Progetto programmazione a. 2024/2025

Tommaso Lalicata - Carlo Cataldo Panariello - Luca Pinardi

1. Scelte implementative dei livelli, menù e timer:

Livelli:

Nei campi private della classe livelli c'è solo un puntatore alla testa della lista bidirezionale, nel public sono presenti il costruttore che inizializza la testa a nullptr, un metodo che restituisce il puntatore alla testa di lista completa dopo aver assegnato ad ogni val il valore corretto ed infine un metodo che ritorna il valore di un nodo passato come argomento.

Menù:

I campi private della classe menu sono un array di caratteri, per il nome del giocatore corrente, un intero che determina che stato del gioco stampare (menù principale, classifica, etc…), un intero per tenere conto della difficoltà del livello selezionato ed un array di booleani ausiliario che, nel menù di selezione del livello, permette di stampare nella finestra a destra quali livelli risultano non ancora giocati e nella finestra principale il numero corrispondente di grigio, qualora il livello sia già stato giocato nella stessa sessione dello stesso player.

Il costruttore inizializza tutti i campi. Il metodo colorsetup() salva diverse coppie di colori che verranno utilizzate per tutto il gioco e start_up() gestisce gli stati di gioco e chiama gli altri metodi grazie alla variabile state, è l'unica funzione da chiamare nel main per far partire il gioco. I metodi main_menu(), player_select(), level_select() sono quelli con cui il player interagisce inserendo input da tastiera (principalmente freccette, ENTER ed ESC) per iniziare una nuova partita, inserire il proprio nome e selezionare la difficoltà rispettivamente. Il metodo new_game() funge da scheletro per il gioco vero e proprio, implementato dagli altri colleghi. Se il giocatore preme il carattere "p" durante una partita allora viene chiamato il menù di pausa tramite il metodo pause() restituendo uno stato aggiornato qualora il giocatore decida di tornare al menù principale o a quello di selezione del livello. Se il giocatore vuole continuare la partita messa in pausa la funzione ritorna un intero (nel nostro caso il numero 573 per nessuna ragione specifica) che invece di far ritornare lo stato aggiornato, fa break dallo switch, continuando ad eseguire il while del gioco.

Timer:

I campi private della classe timer sono due interi, uno per i secondi che si vuole assegnare al timer e uno per contare i secondi che passano mentre il gioco è in pausa, due time_t rispettivamente per l'inizio del timer e per la pausa ed infine un booleano che mi dice se il timer è in pausa.

Il costruttore inizializza tutti i campi tranne il time_t pausa, che viene inizializzato dal metodo pause_timer(), da chiamare ogni volta che si vuole mettere in pausa. Il metodo resume_timer() invece aggiorna i secondi di pausa correttamente e va chiamato subito dopo la chiamata al menu di pausa. Il metodo time_left() ritorna in secondi il tempo rimanente, controlla se il timer è in pausa per ritornare valori attendibili in qualunque momento del gioco. Il metodo time_out() ritorna true se è finito il tempo; considerato che nella nostra implementazione un livello si "vince" per timeout, nel while di gioco vengono utilizzate come guardie sia il metodo negato time_out() su un'istanza di timer sia un booleano game che viene settato a false qualora avvenga una collisione con il corpo dello snake. Questa implementazione permette di differenziare un fine partita per time_out (vittoria) da un fine partita per collisione (non propriamente un game over, ma il punteggio fatto in quel livello non sarà salvato). Il metodo display() permette di stampare in una finestra in determinate coordinate il tempo rimanente. Per concludere, ispaused() è un metodo ausiliario utilizzato dagli altri colleghi.

2. Scelte implementative di snake, movimento e classifica:

Snake:

Ho deciso di implementare il corpo del serpente attraverso una lista con puntatori. Ogni parte del corpo ha un carattere per visualizzare il serpente ('o' per il corpo e '@' per la testa) e delle coordinate per identificare la sua posizione nella matrice. Il campo in cui si muove lo snake infatti è una matrice booleana in cui sono indicate come true le caselle in cui si trova il serpente. Inizialmente il costruttore della classe snake inizializza la matrice come vuota, cioè tutta falsa, e il serpente viene creato di default che guarda verso destra, con la testa al centro della matrice e il corpo dietro. Ovviamente il serpente è costruito attraverso una funzione che aggiunge elementi in testa alla lista ("plista crea_corpo" che mi restituisce la lista aggiornata). Il costruttore prende in input pure una finestra e la difficoltà del livello.

Movimento:

Il movimento è stato realizzato sfruttando la matrice booleana e le proprietà della lista. Infatti il serpente si muove rimuovendo un pezzo della coda e aggiungendone uno in testa e, per fare ciò, mi servo di una funzione "void dequeue" che rimuove l'ultimo pezzo della coda e setta false e "crea_corpo" ovvero la funzione usata per aggiungere in testa gli elementi. Inoltre siccome voglio che la testa venga contrassegnata sempre da '@'.Per muovere il serpente nelle quattro direzioni ho creato delle funzioni "void UP, DOWN, RIGHT e LEFT" che eseguono "dequeue" aggiungendo la testa in base alla direzione. Inoltre, siccome la collisione con i bordi non dà game over ma crea un effetto pac-man, aggiorno le nuove coordinate in modo da far spuntare il serpente dalla parte opposta. Alla fine con la funzione "void movements" (che ha in input due parametri, ovvero uno dell'azione appena inputtata e un altro per l'ultima azione eseguita) imposto che vengano presi gli input da tastiera delle quattro frecce e con uno switch faccio in modo che il serpente si muova nelle direzioni richieste. Ho messo dei vincoli, ad esempio il serpente non può cambiare immediatamente la propria direzione in modo opposto al verso in cui sta andando e ciò viene controllato da una funzione booleana chiamata "inversione" . Inoltre se vengono schiacciati altri tasti, questi non vengono letti e il serpente continua a muoversi nella sua direzione. In questo modo il serpente si muove da solo e il giocatore gli assegna solo la direzione, rimuovendo anche possibili buffer di memoria, in modo che il serpente sia sempre responsivo. Infine attraverso la funzione "double speed" effettuo una divisione tra una constante e la difficoltà e ottengo il tempo di aggiornamento dello schermo, cioè la velocità del serpente: più il numero è piccolo più il serpente andrà veloce dato che napms conta in millisecondi.

Infine ho due funzioni "pos (variabile struct con due interi) get_head" che mi restituisce la posizione della testa e "isoccupied" che mi restituisce la posizione occupata con true, presi input due coordinate.

Classifica:

Ho implementato la classifica come un array di nodi in cui ogni nodo è una struct che ha al suo interno una stringa (array di char), la quale conterrà il nome del giocatore e un intero che contiene il punteggio. Il costruttore della classe classifica inizializza l'array rendendolo vuoto. Ho creato delle funzioni void "bubble_sort" che mi ordina l'array in modo decrescente in base ai punteggi (inoltre se due persone hanno lo stesso punteggio vengono ordinate in modo alfabetico) e una funzione void ausiliaria "scambianodo" (prende in input due variabili nodo) che è fondamentale per ordinare l'array con il bubble. Inoltre ho creato una funzione "booleana isempty" che restituisce true se c'è almeno uno spazio vuoto e una funzione "void right_shift" che sposta gli elementi dell'array a destra a partire da una posizione presa in input. Le funzioni fondamentali sono "void leggi_file" (ha in input la finestra) che apre un file di testo in cui si visualizzano i suoi dati e vengono ordinati con il "bubble_sort", "void scrivi_file" che serve per aggiornare il file con la nuova classifica e "inserimento" che aggiunge o meno un player nella classifica. Questa funzione prende in input il punteggio e il nome del giocatore e ed effettua vari controlli prima di inserire o no il nuovo nodo:
se l'array è vuoto ("bool isempty") viene aggiunto il nuovo nodo e si ordina la lista;
se l'array è pieno ma il nome è già presente si aggiorna il punteggio se è maggiore del precedente;

se l'array è pieno e il nome è nuovo allora esegue un "right_shift" e inserisce il nuovo player se rientra in classifica, altrimenti non lo inserisce;

Infine uso una finzione "int stampa_file" (ha in input una finestra) in cui vengono richiamate "leggi_file" (che richiama "bubble_sort") e "scrivi_fie" e che stampa la classifica finché non viene premuto esc. A quel punto si restituisce 0. La classifica presenta un podio e poi una sorta di tabella in cui vengono mostrati gli altri giocatori con i relativi punteggi. Il titolo è stato creato attraverso un sito che trasforma le frasi in ascii.

3. Scelte implementative della griglia di gioco, oggetti, collisioni e punteggio:

Griglia di gioco:

La classe Grid si occupa del monitoraggio delle collisioni e della stampa della finestra di gioco.
Nel campo protected è salvata la matrice di gioco, che è una matrice in cui ogni cella ha tre campi: un booleano che indica se la cella è occupata, un carattere che indica cosa la occupa e un id che indica l'id del frutto che la occupa. Inoltre sono presenti anche diversi valori ausiliari come un booleano che indica se vi è un gameover, un intero per lo score del livello, un intero che serve ad assicurarsi che ogni frutto generato abbia un id diverso, due timer, uno che genera un oggetto ad intervalli semi-regolari ed uno che si assicura che quando un item di tipo chain viene rimosso o mangiato venga generato nuovamente.
Infine vi sono diversi valori ausiliari quali 3 interi per settare il tempo dei timer, un booleano che serve a contollare se un frutto del timer chainitem è già presente in campo, un intero per la difficoltà e un booleano che avvisa se al ciclo precedente il gioco era stato messo in pausa (serve per una piccola ottimizzazione)

Nel campo public invece vi è il costruttore, che inizializza la griglia di gioco, i vari campi e il tempo dei timer in base alla difficoltà; vi è poi la funzione portante della classe, ossia Updatemtx().
Questa funzione serve a controllare le collisioni con gli oggetti o con il corpo del serpente e i timer sia per la generazione che per la rimozione degli oggetti tramite chiamate alle altre funzioni, e verrà utilizzata dalla funzione UpdateGrid() che va invece a stampare a schermo il contenuto delle celle della matrice.
La funzione Collision() viene chiamata da Updatemtx() e serve a rimuovere un oggetto dalla griglia quando viene mangiato, oltre ad aumentare il punteggio.
Per fare ciò viene chiamata la funzione removeItem() che serve a rimuovere gli oggetti sia mangiati che expired.
Viceversa addItem() aggiunge un oggetto alla griglia di gioco, assicurandosi di non generarlo in un punto già occupato.
La funzione pausetimers() viene chiamata quando il gioco viene messo in pausa, controlla per sicurezza che il gioco effettivamente sia in pausa e di conseguenza blocca temporaneamente tutti i timer.
Le funzioni rimaste sono isendgame(), setendgame() e UpdateScore(), che servono rispettivamente a contollare il valore del campo endgame, a modificarlo , e ad aggiornare il valore dello score nella finestra di gioco.

Oggetti:

La classe Items si occupa della creazione, rimozione e gestione di tutti gli oggetti necessari dalla classe Grid.
Il campo protected contiene solo due variabili, ossia un intero per la difficoltà e un puntatore ad una lista itemlist. Questa a sua volta è formata da molteplici campi, ossia:
un campo pos che tiene conto di due coordinate (rispettivamente y ed x);
un intero che contiene l'id unico per ogni oggetto (grazie al counter della classe Grid);
un char che indica che tipo di frutto è l'oggetto, se Apple, Banana o Cherry;
un intero che indica il valore dell'oggetto
un booleano che indica se l'oggetto è un chainitem o no
un timer specifico per ogni oggetto, che identifica il tempo rimasto prima che il frutto "vada a male"
i due puntatori a next e prev che la rendono una lista bidirezionale

La scelta di implementare la classe tramite una lista è stata fatta in modo da dover istanziare solo una volta un oggetto della classe item, andando a lavorare sulla lista quando è necessario creare, rimuovere o accedere ai campi degli oggetti solo in base all'identificatore.

Nella classe public sono contenute infatti tutte le funzioni necessarie ad eseguire le operazioni per la gestione degli oggetti.

La prima funzione è ovviamente il costruttore che inizializza il campo difficulty, mentre la testa viene settata di default a nullptr;
A seguire c'è la funzione newitem(), utilizzata per generare un oggetto, inizializzare tutti i suoi campi ed aggiungerlo in testa (inizializzando anche la lista nel caso sia vuota)
La funzione removeitem() rimuove invece un oggetto dalla lista deallocando la memoria e ricollegando in maniera adeguata i nodi senza creare dangling pointers.
La funzione changepos() viene chiamata dalla funzione Grid::addItem() qualora l'oggetto generato da newitem() sia stato generato in un punto già occupato dal serpente o da un altro oggetto.
Le funzioni pausealltimers() e resumealltimers() si occupano di fermare o far ricominciare i timer degli oggetti qualora il gioco venga messo in pausa ricollegandosi alle funzioni della classe timer.
La funzione expiredtimers() invece restituisce l'id del primo oggetto che ha esaurito il tempo oppure -1 se non ne trova nessuno. Viene richiamata da Grid::Updatemtx in un ciclo che va avanti finchè tutti gli oggetti che sono expired vengono rimossi.
La funzione deleteallitems() viene chiamata quando il gioco è finito e rimuove tutti gli oggetti dalla lista deallocando la memoria e puntando head a nullptr;
Infine le funzioni getitem(), getpoints(), getposition(), gettimer e gettype() servono ad accedere ai campi degli oggetti.

Collisioni:

Vi sono due tipi di collisioni possibili all'interno del gioco: le collisioni con le pareti e quelle con una cella già occupata. Queste ultime, che comprendono collisioni con oggetti o con il serpente stesso, vengono gestite esclusivamente dalla funzione Updatemtx() della classe Grid, mentre quelle con i bordi della griglia sono gestite dalla classe movimento.

Punteggio:

Il punteggio viene controllato dalla funzione Updatemtx() della classe Grid e dalla classe menu.
Infatti tramite le collisioni con oggetti il punteggio viene incrementato in base al valore dell'oggetto mangiato, ossia rispettivamente 100, 150 e 200 se la collisione avviene con una Mela (A), Banana (B) o Ciliegia (C).
Alla fine di ogni livello il punteggio ottenuto viene aggiunto allo score totale del giocatore.