Eriantys - Prova Finale di Ingegneria del Software

Paolo Pertino [10729600] paolo.pertino@mail.polimi.it
Leonardo Pesce [10659489] leonardo.pesce@mail.polimi.it
Alberto Paddeu [10729194] alberto.paddeu@mail.polimi.it

28 marzo 2022

Indice

1 Introduzione

La **Prova Finale di Ingegneria del Software** dell'anno scolastico 2021-2022 prevede lo sviluppo di una versione software del gioco da tavolo *Eriantys*, un prodotto *Cranio Creations*[1] che si ispira e tenta di rinnovare il già affermato *Carolus Magnus*[2].

Il prodotto finale dovrà soddisfare i requisiti *Game-Specific* e *Game-Agnostic* indicati nel documento *requirements.pdf*. In particolare è richiesto l'utilizzo del design pattern *Model-View-Controller* di cui a breve forniremo una concisa descrizione.
Per incrementare il punteggio ottenuto, il team si concentrerà nell'implementazione delle regole complete del gioco, nel fornire la possibilità ai giocatori di connettersi al server e giocare tramite un'interfaccia a linea di comando (CLI) oppure mediante l'interazione con un'interfaccia grafica (GUI). Infine si darà spazio all'implementazione di quante più possibili delle seguente funzionalità aggiuntive: *12 carte personaggio, partite a 4 giocatori, partite multiple e persistenza.*

1.1 Model-View-Controller

Il *Model-View-Controller* è un design pattern per la progettazione di un'architettura software. Esso permette di separare la logica di presentazione dell'applicativo da quella applicativa(o detta di business).

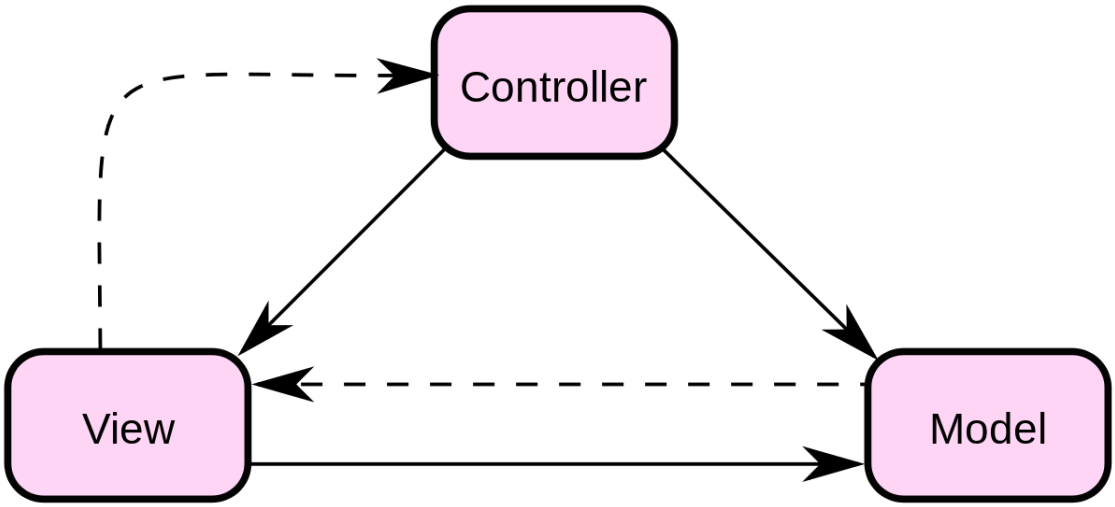

Figura 1: Model-View-Controller Pattern

Come deducibile dal nome, tale architettura è strutturata su 3 differenti layer:

1. *Model* - gestione diretta dei dati, della logica e delle regole del programma. Si noti come l'utente non modifica in modo diretto lo stato del model, bensì si interfaccia con il controller il quale gestisce in separata sede l'interazione con lo stato interno del sistema;
2. *View* - permette la visualizzazione dello stato del model e gestisce l'interazione con gli utenti e agenti esterni;
3. *Controller* - riceve i comandi dell'utente attraverso la view e li attua modificando gli stati degli altri due layers.

Sono possibili viste multiple di uno stesso modello. Nel nostro caso, infatti, saranno implementate due viste: il gioco sarà pertanto accessibile sia attraverso linea di comando sia mediante interfaccia grafica.

2 Devlog

Nella seguente sezione riportiamo settimana per settimana i progressi effettuati dal team, evidenziando, ove necessario, eventuali diagrammi e gli snodi del ragionamento.

2.1 Settimana 1: Un primo sguardo al class diagram del Modello

Durante la prima settimana di corso abbiamo analizzato i componenti fisici del gioco e le sue regole, cercando di riprodurre uno schema logico di tali elementi attraverso un *Class Diagram UML*. Esso contiene una prima bozza della struttura del Model:

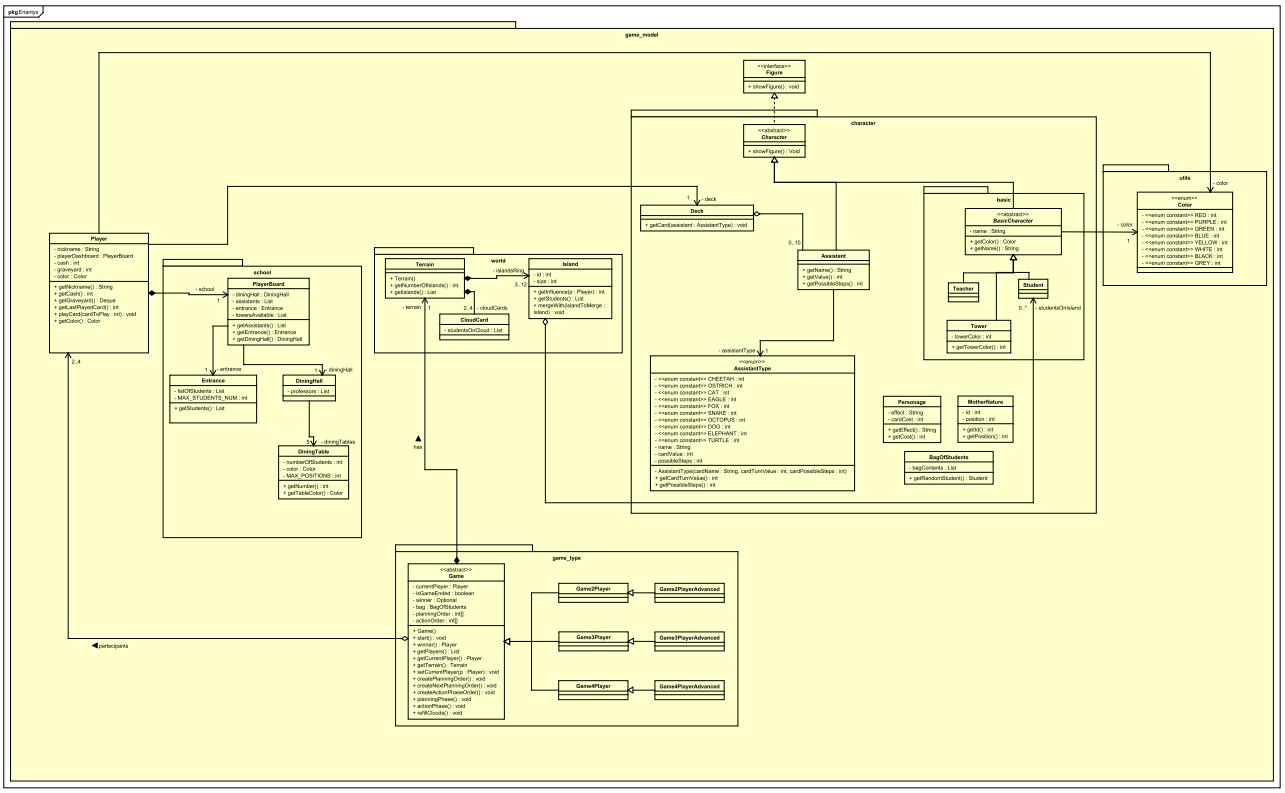

Figura 2: Class Diagram del Model - Bozza

Come indicato, lo schema sopra riportato è una bozza primitiva e di seguito riportiamo i principali ragionamenti effettuati:

- Tutti i componenti fisici, in futuro, avranno una loro grafica che dovrà essere mostrata. Pertanto implementano l'interfaccia GameObject che prevede l'implementazione di un metodo specifico per conseguire tale obiettivo.
- Il cerchio di isole che costituisce la board di gioco è stato pensato come *Doubly Circular Linked List*[3]. Con tale rappresentazione sarà più agevole lo spostamento di madre natura e l'operazione di merge di 2 isole consecutive a valle della loro conquista da parte di un giocatore.
- Volendo implementare le regole complete, quindi prevedendo la possibilità di giocare una partita seguendo la modalità per esperti, e tenendo conto della possibilità di implementare partite multiple sullo stesso server, ci siamo interrogati su come far impattare tale scelta sui diversi metodi delle varie classi. L'idea è quindi quella di utilizzare un Template

Pattern creando una classe astratta di gioco dalla quale saranno derivate le 3 versioni di gioco (per 2, 3 e 4 giocatori). Da esse discenderanno successivamente le corrispettive versioni a 2, 3 e 4 giocatori in modalità per esperti.

- In vista dell'implementazione della funzionalità di persistenza della partita, è stato brevemente analizzato il pattern *Memento* ed il suo funzionamento per capire se esso possa essere utile in futuro.

2.2 Settimana 2: Eriantys Model

Nella seconda settimana è stato scritto parte del codice del model, identificando i punti critici in cui è richiesta un'interazione con l'utente. Inoltre è stata rifinita la struttura del model che riportiamo di seguito aggiornata:

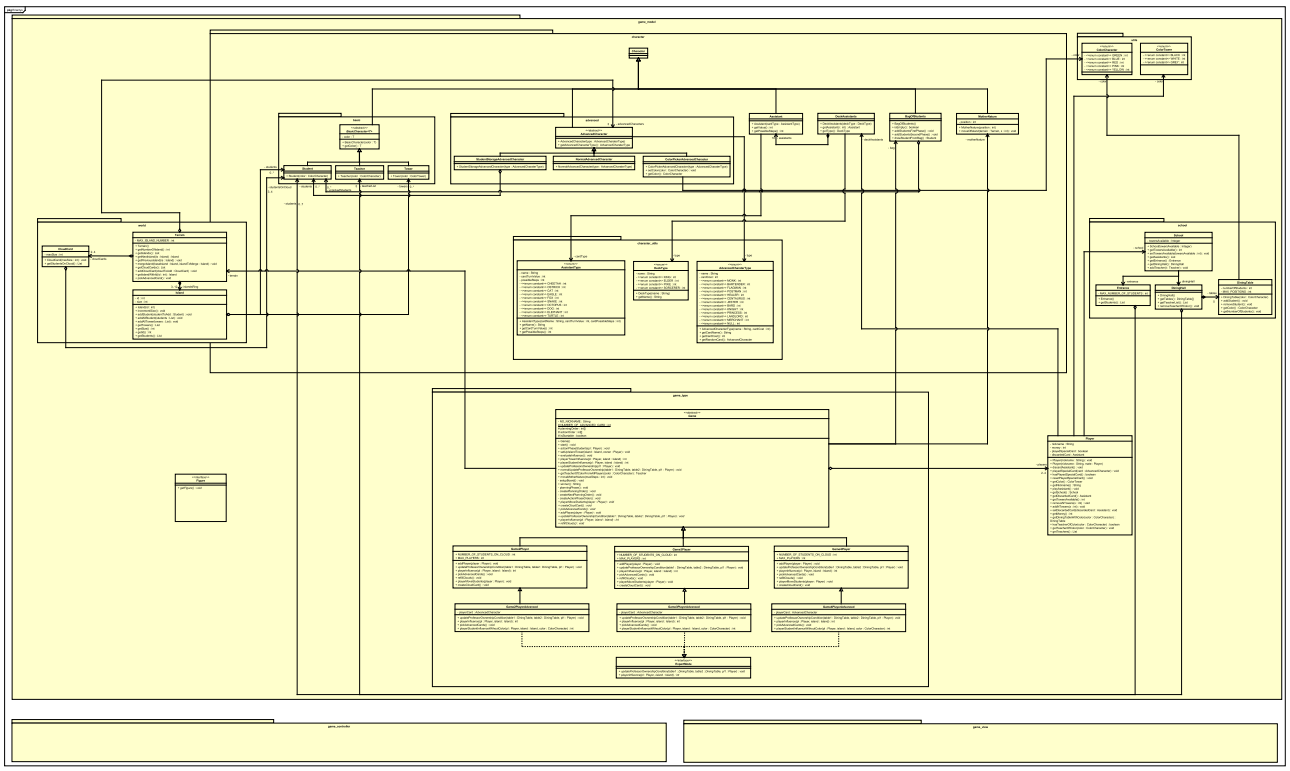

Figura 3: Class Diagram del Model - Seconda settimana

É stata prestata attenzione a:

- individuare un flow di operazioni che contraddistinguono la partita (una volta connessi i giocatori, finchè non è presente un vincitore vengono eseguite Planning Phase, Action Phase rispettivamente per tutti i giocatori continuamente)
- studiare il meccanismo delle carte personaggio per l'implementazione della modalità per esperti più approfonditamente
- restyle del diagramma UML per renderlo coerente con il codice scritto

2.3 Settimana 3: Eriantys Model update e Controller

Nella terza settimana è stato rivisitato e riscritto parte del codice del model e implementato la parte di controller. Tra le varie modifiche le più importanti risultano essere la nuova implementazione della classe Game che adesso risulta essere accorpata nelle varie versioni da 2, 3 e 4

giocatori, la nuova versione per l'implementazione delle carte avanzate e il ribilanciamento del carico tra controller e la classe game.

Figura 4: Class Diagram del Model - Seconda settimana

Seguendo l'ordine sopra elencato delle modifiche:

- **Modifica struttura del game:** Visto che dalle precedenti implementazioni le modifiche effettive nelle istanze di gioco a 2, 3 e 4 giocatori erano poche, è stato ripensato un modo per unirle trasformando la classe game in una generica che varia gli effetti dei suoi metodi in base al numero di giocatori assignati inizialmente. Ciò non ha implicato alcun uso di "if" o "switch" aggiuntivi in quanto con opportuni accorgimenti e con l'uso della funzione modulo "%" si potevano trasformare le funzioni da specifiche a generiche rispetto il numero di giocatori. Questa unione ha comportato anche l'unione delle tre precedenti classi avanzate che a livello pratico implementavano lo stesso codice. Tutto ciò a portato ad avere un codice più snello e pulito senza inutili ripetizioni di codice.
- **Nuove carte avanzate:** Per implementare più carte possibili cercando di non dover scrivere gli effetti delle carte all'interno del codice, ma di tenerli solo all'interno della classe della carta, è stato ripensato come venivano implementate. Queste adesso, in caso di una loro attivazione positiva, vanno a fare override di parametri o classi che poi al momento corretto (in modo da poter giocare la carta in qualsiasi momento) vengono eseguiti.
- **Ribilanciamento tra Model e Controller:** abbiamo notato che molti metodi che appartenevano alla classe controller erano stati implementati inizialmente nel model, questi sono stati spostati e verificati in modo da ribilanciare il peso tra i due componenti.

3 Strumenti utilizzati

Nella seguente sezione verranno indicati i principali strumenti di sviluppo utilizzati:

- *IntelliJ IDEA Ultimate 2021.3.2* - Principale IDE utilizzato.
- *Maven* - Gestione dello sviluppo del progetto software e di tutte le sue fasi.
- *JUnit* - Framework principale di unit testing.
- *AstahUML* - Creazione di diagrammi UML.
- *GitKraken* - Git GUI per visualizzare il workflow di sviluppo ed utilizzare efficientemente Git.
- *TEXStudio* - Gestione e aggiornamento del report.

Riferimenti bibliografici

[1] Eriantys, Cranio Creations

[2] Carolvs Magnvs, Winning Moves

[3] Circular Doubly Linked List