolto di ciò che richiede il client (o delle eventuali invocazioni RMI), e i metodi utili per inviargli le modifiche effettuate sul model
- la classe *GameController*, che ascolta gli aggiornamenti del model mediante il pattern Listener (metodo *propertyChange()*) e attua le modifiche su esso usando i metodi *insertCard()* e *selectCard()*, invocati dal *ConnectionControl*. Questi per prima cosa controllano se è il turno del client invocante: in caso negativo ritornano un messaggio di errore, altrimenti invocano i rispettivi metodi sul *GameModel*. Tra i vari attributi, si notino *currPlayer* e *turnPhase*, che tengono traccia del giocatore corrente e della fase del gioco per gestire l'interazione con i client

2 Client

All'interno del package *Controller*, è presente la classe *ClientController*, che conserva le copie della *Board* e della *Bookshelf* del giocatore. Questa classe viene invocata dal *ClientConnection* (che dialoga con il server, sempre in base alla tecnologia del client) e, a sua volta, invoca la view chiedendole eventuali azioni e ponendosi in ascolto su di essa (Pattern Listener). A seguito delle azioni eseguite dal client sulla view, la classe effettua su esse dei primi check, che evitano l'invio di dati errati al server.

La classe *ClientConnection*, in modo simile alla classe *ConnectionControl* del server, viene estesa in base al tipo di connessione del singolo client.

3 Classe Position

La classe *Position* è una classe astratta che contiene dei metodi statici utili per convertire in coordinate gli interi che indicano una posizione (sulla board o sulla bookshelf). Viene utilizzata per effettuare i controlli sulle posizioni delle *ItemCard* sulla *Board*.