Survival Race

Survival Race è un progetto realizzato da Han Chu, Yuqi Zhang e Shicheng Liu per il corso di Programmazione (A.A. 2019-2020) della laurea triennale in Informatica dell'Università di Bologna.

Descrizione del gioco
È un gioco di corsa automobilistica arcade. Il player è una macchina e dovrà percorrere una strada infinita evitando alcuni ostacoli ed accumulando punti. Arrivati ad un certo punteggio, si salirà di livello e più aumenta il livello, più aumenta il numero di oggetti sulla strada ed aumenterà la velocità della macchina. Se si scontra con un ostacolo, il punteggio si abbassa e se non supera il punteggio minimo richiesto per tale livello, si scende al livello precedente, ripercorrendo la strada già giocata precedentemente. Il gioco termina se il punteggio scende a 0 oppure se si clicca un tasto specifico.

Requisiti
- La mappa viene generata un livello alla volta dinamicamente
- La mappa deve essere implementata in una semplice grafica ASCII
- È vietato usare librerie grafiche
- Ogni categoria di personaggi/strumenti del gioco è rappresentato da un simbolo diverso
- Non esistono traguardi
- Si passa al livello successivo raggiungendo un certo punteggio (ad esempio 1000 punti)
- Si torna al livello precedente se si scende sotto un certo punteggio
- Più si sale di livello, maggiore deve essere la difficoltà di gioco.

Implementazione

Un modo per realizzare gli oggetti (macchina, ostacoli e strada) senza utilizzare librerie grafiche è quella di rappresentare la macchina e gli ostacoli con dei caratteri, mentre la strada si può realizzarla con una matrice di caratteri, ovvero un array bidimensionale. Abbiamo usato degli array statici, e siccome la strada è infinita, sarà composta da una lista di array.
Gli ostacoli essendo fermi sulla strada possono essere implementati internamente all'oggetto "strada", mentre la macchina abbiamo scelto di definirla in una classe a parte.
L'idea alla base è la definizione di diversi oggetti (le classi *Car*, *Road*, *Points*) i quali interagiscono tra di loro tramite delle funzioni (*Controls*, *Graphics*). Inoltre vi sono alcune funzioni ausiliarie come *Locate* e tutte le funzioni di *Words*. Infine, il file *Debug* serve per controllare alcuni valori durante l'esecuzione del programma e verificare la sua correttezza.

Librerie esterne
- Windows.h (*SetConsoleCursorPosition*, *GetStdHandle*, *Sleep)*
- conio.h (*_kbhit*, *_getch)*
- iostream (*std::cout*, *std::endl*)
- stdlib.h (*system*, *rand*, *srand*)
- time.h (*time*)

OGGETTI:

Car
È l'oggetto che rappresenta la macchina, ed i suoi campi sono il carattere *symbol,* che indica la macchina, e *x_pos*, che indica la sua posizione sull'asse delle ascisse. I suoi metodi sono il costruttore, che imposta la macchina al centro della strada, alcune funzioni per ottenere e cambiare il valore di *x_pos*, e la funzione di stampare la macchina in base alla sua *x_pos*.

Points
È l'oggetto che indica il punteggio del gioco. Ha un solo valore (il punteggio), e i suoi metodi sono il costruttore e alcune funzioni per inizializzare, ottenere e modificare il punteggio.

Road
È la strada che la macchina percorre. Essendo il gioco a livelli ed infinito, abbiamo definito la strada (*Road*) come una lista ordinata contenente i livelli in ordine crescente (*Level*); ogni livello conterrà una parte della strada. Dato che si sale di livello in base al punteggio (e non alla strada percorsa), e quindi un giocatore potrebbe restare al livello "x" per sempre, in ogni livello la porzione di strada è implementato con una lista di segmenti (*Segment*), creando una lista di liste.
Con una descrizione bottom-up, *Segment* è la struttura base che contiene una porzione di strada definita tramite una matrice di caratteri (*field*) e un puntatore a segment (*next*); ogni livello del gioco è implementato con la struttura *Level*, che ha un puntatore a *Segment*, un identificatore a se stesso (*difficulty*), un indicatore della velocità del gioco al suo livello (*speed*) e un puntatore a Level (*next*), che punta al prossimo livello.
Quindi *Road* (figura 1) ha un puntatore a *Level*, dove sono memorizzate tutti i livelli e le strade, e ha alcuni campi che indicano la difficoltà attuale (*current_difficulty*), la velocità attuale (*current_speed*), il segmento di strada che viene stampato sul terminale (current_segment) e il segmento successivo (*next_segment*). Il puntatore *next_segment* è essenziale poiché serve per stampare sul terminale la strada quando non basta un solo segmento e server per il cambio di livello.
I suoi metodi sono:

- Il costruttore, che crea il primo livello e il primo segmento del gioco.
- Alcune funzioni *get* che ritornano il valore dei campi del Road.
- *new_level()*, che crea (se non esiste) il livello successivo a quello attuale, iterando la lista di livelli tante volte quanto il numero di *current_difficulty.*
- *new_segment(int difficulty)*, che crea un segmento per il livello con difficoltà *difficulty*, e lo aggiunge in coda alla lista corretta (se esiste una lista). Esso crea e inizializza la strada, definisce il numero di ostacoli in base alla difficoltà, li assegna randomicamente nella matrice ed infine cerca il livello giusto e lo inserisce in coda alla lista di segmenti.
- *shift(int y)*, che con l'argomento in input controlla se il *current_segment* è stato stampato del tutto (ovvero che alla prossima iterazione il segmento puntato da *current_segment* non verrà più visto sul terminale), e sostituisce *current_segment* con il suo successore *next_segment*.
- *after(int y)*, che con l'argomento in input controlla se la stampa sul terminale richiede uno o due segmenti; se servono 2 segmenti controlla l'esistenza di *next_segment*, e se è negativo crea il segmento dello stesso livello.
- *level_up()*, che cambia il puntatore *next_segment* col primo segmento del livello successivo, iterando *current_difficulty* più una volta sui livelli.
- *level_down()*, che cambia il puntatore next_segment col primo segmento del livello precedente, interando *current_difficulty* meno una vota sui livelli.
- *print(int y)*, che stampa una porzione di strada sul terminale. Dato che la strada è continua ed è implementata con una lista di segmenti, in base all'indice y si decide se stampare un solo segmento oppure due segmenti.

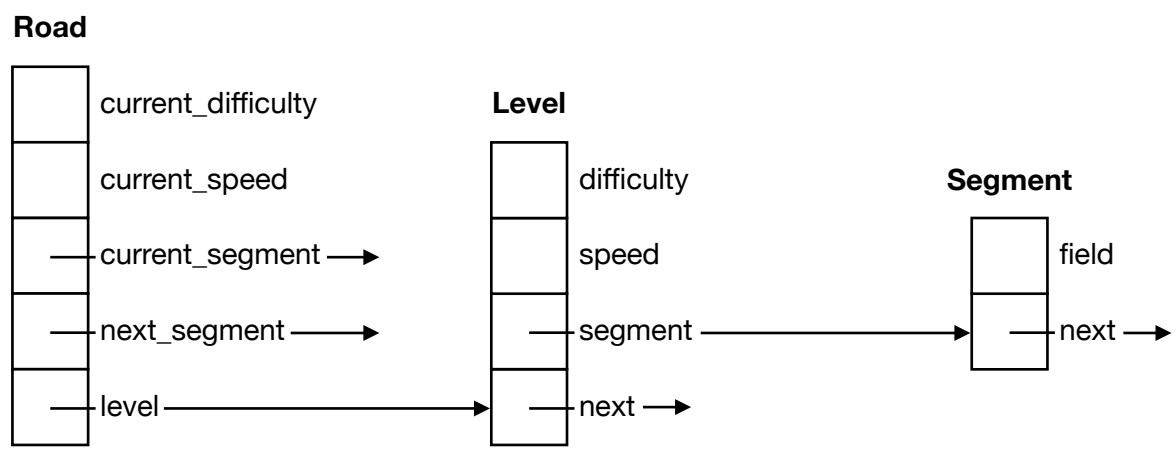

Figura 1: la struttura Road, Level e Segment.

INTERAZIONI:

Controls
Ha il compito di far interagire gli oggetti, e le sue funzioni sono:
- *Move(Car& car, Points& points)*, che muove la macchina in base al tasto in input che si clicca. Si passano gli argomenti per riferimento poiché tale funzione va a modificare il valore degli oggetti. Per rilevare la presenza di un tasto cliccato si usa *_kbhit()* mentre per catturarlo si usa *_getch()*.
  La macchina si può muovere solo a sinistra e a destra, ed è vincolato dal bordo, mentre il punteggio viene passato come parametro poiché per uscire dal gioco basta azzerare i punti.
- *Hit(Car& car, Points& points, Road road, int index)*, che controlla se la macchina si è scontrata con un oggetto sulla strada. Per identificare se c'è stato uno scontro, bisogna localizzare la posizione della macchina sulla strada, che è possibile grazie all'indice *index* (indirettamente) che indica l'altezza del segmento e alla posizione della macchina, che ne indica la profondità. In base al tipo di oggetto che c'è si vanno a modificare i valori del punteggio.
- *Change_Level(Points points, Road& road, int index)*, che controlla se bisogna cambiare il livello all'iterazione precedente alla stampa di 2 segmenti sul terminale. Dunque controlla il punteggio e se supera il range di punteggio per uno specifico livello, si sale fino al livello del punteggio (che non per forza è il livello immediatamente successivo), oppure fa il contrario. Il range di valori di un livello è:

Graphics
Contiene le funzioni che si occupano della grafica. Durante la realizzazione abbiamo utilizzato la tabella ASCII estesa, ma alcuni computer del team non riuscivano a leggerlo, quindi abbiamo adottato anche una grafica più semplice nel caso non funzionasse quella principale; la procedura è scritta come commento su Graphics.h.
Le funzioni sono:
- *Border()*, che disegna il bordo del gioco, facendo due cicli per stampare le colonne e le righe, e poi stampa gli angoli.
- *Intro()*, che stampa il bordo e un messaggio di benvenuto.
- *Score(int score, int difficulty)*, che stampa il bordo e divide la schermata in due blocchi, in una parte c'è il gioco e sulla colonna di destra ci sono delle informazioni utili (il punteggio, il livello, le istruzioni e la legenda).
- *TheEnd()*, che stampa un messaggio finale.

LIBRERIE AUSILIARIE:

Words
È una libreria ausiliaria a *Graphics.cpp* dove vi sono le funzioni per stampare dei caratteri. Non avendo nessuna libreria grafica, per stampare dei caratteri in maniera più grande abbiamo scelto di rappresentarlo in maniera "cubitale". Tale libreria stampa i caratteri con tale "font", inoltre richiede due parametri in input che sono le coordinate sul terminale di dove far partire la stampa.
Poiché le parole vengono scritte su più caselle abbiamo scelto di scrivere una lettera una riga per volta (incrementando la posizione dell'altezza per ogni riga nuova), e dato che non c'è uno standard per la dimensione di un carattere, abbiamo deciso di passare per parametro la coordinata delle ascisse, in maniera tale che esso venga incrementato nella funzione e non all'esterno.

Locate
È una libreria ausiliaria composta solo da una funzione *Locate*, che serve a posizionare il cursore nel terminale in una posizione (x, y). L'origine (0, 0) è dato dall'angolo in alto a sinistra, e la sua altezza (y) si sviluppa lungo l'altezza del terminale, dall'alto verso il basso.

Debug
È una libreria che serve per controllare alcuni valori per verificare la correttezza dei requisiti del gioco. Le sue funzioni sono la stampa di alcuni valori.
- *print_index(int index, int x, int y)*, che stampa il valore dell'indice *index*, esso non coincide per forza col punteggio, nonostante per ogni iterazione si incrementa l'indice e il punteggio di un punto, poiché quest'ultimo subisce cambiamenti da parte di Hit.

- *print_current_speed(Road road, int x, int y)*, che stampa la velocità attuale del gioco in millisecondi, ovvero ogni quanto tempo viene fatto un'iterazione.
- *print_current_segment(Road road, int x, int y)*, che stampa il valore del puntatore attuale.
- *print_next_segment*(Road road, int x, int y), che stampa il valore del puntatore successivo.
- *print_all_segment(Road road, int x, int y)* che stampa tutti i segmenti del gioco, divisi per livelli. Esso è utile per capire quando si scende di livello se si stampa i segmenti vecchi. Inoltre si può vedere le variazioni del puntatore *next_segment*.