Peer-Review 1: UML

Alessandro Amandonico, Francesco Buccoliero,
Kaixi Matteo Chen, Lorenzo Cavallero
Gruppo 10

30 marzo 2023

Valutazione del diagramma UML delle classi del gruppo `IS23AM19`.

1 Lati positivi

Ad alto livello, il design proposto dal gruppo 19, astrae correttamente il concetto di `Model View Controller` identificando le opportune classi quindi oggetti, necessari per la implementazione del software richiesto. La struttura, permette la corretta centralizzazione del modello, quindi stato del `game` su un singolo server, con client distribuiti.

La soluzione adottata di caricamento dei `Private Goals` da file di configurazione `JSON` è stata ritenuta solida, scalabile ed estendibile. La scalabilità ed estendibilità dei controllori (quindi metodi) di tali regole non risulta chiaro dal diagramma `UML`.

Ottima la integrazione di una `Lobby` a livello client, la quale gestisce tutte le possibili forme di richiesta di partecipazione ad un game (`New Game`, `Join Game`) e riconnessione di un client in caso di disconnessione.

2 Lati negativi

La creazione di molteplici classi per la modellizzazione dei `Common Goals` riscontra un basso processo di ingegnerizzazione nell'astrarre il concetto di `Common Goal` all'interno del modello di gioco risultando molto verboso e poco estendibile. Si ritiene che l'utilizzo di un pattern `Strategy + Composition`,

sarebbe stato più adatto, ottenendo quindi una singola classe `Common Goal`. Inoltre, non si ritiene che la `ownership` dell'attributo `alreadyScored` sia di competenza dell'oggetto `Common Goal`, il quale dovrebbe indicare solamente la natura dell'obiettivo.

Il modo in cui è stata duplicata la classe `Player` tra server (`Player`) e client (`Playing Player`) risulta poco ottimale, si ritiene che gli attributi in comune debbano essere spostati in una classe base quindi estendere su necessità.

Per evitare una `Inheritance Chain` troppo lunga[1], si ritiene che la corretta astrazione sia tra `Player` e `Playing Player` e non tra questo ultimo e `Lobby Player`.

Data la presenza di costruttori non banali, almeno per `Game Master` e `Player` si ritiene che sarebbe stato opportuno inserire un `Creational Pattern` per gestire diversi casi di creazione condizionata di oggetti interni [2].

Si nota la scelta di non implementare la classe `Common Goal Card` (o `Tupla`) la quale, oltre al `Common Goal` potrebbe contenere la descrizione di tale obiettivo in modo da essere mandata al componente `View`. Ci risulta sconosciuto come tale descrizione venga gestita a livello client dal diagramma `UML` ma data la assenza di una configurazione comune per i `Common Goals` ipotizziamo che questo legame venga spezzato [3].

Infine, non risulta ben chairo se il gruppo abbia intenzione di sviluppare la feature aggiuntiva del multipartita, in caso affermativo, non sono stati riscontrati gli attributi adeguati in merito all'interno della classe `Controller`.

3 Confronto tra le architetture

Non sono stati riscontrati scelte architetturali o design patterns con maggiore forza rispetto al nostro progetto.

[1] `AbstractPlayer,Player, ClientPlayer o ServerPlayer`

[2] Alcuni errori possibili sono: `DuplicatePlayerName, AlreadyUsedCommonGoal, AlreadyUsedPrivateGoal`

[3] Presupponendo che a livello client venga mostrata la descrizione dell'obiettivo, esso dovrebbe essere accoppiata alla sua implementazione logica