Bozza del 12/10/2025

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI SCIENZA E INGEGNERIA

Corso di Laurea in Informatica per il Management

Lorem ipsum dolor sit amet.

Relatore:
Prof. Luca Padovani

Presentata da:
Alessandro Nanni

Sessione di Dicembre
Anno accademico 2024/2025

Bozza del 12/10/2025

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI SCIENZA E INGEGNERIA

Corso di Laurea in Informatica per il Management

Lorem ipsum dolor sit amet.

Relatore:
Prof. Luca Padovani

Presentata da:
Alessandro Nanni

Sessione di Dicembre
Anno accademico 2024/2025

Bozza del 12/10/2025

Sommario

In questo documento tratterò del mio lavoro svolto sotto la supervisione del prof. Padovani nello sviluppare un sistema software che agevola l'utilizzo della *Domain Specific Language* del videogioco Minecraft.
Verranno inizialmente illustrati i problemi sintattici e strutturali di questo ampio ecosistema di file.
Successivamente mostrerò come ho provato ad ovviarli, o almeno ridurli, tramite una libreria che si occupa di svolgere le operazioni più tediose e ripetitive. Tramite un *working example* esporrò in che modo ho semplificato lo sviluppo di punti critici, facendo confronti con l'approccio abituale.
Infine, mostrerò la differenza in termini di righe di codice e file creati tra i due sistemi, con l'intento di affermare l'efficienza della mia libreria.

Bozza del 12/10/2025

Indice dei contenuti

Introduzione

Se non fosse per il videogioco *Minecraft*, non sarei qui ora. Quello che per me inizialmente era un modo di esprimere la mia creatività piazzando cubi in un mondo tridimensionale, si è rivelato presto essere il luogo dove per anni ho scritto ed eseguito i miei primi frammenti di codice.
Motivato dalla mia abilità nel saper programmare in questo linguaggio non banale, ho perseguito una carriera di studio in informatica.

Il sistema che inizialmente era stato pensato dagli sviluppatori della piattaforma come un modo di «barare» tramite comandi per ottenere oggetti istantaneamente e senza il minimo sforzo, si è col tempo evoluto in un ecosistema di file e codice che permette agli sviluppatori che decidono di usare questa *Domain Specific Language* per modificare moltissimi comportamenti dell'ambiente videoludico.

Minecraft è scritto in Java, ma questa DSL chiamata *mcfunction* è un linguaggio completamente diverso. Non fornisce agli sviluppatori il modo di aggiungere comportamenti nuovi, modificando il codice sorgente. Permette piuttosto di aggiungere *feature* aggiungendo frammenti di codice che vengono eseguiti solo sotto certe condizioni, dando ad un utilizzatore l'illusione che queste facciano parte dei contenuti classici del videogioco. Negli ultimi anni, in seguito ad aggiornamenti, tramite una serie di file JSON sta gradualmente diventando possibile creare esperienze del tutto nuove. Tuttavia questo sistema è ancora limitato, e gran parte della logica è comunque dettata dai file *mcfunction*.

1.1. Cos'è un *pack*

I file JSON e *mcfunction* devono trovarsi in specifiche cartelle per poter essere riconosciuti dal compilatore di *Minecraft* ed essere integrati nel videogioco. La cartella radice che contiene questi file si chiama *datapack*.
Un *datapack* può essere visto come la cartella `java` di un progetto Java: contiene la parte che detta i comportamenti dell'applicazione.

Come i progetti Java hanno la cartella `resources`, anche *Minecraft* dispone di una cartella in cui inserire le risorse. Questa si chiama *resourcepack*, e contiene principalmente font, modelli 3D, *texture*, traduzioni e suoni[1].
Le *resourcepack* sono state concepite prima dei *datapack*, e permettevano ai giocatori sovrascrivere le texture e altri asset del videogioco. Gli sviluppatori di *datapack* hanno poi iniziato ad utilizzarle per definire nuove risorse, inerenti al progetto che stanno sviluppando.

Datapack e *resourcepack* formano il *pack* che, riprendendo il parallelismo precedente, corrisponde all'intero progetto Java. Questa sarà poi la cartella che verrà pubblicata.

1.2. Struttura di *datapack* e *resourcepack*

All'interno di un *pack*, *datapack* e *resourcepack* hanno una struttura molto simile.

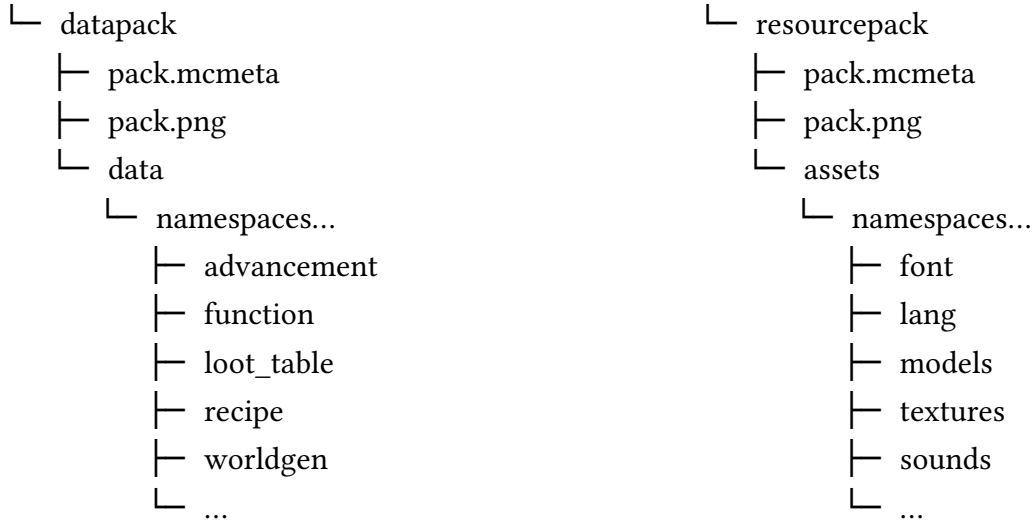

Figura 1: *datapack* e *resourcepack* a confronto.

Anche se l'estensione non lo indica, il file è in realtà scritto in formato JSON e definisce l'intervallo delle versioni (chiamate *format*) supportate dalla cartella, che con ogni aggiorna-

[1] Con l'eccezione di *texture* e suoni, tutti gli altri file sono in formato JSON.

mento di *Minecraft* variano, e non corrispondono all'effettiva *game version*[2].

Ancora più rilevanti sono le cartelle al di sotto di `data` e `assets`, chiamate *namespace*. Se i progetti Java seguono la seguente struttura `com.package.author`, allora i *namespace* possono essere visti come la sezione `package`.
I *namespace* sono fondamentali per evitare che i file omonimi di un *pack* sovrascrivano quelli di un altro. Per questo, in genere i *namespace* o sono abbreviazioni o coincidono con il nome stesso progetto che si sta sviluppando, e si usa lo stesso per *datapack* e *resourcepack*.
Tuttavia, in seguito si mostrerà come operare in namespace diversi non è sufficiente l'assenza di conflitti tra i *pack*, che spesso vengono utilizzati in gruppo.

All'interno dei *namespace* si trovano directory i cui nomi identificano in maniera univoca la natura e la funzione dei contenuti al loro interno: se metto un file JSON che il compilatore riconosce come `loot_table` nella cartella `recipe`, il questo segnalerà un errore e il file non sarà disponibile nella sessione di gioco.

In `function` si trovano file e sottodirectory con testo in formato *mcfunction*. Questi si occupano di far comunicare tutte le parti di un *pack* tra loro tramite una serie di comandi.

1.3. Comandi e Funzioni

Prima di spiegare cosa fanno i comandi, bisogna definire gli elementi basi su cui essi agiscono. In *Minecraft*, si possono creare ed esplorare mondi generati in base a un *seed* casuale. Ogni mondo è composto da *chunk*, colonne dalla base di 16x16 cubi, e altezza di 320.
L'unità più piccola in questa griglia è il blocco, la cui forma coincide con quella di un cubo di lato unitario. Ogni blocco in un mondo è dotato di collisione ed individuabile tramite coordinate dello spazio tridimensionale. Si definiscono entità invece tutti gli oggetti dinamici che si spostano in un mondo: sono dotate di una posizione, rotazione e velocità.

I dati persistenti di blocchi ed entità sono memorizzati in una struttura dati ad albero chiamata *Named Binary Tags* (`NBT`). Il formato «stringificato», `SNBT` è accessibile agli utenti e si presenta come una struttura molto simile a JSON, formata da coppie di chiave e valori[3].

Un comando è un'istruzione testuale che Minecraft interpreta per eseguire una specifica azione, come assegnare oggetti al giocatore, modificare l'ora del giorno o creare entità. Sebbene non disponga delle funzionalità tipiche dei linguaggi di programmazione di alto livello — come cicli for e while, strutture dati complesse o variabili generiche — il sistema dei comandi

[2] Ad esempio, per la versione 1.21.10 del gioco, il `pack_format` dei *datapack* è 88 e quello delle *resourcepack* è 69.

[3] Esempio: `{name1:123,name2:"sometext1",name3:{subname1:456,subname2:"sometext2"}}`.

fornisce comunque strumenti che consentono di riprodurre alcuni di questi comportamenti in forma limitata.

I comandi che più si avvicinano ai concetti tipici della programmazione sono

NOTA: Forse da entrare più in dettaglio

- `scoreboard`: definisce delle variabili di tipo intero globali, sulle quali possono essere semplici operazioni di somma, sottrazione, moltiplicazione e divisione;
- `execute`: permette di eseguire comandi solo sotto certe condizioni;
- `data`: consente di accedere e modificare dati;
- `function`: esegue una funzione.

TODO: Cos è una funzione, contesto di esecuzione, command stack

Come agevolare lo sviluppo

La mia implementazione

Conclusione

Bibliografia