Progetto

esame di Programmazione 2024/2025

Introduzione

Il progetto è stato svolto in coppia dagli studenti di informatica Esposito Francesco (terzo anno) e Belbusti Luca (primo anno), il lavoro in questione prevedeva la realizzazione del celebre gioco Snake tramite il linguaggio di programmazione object oriented C++ e la libreria grafica ncurses. Le specifiche da rispettare richiedevano la presenza di una classifica con punteggi ordinati in ordine descrescente accessibile tramite un menù di gioco, il suddetto menu permette anche di iniziare una nuova partita.

Il gioco non si sviluppa su mappe diverse ma su livelli, ciascuno con una velocità di movimento differente e proporzionale alla difficoltà del livello stesso. Un timer determina la fine della partita, a quel punto il giocatore si vedrà assegnato un punteggio che tiene conto del numero di mele mangiate e anche del livello di difficoltà scelto, premiando con un moltiplicatore i players che avranno optato per un livello più arduo.

Ci sono altre vie che possono portare il giocatore al game over, ad esempio se il serpente collide con se stesso o se il giocatore decide di uscire volontariamente dal gioco premendo un tasto a questo designato ( q nel nostro caso ).

La principale differenza dal gioco originale è l’assenza della feature che per ogni mela mangiata dal serpente aggiunge al suo corpo un segmento, rendendo quindi più complicato muoversi col procedere della partita.

Divisione compiti

*Belbusti Luca:*

* Snake e sistema di movimento
* Mappa di gioco
* Spawn e dinamiche della mela
* Timer
* Ciclo di gioco

*Esposito Francesco:*

* Salvataggio e lettura su e da file
* Punteggio ( Bonus, classifica, highscore )
* Sistema di livelli e difficoltà
* Menu

Implementazione delle specifiche

Classe snake

La classe Snake gestisce il movimento del serpente, eventuali cambi di direzione, le collisioni con sé stesso, e il disegno dello stesso sulla griglia di gioco. Per fare ciò tra i campi della classe troviamo la variabile “length” che tiene traccia della lunghezza predefinita del serpente; “tail” e “head” che sono strutture del tipo “coordinates” in modo tale da tenere sempre traccia delle coordinate di testa e coda del serpente; “position” ovvero un array con le coordinate delle parti del serpente(all’indice 0 dell’array è contenuta la posizione della coda e, a scalare, in ultima posizione quella della testa); e infine “direction” che contiene una direzione tra quelle dell enum DIRECTION (U,D,L,R).

Il costruttore della classe crea una matrice della grandezza del campo da gioco per mappare il serpente e successivamente imposta a true le caselle in cui si trova inizialmente (in posizione centrata allo schermo) e salva le posizioni di testa e coda che serviranno per il movimento.

Il metodo Move restituisce un booleano per controllare che quando il serpente si muove, non si scontri con sé stesso (restituisce false in questo caso, true altrimenti): all’interno del metodo viene creata una nuova testa in base alla direzione in cui si muove il serpente, con la funzione CheckSelfCollision() si controlla che la nuova testa non faccia già parte del corpo, in caso contrario viene eliminata la coda del serpente mettendo a false quella casella, vengono fatti scalare di uno tutti i componenti dell’array position in modo da avere le coordinate della nuova coda all indice 0 e viene inserita in ultima posizione dell’array la posizione della testa. In questo modo il movimento del serpente viene realizzato togliendo un pezzo dalla coda e aggiungendolo in testa.

Andando avanti il metodo ChangeDirection(int key) riceve come input l’input dell’utente e cambia la direzione vietando i cambi sullo stesso asse.

Infine, oltre ai metodi get, troviamo il metodo Draw() che semplicemente passa l’intera matrice e disegna con il carattere “0” le casella segnate con true, in cui quindi è presente il corpo del serpente, e con il carattere “X” la testa del serpente.

Classe Apple

La classe Apple disegna in punti casuali le mele che il serpente deve mangiare e, quando ciò avviene, rigenera un’altra mela in un altro punto casuale.

Tra i metodi è da notare NewApplePosition() che viene chiamato dal ciclo di gioco quando il serpente mangia una mela: viene quindi generata un'altra mela in posizione casuale facendo però attenzione che la coordinata non sia occupata dal corpo del serpente.

Classe Board

La classe Board si occupa di generare il campo di gioco e di stampare a schermo il timer, lo score attuale, l’highestscore e il livello in cui ci si trova. Nel costruttore viene inizializzata una finestra di gioco “win” dell’altezza length e larghezza width, vengono poi inizializzati il livello e il timer. Per la gestione del timer si fa uso di 5 variabili: totalDuration che indica la durata in secondi che vogliamo dare ad ogni livello; frozenTime che contiene il tempo rimanente ogni qualvolta mettiamo in pausa il gioco e quindi fermiamo il timer; timerActive, un booleano che restituisce lo stato del timer (se attivo o in pausa); startTime, una variabile di tipo time\_t (della libreria ctime) che immagazzina il momento in cui inizia il conteggio del timer. Il timer quando il gioco è in pausa mostra il valore di frozenTime e ritorna 1 se il tempo ancora non è scaduto; invece, quando il timer è attivo calcola il tempo trascorso e lo mostra a schermo, ritorna poi 1 se ancora il tempo non è scaduto. Le funzioni StopTimer e StartTimer permettono di fermare e far ripartire il timer.

Il campo da gioco viene disegnato con il metodo drawBorder() e le varie informazioni vengono stampate a schermo con i metodi printHighestScore, printLevel e score.

Ciclo di Gioco

Il metodo run() della classe SnakeGame contiene il ciclo di gioco e gestisce tutto ciò che riguarda il gioco in sé per sé. Nel metodo, dunque, viene dichiarata una variabile booleana gameon, finchè il suo valore è true il gioco continuerà ad’andara. Nel ciclo di gioco vengono presi input dall’utente tramite la funzione getch() di curses e, se l’input è la barra spaziatrice viene messo il gioco in pausa tramite il metodo PauseGame(); se l’input invece è il carattere ‘q’ si abbandona la partita e viene aggiornata la classifica; altrimenti se l’input è una della 4 frecce viene mosso il serpente. Una volta preso l’input, se il gioco non è in pausa viene aggiornata la direzione verso cui si muove il serpente(se questa nuova è valida), viene fatto muovere il serpente tramite snake.Move() che restituisce false e fa dunque finire la partita se il serpente si morde la coda, viene aggiornato lo score e si controlla con il metodo CheckAppleCollision() se il serpente sia riuscito o meno a mangiare una mela, in caso positivo viene aggiornato lo score, infinte vengono stampate le informazione con i metodi della classe Board e viene refreshata la griglia di gioco, se il timer non è scaduto il ciclo ricomincia da capo.

Il serpente viene fatto muovere servendosi delle variabili TickCount e Ticks\_per\_move per fargli mantenere una velocità costante, e per fare in modo che, tenendo premuta una freccia il serpente non vada più veloce di come dovrebbe andare: il serpente viene quindi mosso ogni ticks\_per\_move cicli in modo da fargli mantenere una velocità costante, dunque tutte le azioni che modificano lo stato del serpente o della griglia di gioco avvengono solo quando il tickCount(che viene incrementato a ogni ciclo “a vuoto” e messo a 0 invece quando diventa uguale a ticks\_per\_move) è uguale ai Ticks\_per\_move. In questo modo è anche facile modificare la velocità a cui far andare il serpente tra un livello e l’altro: ciò viene fatto sottraendo a una costante il numero del livello, più alto sarà il livello più frequentemente verrà aggiornato il serpente e più veloce si muoverà sulla mappa.

Salvataggio, Punteggi e Classifica

La parte di salvataggio dei progressi viene svolta salvando su due file testuali diversi la classifica e il punteggio più alto ottenuto fino a quel momento, i file sono rispettivamente classifica.txt e punteggio.txt . Da questi file sono letti quindi i punteggi qualora il giocatore decidesse di leggere la classifica o giocando desideri visualizzare qual è il punteggio più alto mai registrato.

La classifica viene ordinata tramite un bubble sort e permette all’utente di vedere al termine di quale partita è stato registrato un certo punteggio ed anche il livello giocato fino ad un massimo di 100 partite. Se il codice non rileva punteggi salvati informa il giocatore tramite testo.

Quando il giocatore termina di consultare la classifica può tornare al menu principale premendo un qualsiasi tasto.

La funzione showClassifica() nel file menu.cpp si occupa di leggere i punteggi da file e ordinarli prima di renderli disponibili per la lettura da parte dell’utente dal menu di pausa.

La funzione Classifica() invece situata nel file SnakeGame.cpp salva i dati nel file classifica.txt

La funzione ScoreMultiplier() si occupa di attribuire al giocatore un Bonus se e solo se egli ha terminato la partita in corso, quindi timeout. la funzione non è chiamata quando il gioco termina per scelta diretta del giocatore, quindi premendo “q” durante la partita o durante la pausa.

Il numero di mele mangiate determina il punteggio ottenuto fino a quel punto che viene poi moltiplicato per il livello di difficoltà scelto, premiando quindi con un bonus maggiore il giocatore che avrà scelto un livello più arduo.

Livelli

Il sistema di livelli ha richiesto l’uso delle liste bidirezionali per poter essere realizzato, in questo modo il giocatore può, tramite il menù di pausa, scegliere di interrompere il livello giocato sino a quel momento per provarne uno più semplice o difficile.

Infatti c’è una stretta relazione tra ognuno dei 6 livelli disponibile e la velocità di movimento del serpente resa possibile dall’implementazione di un sistema basato sulle variabili tickCount e TICKS\_PER\_MOVE di cui si è occupato il mio collega e dalla manipolazione della funzione “napms” di ncurses per mettere in pausa l’esecuzione del programma per un numero scelto di millisecondi.

Una delle funzioni più importanti per gestire i livelli è getLivelloVicino(NodoLivello\* start, int targetNumero), essa prende come argomento start e targetNumero, relativamete il nodo inteso come difficoltà nel quale siamo e il livello sul quale ci vogliamo spostare per giocare, con tale funzione possiamo cercare il nuovo nodo partendo da quello già scelto risparmiando tempo e risorse non dovendo sempre partire dalla testa, questo grazie alle liste bidirezionali; se targetNumero è più grande di start ci muoviamo in avanti, se più piccolo indietro.

cambiaLivelloDurantePausa (int oldLevel, ListaLivelli& listaLivelli)

è la funzione che permette di attuare il cambio di livello dal menù di pausa settando le variabili Speed e Level sul nuovo livello, gestisce anche il reset del punteggio per evitare che il giocatore si appropri di un bonus che non gli spetta e controllo se il livello scelto esiste e se il livello è uguale a quello giocato fino a quel momento.

Menu

il menù è il sistema di controllo centrale del gioco, tramite il menù il giocatore ha subito davanti a se 3 scelte: cominciare una nuova partita, vedere la classifica con i punteggi ottenuti fino a quel momento o uscire dal gioco. Se sceglie di giocare gli verranno messe davanti le 6 difficoltà disponibili che incrementando gradualmente la velocità in ordine crescente riducono il tempo di reazione disponibile per muoversi e quindi fare punti.

È presente un sottomenù, quello di pausa, invocabile solo mentre si è in partita che consente al giocatore di cambiare livello di difficoltà perdendo però tutti i punti accumulati fino a quel momento, un'altra scelta disponibile è uscire dal gioco alla pressione di q, il punteggio verrà inserito in classifica senza ricevere però alcun bonus.