Proposta sulla descrizione logica di generici giochi da tavolo

Università di Modena e Reggio

Alessandro Mezzogori

Domenica 4 Dicembre 2022

Sommario

Capitolo 1

Introduzione

Negli ultimi anni si è presentata una notevole evoluzione dei meccanismi principali sfruttati dai creatori dei giochi da tavolo per creare esperienze immersive, originali e stimolanti per i giocatori coinvolti.
I nuovi generi dei giochi si portano dietro le loro meccaniche caratteristiche, come gli engine building con l'interazione tra componenti e la gestione delle risorse, i giochi di strategia che si concentrano sull'abilità del giocatore di pianificare e adattarsi ai cambiamenti dell'ambiente di gioco, i cooperativi che unisco i giocatori per adempire a un singolo compito comune oppure i giochi di piazzamento lavoratori che obbligano a scegliere con accortezza le poche mosse a disposizione.
Gli avanzamenti nello sviluppo dei giochi da tavolo generano con sé delle difficoltà nel descrivere logicamente e in maniera strutturata i flussi di gioco, limitando la possibilità di effettuare la prototipizzazione direttamente in modo digitale delle meccaniche di possibili nuovi giochi da tavolo, prima ancora di creare una versione fisica.
Nelle seguenti pagine si illustrerà il principale linguaggio usato attualmente seguito da una nuova alternativa per descrivere i giochi da tavolo, con l'obiettivo sia di fornire un'alternativa ai metodi in uso a oggi e sia per stimolare e inspirare la discussione sull'argomento del general game playing.
Importante notare che la proposta non si prospetta come definizione rigida dei requisiti dell'implementazione del linguaggio ma come linea guida comune di partenza e ispirazione per futuri sviluppi, aggiunte e modifiche per creare degli strumenti facilmente utilizzabili e apprendibili velocemente.

Capitolo 2

Game Description Language

Capitolo 3

Tabletop Game Description Language

3.1 Nozioni Fondamentali

3.1.1 Identificatori

Gli Identificatori sono una qualsiasi stringa di caratteri che inizia per un trattino basso "_" o per lettera, opzionalmente seguiti da un qualsiasi ammontare di lettere o numeri senza spazi.

Un identificatore per essere legale deve essere unico all'interno del suo ambito di Visibilità, questo significa che deve essere diverso sia dagli altri identificatori ma anche da operatori e clausole.

3.1.2 Operatore e Clausole

Gli operatori e le clausole sono delle parole chiave o simboli riservati che permettono rispettivamente la costruzione delle espressioni e delle istruzioni.

3.1.3 Blocchi

I Blocchi sono un insieme struttura d'istruzioni, tag e altri blocchi. Un blocco si definisce come tutto il contenuto all'interno di una coppia di parentesi graffe, rispettivamente aperta e chiusa.
Il blocco definire l'ambito, o scope, e vita di una variabile definita al suo interno ovvero il suo campo d'azione che parte dall riga dov'è stata dichiarata fino alla fine del blocco che contiene la sua dichiarazione.
La Visibilità di una variabile si estende a tutti gli ambiti figli del blocco di dichiarazione, dove figlio significa definito all'interno del padre (annidato).

3.1.4 Tags

I Tag sono delle parole chiave riservate che modificano il comportamento del blocco associato, ogni tag descrive cosa è permesso definire al suo interno. All'interno della proposta è possibile che si trovi la dicitura "blocco ¡tag¿" come abbreviazione per indicare la coppia tag e relativo blocco associato.

3.1.5 Commenti

I commenti sono linee di codice che vengono ignorare durante l'esecuzione del programma, per commentare una riga è sufficiente inserire una doppia barra "//", qualsiasi scritta oltre le due barre è considerata parte del commento.

3.2 Primitive del linguaggio

Tabletop Game Description Language si basa su un insieme di primitive da cui sono costruire tutte le funzionalità del linguaggio.

3.2.1 Oggetti

Un oggetto è un'istanza di una certa classe, un blocco di memoria che viene allocato e configurato secondo le specifiche definite dalla classe.
Gli oggetti hanno due comportamenti principali, gli oggetti valori e gli oggetti riferimento:
Oggetti valore o Value objects, si riferiscono direttamente al valore, quando avviene una copia dell'oggetto viene creata una nuova istanza della stessa tipologia di oggetto che contiene lo stesso valore. Il comportamento degli oggetti valore è principalmente per i tipi primitivi come number, string, bool.
Oggetti riferimento o Reference objects si riferiscono al blocco di memoria che gli è stato allocato, segue che quando un reference object è copiato in una nuova variabile, nel nuovo oggetto si copierà il riferimento alla zona di memoria a cui l'oggetto originale si riferisce, evitando di allocare una nuova zona di memoria.
Una conseguenza importante del meccanismo dei riferimenti è che qualsiasi cambiamento effettuato su un oggetto riferimento sarà rispecchiato da tutti gli altri reference objects che possiedono lo stesso riferimento al blocco di memoria

3.2.2 Tipi primitivi

I tipi primitivi sono degli oggetti valore particolari che rappresentano i valori puri del linguaggio come i numeri, le stringhe di testo, i valori booleani.

Numeri

I numeri sono rappresentati dal tipo primitivo "number" che accoglie un qualsiasi numero tale che sia compreso tra $\pm 5.0 * 1-^{-324}$ e $\mp 1.7 * 10^{3}08$ con una precisione tra le 15 e le 17 cifra, il tipo number equivale al tipo double presente in linguaggi come C, C#, ecc...

Stringa

Il tipo stringa, o string, rappresenta una qualsiasi successione di caratteri codificata in UTF-16, non ha una lunghezza massima predefinita. I valori string sono immutabili (ovvero non è modificabile), quando si eseguono delle espressioni che andrebbero a modificare il valore di una stringa, viene instanziata una nuova stringa con il valore modificato.

Booleano

Il tipo booleano, o bool, rappresenta una decisione binaria vera o false. Le variabili di tipo booleano dunque possono assumere due singoli valori rispettivamente rappresentati dalle parole chiave "true" per il valore vero e "false" per il valore falso.

Null

Il valore null non è un tipo primitivo in se ma serve per indicare l'assenza del valore in una variabile, Ogni tipo può assumere il valore null.

3.2.3 Espressioni

Una espressione è una qualsiasi operazione che coinvolge zero o più operatori e uno o più operandi (o espressioni) che può essere valutata all'interno del linguaggio.
Ogni espressione ha un tipo di ritorno che indica il tipo dell'oggetto che sarà restituito dopo la valutazione dell'espressione.
Gli operatori sono predefiniti e si possono trovare tutti nella lista di definizione degli operatori.
Le espressioni vengono classificate in base al tipo di ritorno o al numero di operatori.
Nella definizione delle espressioni con il tipo del valore si intende una qualsiasi espressione che ritorna un oggetto dello stesso tipo.

Espressioni letterali

Un'espressione letterale è un qualsiasi valore puro di un tipo primitivo (number, string, bool).
Seguono alcuni esempi di espressioni letterali separata dalla virgola: 1, 0.2, "ciao mondo", true, false

Espressioni matematiche

Un'espressione matematica è una qualsiasi operazione che coinvolge gli operatori matematici, hanno come tipo di ritorno il tipo number
le seguenti sono tutte espressioni matematiche:

- Addizione `<numero> + <numero>`
- Sottrazione `<numero> - <numero>`
- Prodotto `<numero> * <numero>`
- Divisione `<numero> / <numero>`
- Potenza `<numero> ^ <numero>`
- Modulo `<numero> % <numero>`

l'ordine di precedenza per la valutazione segue quello matematico:

1. Potenza
2. Prodotto, divisione e Modulo
3. Addizione e Sottrazione

Espressioni di confronto

Le espressioni di confronto coinvolgono gli operatori di maggioranza e minoranza, stretta e larga, il tipo di ritorno dell'espressione è un booleano il cui valore è il risultato del confronto.

- Minore di `<espressione> < <espressione>`

- Maggiore di `<espressione> > <espressione>`
- Minore uguale di `<espressione> <= <espressione>`
- Maggiore uguale di `<espressione> >= <espressione>`

Espressioni di uguaglianza

Le espressioni di uguaglianza coinvolgono gli operatori di uguaglianza, il tipo di ritorno dell'espressione è booleano il cui valore è il risultato dell'uguaglianza.
Come per le espressioni di confronto, il tipo delle due espressioni operandi deve essere il medesimo per avere una espressione di uguaglianza legale.

- Uguale a `<espressione> == <espression>`
- Diverso da `<espressione> != <espression>`

Espressioni logiche condizionali

Le espressioni logiche condizionali coinvolgono gli operatori logici AND e OR, il tipo di ritorno dell'espressione è booleano il cui valore è il risultato dell'operazione logica.

1. AND `<booleano> && <booleano>`
2. OR `<booleano> || <booleano>`

Le espressioni logiche condizionali cercano di valutare il minimo possibile per ottimizzare i controlli. Se una espressione si rivela vera o falsa alla valutazione del primo operando l'operazione non valuterà il secondo operando ma ritornerà direttamente il corrispettivo risultato.

Espressioni logiche binarie

Le espressioni logiche binarie sono simili alle espressioni logiche condizionali ma valutano sempre entrambi gli operandi

1. AND `<booleano> & <booleano>`
2. OR `<booleano> | <booleano>`
3. XOR `<booleano> ^ <booleano>`
4. NOT `!<booleano>`

Espressioni di accesso

Le espressioni di accesso permettono d'interagire con i valori contenuti all'interno delle variabili. In base all'espressione si accederà a tipologie di oggetti diversi.

- Accesso a membro: `<oggetto>.<membro>` utilizza operatore di accesso `.` per accedere al valore di uno specifico membro di un oggetto. Il tipo di ritorno dell'espressione è il tipo del membro a cui si sta accedendo.

- Accesso tramite indice: `<lista>[<numero>]` utilizza l'operatore di accesso tramite indice `[<numero>]` per accedere allo specifico indice della lista definito dall'numero all'interno dell'operatore, ha come tipo di ritorno lo stesso tipo degli oggetti della lista.
- Accesso a variabile: `<identificatore>` l'identificatore unico deve appartenere ad variabile per poter essere legale. Accede al valore contenuto nella variabile, il tipo di ritorno dell'espressione è lo stesso tipo della variabile a cui si sta accedendo

Espressione tipo

Le espressioni tipo o type expression, sono tutte quelle espressioni che si occupano della gestione dei tipi degli oggetti:

- Controllo del tipo: `<oggetto> is <tipo>` controlla se l'oggetto è del tipo richiesto. L'espressione ha come tipo di ritorno un valore booleano cui valore corrisponde il risultato del controllo: vero se l'oggetto è del tipo richiesto, falso altrimenti.
- Conversione del tipo, type casting: `<oggetto> as <tipo>` il type casting é utilizzato per convertire un oggetto in un oggetto di tipo diverso, il tipo di ritorno é lo stesso del tipo richiesto. Nel caso la conversione sia ammissibile l'espressione ristornerà il nuovo oggetto con il tipo prestabilito, in caso contrario sarà ritornato un valore nullo.

Espressioni d'istanziazione

L'espressione d'istanziazione `new <tipo>()` è usata per creare una nuova istanza del tipo richiesto, il tipo di ritorno corrisponde al tipo richiesto.
L'istanziazione di un oggetto può essere seguita da una lista d'inizializzazione per popolare velocemente con dei valori il nuovo oggetto.

Espressione d'assegnamento

Sintassi: `<espressione di accesso> = <espressione>;`
L'espressione di assegnamento è utilizzata per assegnare il risultato di un espressione a una variabile, attributo o a uno specifico indice di una lista.
Per avere un espressione di assegnamento legale è necessario che i tipi di ritorno delle due espressioni siano gli stessi.
Il tipo di ritorno dell'espressione è nullo.

Espressione di chiamata

L'espressione di chiamata è utilizzata per chiamare l'esecuzione di una funzione con degli specifici argomenti definiti nella lista degli argomenti. Il tipo di ritorno dell'espressione corrisponde al tipo di ritorno della funzione.

Espressione giocatore

Una espressione giocatore, o player expression, è un espressione speciale utilizzata per ritornare un oggetto giocatore.
Gli oggetti giocatori sono identificati tramite un numero partendo da zero fino al numero di giocatori meno uno, questi identificatore numerici possono essere utilizzati all'interno delle espressioni giocatore, che equivale a una espressione matematica il cui risultato è ristretto per poter essere mappato a un giocatore.
La funzione che si occupa di riportare il risultato dell'espressione matematica nell'intervallo di mappatura dei giocatori è la seguente

$$\begin{cases} r\%P & r >= 0 \\ (P - |r|\%P)\%P & r < 0 \end{cases}$$

Dove:

- r: è il risultato dell'equivalente espressione matematica
- P: è il numero di giocatori
- %: è l'operatore modulo

Ordine di precedenza

L'ordine standard (senza modificatori di precedenza) di valutazione delle espressioni è il seguente:

1. Espressioni d'istanziazione
2. Espressioni di chiamata
3. Espressioni di accesso
4. Espressioni matematiche
5. Espressioni giocatore
6. Espressioni di confronto
7. Espressioni di uguaglianza
8. Espressioni logiche binarie
9. Espressioni logiche condizionali

Se non specificato altrimenti le espressioni all'interno di una stessa categoria si risolvo da sinistra verso destra.
Qualsiasi espressione può essere inserita all'interno di parentesi tonde per aumentare la priorità di valutazione, la valutazione partirà delle espressioni maggiormente innestate.

3.2.4 Istruzioni

Le istruzioni, o statements, sono i macro comandi del linguaggio, ovvero una combinazione di clausole ed espressioni usate per adempire a un compito preciso. Le istruzioni devono terminare con il carattere `;`, ma nel caso sia possibile definire un blocco d'istruzioni associato è possibile omettere il `;`

Istruzione di ritorno

Sintassi: `return [<espressione>];`
L'istruzione di ritorno è utilizzata per finire l'esecuzione di un effetto, ritornando il controllo al codice che ha chiamato la funzione, restituendo (o no) un risultato in base al tipo di ritorno della funzione.
Il risultato restituito proviene dalla valutazione della espressione definita nella istruzione, nel caso si abbia un tipo di ritorno senza tipo bisogna omettere la espressione.

Istruzione espressione

Sintassi: `<espressione>;`
l'istruzione espressione, o expression statement, serve per eseguire una certa espressione;

Istruzione di dichiarazione

Sintassi: `<tipo> <identificatore> = <espressione>;`
L'istruzione di dichiarazione è utilizzata per definire delle nuove variabili del tipo richiesto e inizializzate con il risultato dell'espressione alla destra dell'operatore d'assegnamento.
Le variabili dichiarate sono utilizzabili dopo la loro dichiarazione tramite l'identificatore, ovvero tramite una espressione di accesso.

Istruzione di selezione

Sintassi: `if(<espressione booleana>) <blocco istruzioni>`
L'istruzione di selezione è utilizza per selezionare se eseguire o no il blocco d'istruzioni associato, rispettivamente se la valutazione della espressione booleana abbia esito vero o falso.

Istruzione Else

Sintassi `else <blocco istruzioni>`.
L'istruzione else permette di eseguire il blocco d'istruzioni associato quando l'espressione di selezione precedente sia risultata false.
Una istruzione else è usabile solo direttamente successivamente a una istruzione di selezione.

Istruzione Else if

Sintassi: `else if(<espressione booleana>) <blocco istruzioni>`
L'istruzione else if equivale all'unione di un istruzione else e un Istruzione if poiché eseguirà il blocco d'istruzioni associato solamente se l'espressione booleana della

istruzione precedente ha avuto esito falso e se il risultato della espressione booleana dell'istruzione else if ha esito vero.
Come per l'istruzione else è possibile definirla solo direttamente successivamente a un Istruzione di selezione, è possibile concatenare molteplici else if poiché considerata anch'essa come istruzione di selezione.

Istruzione while

Sintassi: `while(<espressione boolena>) <blocco istruzioni>`
L'istruzione while fa parte della categoria delle istruzioni di ciclo, viene usata per ripetere il blocco d'istruzioni associato fino a che l'espressione booleana, valutata precedentemente a ogni esecuzione del blocco, non risulti falsa.

Istruzione for

Sintassi: `for([<espressione>` ¡istruzione di dichiarazione¿]; [¡espressione booleana¿]; [¡espressione¿]) ¡blocco istruzioni¿—
L'istruzione for é un altro modo per creare cicli, utilizzata principalmente come abbreviazione dell'istruzione while.
Si divide in tre parti principali:

1. Inizializzazione: è un espressione che viene valutata una volta all'inizio del ciclo,
   é legale definire al posto di un espressione una istruzione di dichiarazione (la variabile dichiarata avrà come ambito il blocco dell'istruzione for).

2. Espressione di controllo: definisce se continuare a eseguire il ciclo se risulta vera o terminarlo se risulta falsa.
   Se omessa verrà preso come default l'espressione letterale `true` creando un ciclo infinito.

3. Espressione post ciclo: è un espressione che viene valutata dopo l'esecuzione del blocco di istruzinoi del ciclo ma prima della valutazione della espressione di controllo.

Istruzione continue

Sintassi: `continue;` L'istruzione continue avvia una nuova iterazione dell'istruzione di ciclo più vicina che la racchiude.

Istruzione break

Sintassi: `break;` L'istruzione break termina l'istruzione di ciclo più vicina che la racchiude.

3.3 Classi

3.3.1 Attributi

3.3.2 Visibilità

3.3.3 Azioni

3.4 Eventi

3.5 Funzioni

3.6 Interactables

3.7 Mazzi

3.8 Giocatori

3.9 Boards

3.10 Ereditarietà

Capitolo 4

Conclusione