Progetto Programmazione per la Fisica A.A. 2024/25
Baba Is Us

Diego Arcari, Gabriele De Astis

Martedì 8 Luglio 2025

Modifiche dall'ultima consegna

A seguito dell'ultima consegna, in data 5 Luglio, abbiamo ottimizzato il codice sorgente nei seguenti modi:

- Reso varie funzioni e parametri const, per buona pratica, o constexpr, in modo da alleggerire la compilazione al runtime.
- Eliminato alcune funzioni inutilizzate e ottimizzato altre funzioni rendendole più corte, eseguendo la chiamata solo una volta o creando degli overload.
- Modificato il Livello 1 in modo da avere più soluzioni e il Livello 2 per far capire meglio la meccanica "Open".
- Risolto un potenziale rischio di memory leak non incontrato prima, riguardante la modifica di un vector a cui era associato un iteratore in un range for loop.
- Cambiato il processo di aggiunta delle proprietà in seguito alla riscrittura delle regole, evitando loop del tipo "Wall Is Wall"
- Aggiustato i test di conseguenza alle modifiche sovracitate.

1 Introduzione

Lo scopo di Baba Is Us è di ricreare la base del puzzle-game platformer indie 2D "Baba Is You" nel linguaggio C++20, aggiungendo alcune funzionalità di gioco non presenti nell'originale create liberamente da noi.

2 Ispirazione

Il gioco di riferimento pone il giocatore in vari livelli costruiti su due griglia bidimensionali (una per la parte delle sprite e una che tiene gli oggetti effettivi). I vari oggetti possono avere diverse proprietà. Essenziali nel gioco sono i Blocchi Parola, ossia delle parole che servono per creare le regole che determinano le possibili interazioni fra gli oggetti.

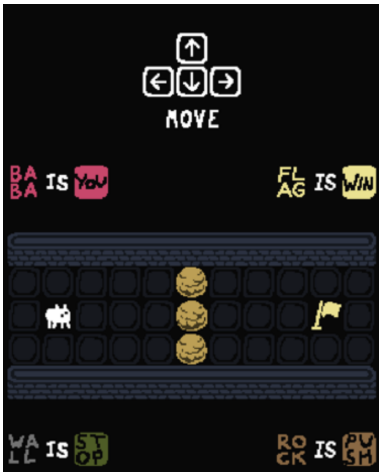

Figura 1: In questo caso, il gioco interpreta la regola "Baba Is You" dando al giocatore (You) il comando del coniglietto Baba, e "Flag Is Win" dando alla bandiera la proprietà Win, che permetterà di superare il livello quando ci andremo in contatto. Inoltre, i muri hanno la proprietà Stop, che li rende non-spostabili, mentre le rocce hanno la proprietà Push, che permette al giocatore di spostarle andandoci contro.

È importante notare che i Blocchi Parola, generalmente, sono spostabili dal giocatore, che può quindi modificare le regole del gioco a suo piacimento, lavorando con i blocchi a sua disposizione.

3 Librerie aggiuntive

Abbiamo fatto ampio uso della libreria grafica SFML, disponibile su https://smfl-dev.org/ o eseguendo sul terminale il seguente comando:

sudo apt install cmake ninja-build libsfml-dev

4 Scelte progettuali e implementative

Abbiamo incluso tutto il codice del folder src nel nostro namespace custom, Baba_Is_Us, affichè ogni file potesse accedere alle sue Classi liberamente.

1. enum_objecs.hpp
   Partendo da una enum Class nel file abbiamo creato vari "Type", divisi in Noun_Type, Icon_Noun_Type, Verb_Type e Property_Type, e che definiscono gli "Objects" che saranno posizionati nella griglia 2D.

2. objects.cpp e objects.hpp
   Ogni Object è caratterizzato da m_object, un std::vector di Type, che contiene l'identità di ciò che si trova in quella casella e le varie proprietà con le quali il giocatore può interagire o meno.

3. map.cpp e map.hpp
   Il namespace MapSize contiene le dimensioni della griglia, valori inizializzati che devono corrispondere al livello mandato in input (Sezione 6).

La Classe custom Map è composta da diversi membri e metodi, legati al visualizzare o al memorizzare lo stato del livello:

(a) m_objects è una a griglia 2D fatta con std::array di std::array degli Objects

(b) m_grid è la griglia di gioco bidimensionale, è un std::array di std::array di int corrispondenti agli enum Type. Serve a visualizzare il livello in ogni istante.

(c) alcuni dei vari metodi di Map servono a ottenere una copia i membri sovracitati, ma la restante parte invia come return una reference a questi, indicando che essi verranno modificati.

4. game.cpp e game.hpp
   La Classe Game è il motore di gioco: il metodo update() legge l'input della tastiera tramite window.pollEvent() e chiama di conseguenza il metodo movement(), che si occupa del controllo della posizione del giocatore, del movimento e dello stato del gioco (Playing, Won, o Lose) o il metodo interact(), ovvero l'azione di interagire con gli oggetti

5. rules.cpp e rules.hpp
   Ogni Rule è composta da 3 Type, di cui il secondo è sempre il Verb_Type "Is", e ne si può guardare il contenuto con il metodo getm_rule().
   La Classe Game ha anche un membro di tipo Rule Manager, che ha la lista di regole "attive" in ogni momento di gioco; le può togliere, aggiungere o controllarne il valore.
   Fuori da queste classi c'è la funzione conditions(), che confronta due oggetti e processa le interazioni tra i due tramite delle helper function (handleStop(), handleHot() ecc..).

6. main.cpp
   Il file main_prova.cpp si occupa di mantenere accesso il programma finchè si perde o si vince: grazie alle Classi sf::RenderWindow e sf::Event abbiamo potuto realizzare una schermata interattiva che mostrasse a schermo il gioco in ogni istante e che rispondesse agli input della tastiera. Il while loop principale neil main() chiama sempre la funzione Game::update() che aspetta un input dalla tastiera.

Gli sprite degli oggetti sono memorizzati in assets/png_progetto e sono visualizzati dai metodi Map::redraw() e Game::render() tramite l'uso delle Classi sf::Texture e sf::Sprite.

5 Istruzioni per eseguire il programma

Assicurasi in primis di avere CMake, Ninja e SFML versione 2.6.2 installati nel proprio ambiente di lavoro. Dopo aver scaricato e aperto il pacchetto baba_is_us.zip, spostarsi sul terminale all'interno della cartella baba_is_us (disponibile anche su GitHub con il nome PROJECT_B):

- MacOS: cd ∼/Downloads/baba_is_us
- Linux Ubuntu: cd ∼/baba_is_us

eseguire il seguente comando sul terminale:

cmake -S . -B build -G"Ninja Multi-Config"

Dopodichè, per eseguire uno dei seguenti comandi a seconda dello scopo:

1. Eseguire il debugging: cmake –build build –config Debug

2. Eseguire il debugging dei test: cmake –build build –config Debug –target test
3. Eseguire il programma senza debugging: cmake –build build –config Release
4. Eseguire i test senza debugging: cmake –build build –config Release –target test
5. Eseguire l'eseguibile nella build creata:
   (a) build/Debug/baba_is_us (eseguibile con Debug)
   (b) build/Debug/baba_is_us.t (test con Debug)
   (c) build/Release/baba_is_us (eseguibile)
   (d) build/Release/baba_is_us.t (eseguibile dei test)

6 Input e Output

Appena eseguito uno degli eseguibili verrà chiesto all'utente di inserire il numero del livello che intende giocare (da 1 a 4, per la prima volta si consiglia di selezionare il livello 1). Dopodiché apparirà a schermo la finestra "Baba Is Us" con il livello selezionato.
Input di gioco:

- WASD: Movimento 2D.
- Space: Interagisci con le leve adiacenti.
- Esc: Esci dal gioco.

Alla fine del livello, ovvero quando si tocca un oggetto con attributo Win, o nessun oggetto ha attributo You, il programma termina, con una simpatica scritta se si ha vinto o perso.
Informazioni su eventuali output aggiuntivi post-esecuzione sono nella Sezione 9.1

7 Testing tramite Doctest

La quasi totalità del codice scritto da noi controlla già da solo che i valori letti in input dai file .txt in assets/levels/ siano convertibili in modo sicuro in enum Type e in sprite da disegnare tramite degli assert, quindi i primi test si accertano che la Map abbia convertito correttamente l'input in oggetti all'interno di m_objects.
Gli altri test si occupano di verificare che tutte le funzioni di ogni singolo file si comportino come dovrebbero. La Classe Game richiede più attenzione, dato che si occupa del movimento di vari oggetti e, in caso di spostamento di un Blocco Parola, della modifica delle regole logiche.
In ogni caso, essendo questo un gioco con un numero di interazioni possibili che aumentano fattorialmente, abbiamo provato a testare un numero che ci sembra adeguato di queste combinazioni.
I tre file level.txt dedicati ai testing forniscono situazioni ideali per verificare il corretto funzionamento di tutte le funzioni di ogni file.

8 Uso di Intelligenza Artificiale

Durante lo sviluppo del codice si è fatto uso di AI (chatGPT) principalmente per correggere gli errori che risultavano incomprensibili a noi, come problemi legati alla libreria SFML, errori riportati nel terminale o errori la cui soluzione, dopo avere speso un tempo non irrisorio per ripercorrere i

passaggi del codice, ci sfuggiva. Tra i pochi esempi, Map::pathFinder() poteva risultare in un loop infinito se non si faceva attenzione alla direzione che si controllava. In un altro caso, chatGPT ha consigliato un metodo per risolvere un problema legato al fatto che la build di CMake non riusciva a trovare il contenuto della sub-directory assets mentre si eseguivano i test.

9 Ulteriori Informazioni

9.1 Memory leak

Sono state incontrate, durante lo sviluppo su Ubuntu, delle memory leak: non si conosce precisamente la loro origine, ma sospettiamo che dipenda da librerie come sfml o dall'architettura del computer e non da noi, data la presenza di <unknown module> nei messaggi riportati nel terminale e dal fatto che non viene usata l'allocazione dinamica.
Questi episodi non si sono verificati su MacOS, ma potrebbero capitare su altri sistemi operativi.

9.2 Regole del gioco

9.2.1 Azioni permesse dalle frasi logiche

- una frase deve essere in orizzontale o verticale, ma una parola della regola può essere condivisa da più frasi (e.g: due frasi a croce sono permesse);
- una frase può contenere 2 noun (Baba Is Wall), nel qual caso Baba riceverà i tipi Wall e qualunque altro tipo di Wall;
- Push funziona su tutto, tranne se un oggetto è Melt;
- in una fila di ingranaggi accesi da una leva ancora con switch, se aggiungi un ingranaggio alla fila, devi riaccendere la leva;

9.2.2 Priorità delle Property Type

Di seguito sono elencate le regole che modificano gli oggetti a seconda dei tipi che hanno, dalla più "potente" alla meno "potente" (N.B: X vince su Y vuol dire che X ha più peso di Y nelle interazioni di conditions())

- Defeat e Melt vince su: tutto;
- Shut vince su: tutto a meno di Open;
- Win vince su: tutto a meno di Defeat e Melt;
- Spin vince su: Push
- Push vince su: Stop, Hot

Gli altri tipi invece non modificano il gioco e permettono ogni movimento.