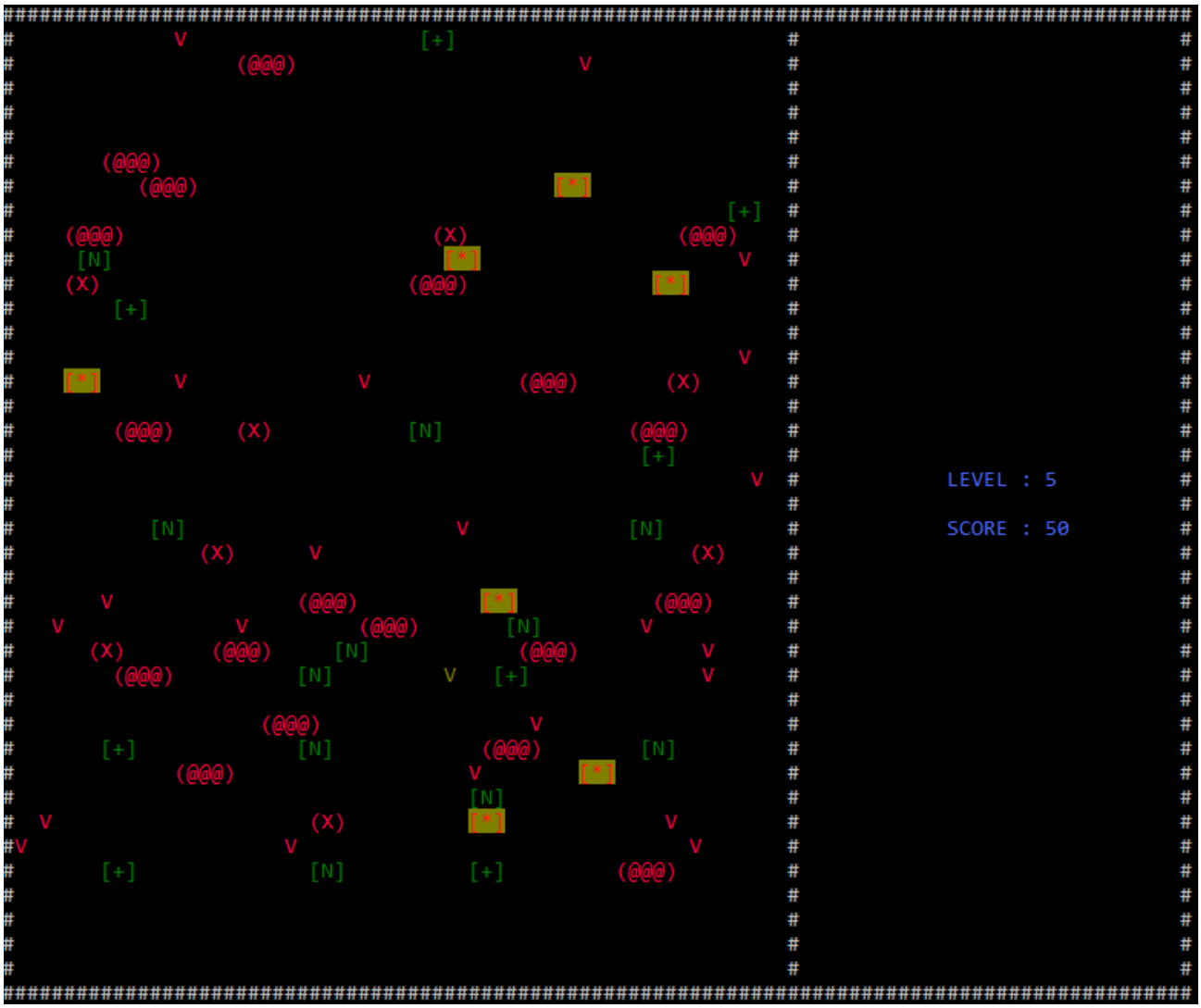

Race Car in C++

Ascii game

Lorenzo Tomesani

Scienze Informatiche

Anno: 2019/2020

Introduzione

Il progetto si basa su una gara automobilistica a punti. Il player che è una macchina ('V' gialla), deve raccogliere punti (gli oggetti in verde) evitando di colpire gli ostacoli (gli oggetti in rosso). L'oggetto in rosso con sfondo giallo consente di essere immune a due collisioni con gli ostacoli. Raggiunto un certo punteggio, il player passa al livello successivo. All'aumentare del livello, aumentano gli oggetti che possono spawnare sulla stessa riga e da un determinato livello in poi (il quinto) aumenta leggermente la velocità con cui gli oggetti si muovono in avanti. Se il player scende sotto un certo punteggio, torna al livello precedente. Il player perde quando lo score raggiunge lo 0.

Ncurses

Per poter gestire il terminale in maniera più semplice è stata usata la libreria *Ncurses*. È progettata per sistemi Unix/Unix-like, ragion per cui non è possibile eseguire il nostro programma sul sistema Windows (in windows 10 è possibile abilitare il sottosistema Windows per Linux). Ncurses viene linkato dinamicamente nella compilazione (-lncurses).
I metodi più importanti che vengono usati in tutto il codice sono:

- wmove(stdscr, y, x): sposta il cursore alla posizione (x, y) nel terminale stdscr;
- wprintw(stdscr, string): stampa alla posizione del cursore la stringa (o meglio la const char) che viene passata in input.

Sempre tramite Ncurses riusciamo a gestire i vari colori del terminale:

- init_pair(ID, COLORE_CHAR, COLORE_SFONDO): associa all'ID (intero) i due colori passati in input;
- attron(COLOR_PAIR(ID))/ attroff(COLOR_PAIR(ID)): vengono attivati o disattivati i colori associati all'ID passato in input.

I metodi usati sono di più ed è possibile trovarli tutti, con annessa spiegazione, in una guida usata durante lo sviluppo del progetto: ***http://www.cs.ukzn.ac.za/~hughm/os/notes/ncurses.html***.

Compilazione

Per la compilazione è stato usato *Make*, un utility di sistema che automatizza la compilazione di software usando un file che è chiamato "Makefile". Un Makefile è formato da regole di dipendenza nella forma:

target: dipendenze

All’interno del file troviamo delle variabili speciali, dette variabili automatiche, e funzioni che ci aiutano nella scrittura del file.

Variabili Automatiche Usate:

- $@: nome del target della regola;
- $<: nome della dipendenza che appare per prima nella regola;
- $^: lista dei nomi di tutte le dipendenze;
- $(variabile): uso delle variabili precedentemente definite all’interno delle regole.

Funzioni:

- $(wildcard pattern): viene restituito in output una lista dei file che rispettano il patter (wildcard appartiene a linux);
- $(addprefix prefix, nomi_file): aggiunge ai file passati in input un prefisso;
- $(patsubst pattern, replace, text): sostituisce al pattern, se presente nel testo, il replace;
- $(notdir *path*…): elimina la parte di directory di uno o più path passato/i in input.

È possibile trovare il file già compilato nella cartella bin.

Descrizione codice

Struttura generale

Le classi principali sono cinque e gestiscono gli aspetti più importanti del programma:

- *Player;*
- *Object;*
- *Level;*
- *Collision;*
- *Map;*

Ad ognuna di esse corrispondono due file: un header e un codice sorgente.

Andiamo ad analizzare meglio le classi.

RaceCar.cpp

Non è una classe, ma è il file sorgente contenente il main dove viene istanziata la classe map e inizializzato il programma.

Map.cpp

Classe che gestisce l'interazione dei vari oggetti con il player, il menù iniziale e il controllo degli input durante il game.

Quando il metodo initWindow() viene chiamato vengono disegnati il campo da gioco, le informazioni riguardanti i punti, il livello e il player stesso e passato il controllo al metodo inGameController().

Dentro questo metodo viene fatto il check degli input dell'utente, il check delle collisioni (tra player/oggetto e player/muro) e degli aggiornamenti delle informazioni stampate a video. Nel caso in cui il player vada contro il bordo di gioco, viene riportato alla posizione iniziale. Durante il game è possibile mettere in pausa il gioco tramite p o abbandonare la partita, prima che lo score scenda a 0, premendo q.

Object.cpp

Gli oggetti sono gestiti da una linked list formata dalla struct "desc_object" in cui sono presenti:

1) due interi che determinano la x e la y dell'oggetto;
2) una const char che è la rappresentazione dell'oggetto stesso;
3) e un puntatore dello stesso tipo della struttura stessa.

La selezione della const char dell'oggetto è randomica, infatti viene usato rand() per selezionare un intero che va da 0 a 5 con probabilità diversa per ogni intero. È possibile trovare le loro rappresentazioni, l'associazione intero-oggetto, gli effetti e le percentuali dentro al file object_char.hpp. Il file viene importato nell'header di Object.

Il metodo di generazione degli oggetti (addRowObjects()) è chiamato nel main quando l'intero clock_cycle raggiunge il valore spawn. Oltre che generare nuovi oggetti, tutti gli elementi della lista vengono spostasti avanti di una y. Una volta che la y di un certo oggetto raggiunge la fine dello schermo, esso viene cancellato dalla lista.

Per mantere privata la lista degli oggetti, ho preso la decisione di valutare la collisione tra player e oggetto all'interno della classe Object.cpp anziché Collision.cpp.

Player.cpp

Classe che gestisce il protagonista del gioco, essa permette lo spostamento tramite W-A-S-D della macchina comandata dal player, le informazioni sullo score e sul livello (decrementare o aumentare a seconda dei casi). Da questa Classe è inoltre possibile fare il set dell'attributo "immunity"

intero dopo che il player ha fatto collisione con l'oggetto star. L'immunità non è cumulabile.

Level.cpp

Gestisce le informazioni stampate sulla destra dell'area di gioco:

- aggiorna le informazioni quando richiesto;
- controlla che lo score sia abbastanza alto per il livello e nel caso non lo sia diminuisce il livello. Se lo score è più alto di quello richiesto per il livello corrente viene aumentato il livello;
- stampa la scritta di Game Over una volta che il player perde tutti i punti.

Collision.cpp

Classe che controlla direttamente se il player si scontra con il bordo/muro di gioco e passa il controllo all'Object.cpp per i motivi spiegati sopra, quando bisogna fare il check della collisione tra player e oggetto. Il metodo a cui viene passato il controllo è checkCollisionPlayer con il quale fa un return con 7 possibili valori:

- i 6 oggetti possibili (0 – 5);
- se non c'è collisione fa il return di -1.

Nel caso in cui ci sia una collisione, differenzia i 6 scenari possibili ed intraprende le azioni da eseguire nei vari casi.