Nome e Cognome: Matteo Pasotto **Matricola:** 0000922259
Nome e Cognome: Matteo Vannucchi **Matricola:** 0000937275
Nome e Cognome: Rocco Pastore **Matricola:** 0000923786
Nome e Cognome: Pasquale Ricciuli **Matricola:** 0000923980

Anno scolastico: 2019/2020

Titolo: Progetto Car++

Obiettivo: Si vuole realizzare un videogioco con il linguaggio c++, che preso in input un comando sposta la macchina del giocatore a destra o a sinistra, il gioco finisce quando il punteggio va a zero o la benzina finisce.

Class menu:

Il menù è suddiviso in più pagine, la pagina principale è formata dal nome del gioco.
Nella seconda pagina troviamo il menù da dove si possono leggere le istruzioni, controllare i migliori punteggi, uscire dal gioco e avviarlo.
Il titolo e il menù vengono posizionati tramite delle coordinate gestite dalla funzione void menu::coord().
L'interazione avviene premendo i tasti w e s per salire e scendere all'interno del menù.

Class Object:

La classe object contiene i singoli oggetti (es: aria, ostacoli, benzina), tali oggetti sono utilizzati da più classi, come LevelGenerator oppure Interaction. La classe Object viene inizializzata per default di tipo air. In modo informale la classe Object può considerarsi l'elemento base della generazione dei livelli e dell'interazione tra macchina e ciò che la circonda.

Class LevelGenerator:

La classe LevelGenerator si occupa della creazione dei livelli. Tra i requisiti richiesti vi è necessità che il livello venga memorizzato, pertanto abbiamo deciso di associare alla creazione del livello un seed e un livello di difficoltà. Per la creazione abbiamo utilizzato un generatore pseudo-casuale, il mersenne twister engine, il quale dato un input (seed) genererà sempre la stessa sequenza di input. In questo modo per memorizzare un livello basta mantenere il seed.

Public Method:

Il livello è stato creato immaginando una serie di righe le quali sono composte di default da spazi vuoti più oggetti quali ostacoli o benzina. Una classe, dopo che il livello è stato creato, può richiamare la funzione Object* GetRow(); che restituisce la riga in formato oggetto(più avanti verrà specificato) oppure la funzione void PrintRow(); che stampa a video la riga del

livello in formato testuale (ogni volta che si richiama la funzione viene ritornata/stampata la riga successiva).
L'ultima funzione richiamabile dall'utente è quella di restart che permette di ricominciare nuovamente il livello.

Protected Method:

Tra i metodi protected vi è il metodo principale ovvero RowGenerator(). Questo metodo genera una riga del livello che può contenere casualmente i vari ostacoli i quali sono definiti dalla classe ObjectType. Il numero di ostacoli incrementa con l'avanzare dei livelli.

Class Map:

La classe Map si occupa di tenere in memoria tutti i vari oggetti della mappa, mettendo a disposizione varie operazioni come l'aggiunta di una nuova riga (per far scorrere la mappa) e la stampa della mappa.

Implementazione:

Per mantenere gli oggetti in memoria è stata implementata una matrice che contiene tutti gli oggetti presenti nella mappa, ovvero mantiene i vari ostacoli, i bonus e gli spazi vuoti (che vengono considerati degli Object di tipo 'air'). I muri non vengono mantenuti in memoria non essendo oggetti che hanno interazioni particolari.

Scorrimento:

L'aggiunta di una riga alla mappa (PushNewRow) è stata implementate in modo tale da non andare ad intaccare troppo le prestazioni del programma. Invece di andare ad ogni frame ad aggiornare tutta la mappa, si tengono in memoria due indici StartLine e EndLine che rispettivamente indicano la prima riga della mappa (quella più in alto) e l'ultima riga della mappa (quella più in basso). Quando viene eseguito PushNewRow aggiorno il valore indicizzato da EndLine con la nuova riga e si aggiornano i valori di EndLine e StartLine.

Stampa:

La stampa della mappa avviene stampando solo gli oggetti "utili", cioè gli oggetti come ostacoli e bonus, invece oggetti come i muri (che vengono stampati all'inizio e mai modificati) o gli spazi vuoti, per evitare un carico inutile sulle performance, non vengono stampati. Oltre a stampare il frame attuale bisogna cancellare il frame precedente. Invece di fare un semplice System("cls") che costerebbe molto in termini di prestazioni e per evitare un effetto di "lampeggiamento", quello che viene cancellato sono solo le "scie" degli oggetti: se ho un oggetto in posizione (x,y), cancellerò (se non c'è già un altro oggetto) la posizione (x, y-1), dando un effetto scia che lo rende fluido ed eliminando il lampeggiamento.

Movimento e interazione della macchina

Per quanto riguarda la protagonista del gioco, la macchina, si muove facendo uso delle funzioni _kbhit() e _getch(). La prima controlla continuamente se viene premuto un qualsiasi tasto dal giocatore, e in caso positivo entra in gioco la seconda, che ne individua il tasto preciso. In questo modo la macchina si muove, se viene premuto il tasto A va a sinistra, D va a destra e così via. Il movimento della macchina con le rispettive scelte del giocatore, sono strettamente legate all'interazione della macchina con ogni singolo oggetto. Per effettuare ciò, si verifica se le coordinate della macchina contrastano le coordinate degli oggetti presenti in quel momento. In caso positivo, si innesca il meccanismo che porterà a vincere o a perdere il gioco. Nel nostro caso gli oggetti in questione sono 4, 3 che portano ad un malus e uno solo che porta ad un bonus. Per i primi, parliamo di un ostacolo, in rosso

sulla mappa, che è il malus meno dannoso tra i tre, poi si passa ad una fossa, in giallo, che farà perdere svariati punti, ed infine ci si può ritrovare davanti ad un chiodo, in verde, il peggiore. L'unico oggetto positivo del giocatore è la benzina, in blu, che servirà per andare avanti nel gioco. Per gestire l'interazione, viene semplicemente incrementato o decrementato il valore del punteggio e della benzina, sempre mostrati in alto a sinistra. E' sempre nel file interaction.cpp che viene gestito il progresso del gioco. Infatti tiene conto del punteggio e della benzina, e tramite questa pagina potrete controllare lo stato attuale dei due elementi. Una volta terminato l'uno o l'altro, comparirà la scritta game over.

Class Score:

La classe Score registra il punteggio finale dell'utente a condizione che questo sia tra i 10 punteggi più alti in un file txt "BestScore.txt". Tra i metodi pubblici abbiamo PrintScore(); che stampa i primi 10 punteggi e Inserisci(int punteggio); il quale controlla se il punteggio supera l'ultimo in classifica e registra nome, punteggio e posizionamento dell'utente.
I metodi Protected invece riguardano la scrittura e la lettura del file txt e sono rispettivamente ReadFile() e WriteFile().

Class Screen:

La classe Screen implementa tutti i metodi a noi utili per semplificarci la scrittura del programma, come la gestione del colore, del cursore del mouse e la stampa in una posizione specifica. È stato deciso di implementare questa classe per problemi di compatibilità con librerie per terminale più performanti e professionali.

Class Game:

La classe Game è la classe centrale di tutto il programma. Il ruolo della classe Game è quello di tramite e di gestore di tutte le altre classi del gioco, come la mappa, la macchina, lo score e i menu.
Implementazione:
La classe Game permette di eseguire varie operazione sul gioco stesso, come terminare il gioco, passare al livello successivo/precedente e gestisce anche quando il giocatore perde, sia per punteggio, sia per benzina. Per far tutto ciò Game mantiene in memoria vari elementi, come i riferimenti al giocatore, alla mappa e ad una lista di tutti i livelli fino ad ora visti.
Il metodo principale di Game è GameLoop, chiamato ad ogni frame (la velocità di un frame varia in base alla difficoltà del livello attuale), che va appunto a gestire tutti gli oggetti come la macchina e la mappa, oltre a controllare l'avanzamento e l'arretramento di livello. Quando si passa da un livello ad un altro si è deciso di mettere un intermezzo nel quale non ci sono né ostacoli né bonus e dove lo score non viene aggiornato.