"THE GAME", un progetto di: Alex Rossi mat:0001089916, Alessandro D'Ambrosio mat: 0001099122, Luca Rimondi mat: 0001078751;
Alma Mater Studiorum, University of Bologna, Italy.

-Informazioni generali:
"THE GAME" è un videogioco platform 2D per terminale, sviluppato in C++, dove si vestono i panni di un eroe che deve sopravvivere giorno e notte ad attacchi di nemici con l'obiettivo di accumulare il punteggio più alto possibile.

-Informazioni riguardanti scelte progettuali:
File main.cpp:
In questo file si trovano l'inizializzazione degli oggetti più importante del gioco come le 10 mappe e il menu di base.
Inoltre si trovano alcune importanti funzioni che servono a settare valori di base del gioco, impedire al giocatore di causare comportamenti anomali durante l'esecuzione del programma (la funzione initialize() ad esempio nasconde il cursore del mouse e non permette di scrivere sul terminale con la tastiera in-game) e la visualizzazione della schermata di introduzione (data dalla funzione pregame()). Infine sono presenti le funzionalità di choice(), che permette il corretto funzionamento del Menu iniziale, la quale permette di scegliere tra: iniziare una nuova partita (opzione 1), decidere di riprendere a giocare da un game-file preesistente (opzione 2) o uscire direttamente dal gioco (opzione 3).

File gameEngine.cpp/gameEngine.hpp:
In questo file si trovano le funzioni che permettono di gestire la logica e l'avanzamento di livello dell'effettivo gioco.
Per quanto riguarda la generazione dei livelli map_randomizer() si occupa della generazione casuale di un seed, attraverso la funzione srand() se il valore next_seed è 'false', che verrà utilizzato per generare la mappa associata da cui inizierà il gioco, altrimenti se next_seed è 'true' allora viene restituito il seed della mappa successiva.
La funzione map_generator() si occupa di restituire il puntatore alla mappa a cui fa riferimento il seed immesso.
La funzione più importante del file è game_flow(), in essa vi è l'effettivo svolgimento della partita in-game, l'avanzamento di livello e l'accesso al market di gioco.
Generalmente il suo funzionamento è il seguente: in essa vengono istanziati sia i nemici che il player (di quest'ultimi però se ne seleziona una tipologia casualmente così da essere inserita nel livello corrente), i quali vengono inseriti in un ciclo che continua fino a quando non si soddisfa una di queste tre condizioni:
1. le vite del player scendono al valore 0 (il giocatore 'muore' ed è quindi 'forzato' a terminare il gioco attraverso death_screen());
2. il player ha raggiunto il punto massimo, o il minimo, raggiungibile nel livello ed è quindi pronto a passare al livello successivo (in questo caso si va ad aggiornare seed e puntatore della mappa, infine viene richiamata ricorsivamente game_flow()), nel caso del minimo viene richiamato game_flow() con i dati per tornare al livello precedente a meno che l'attuale livello non sia la prima stanza generata.
L'avanzamento di livello viene gestito come la struttura dati dello STACK, infatti si mettono in coda di volta in volta livelli, utilizzando sempre l'ultimo livello inserito. In caso di sconfitta o uscita dal gioco lo STACK si svuota completamente;

3. si accede al menu di pausa dove si può decidere di riprendere la partita dal punto in cui si è interrotta, accedere al market del gioco per acquistare potenziamenti o salvare la partita e uscire dal gioco.
Prima di entrare nel ciclo però viene sempre fatto un controllo per vedere se bisogna settare dei valori seguendo un file preesistente salvato o se invece impostare valori di base semi-casuali.

File market.cpp/market.hpp:
In questo file troviamo funzioni che regolano la logica e controllano se è possibile effettuare un acquisto o meno (check_Currency()).
Le funzioni buy_Health(), buy_Jumpboost(), buy_MagicPotion(), buy_Artifact() permettono l'acquisto dei rispettivi potenziamenti nel caso si soddisfano le condizioni di check_Currency(), le quali vanno a modificare i parametri di salto, vita e SCORE del player.
I potenziamenti nel market vengono generati semi-casualmente: ogni volta che si accede al negozio si trova la possibilità di acquistare due potenziamenti (su quattro totali) che vengono di volta in volta mostrati a rotazione (la possibilità di comprare ogni volta potenziamenti differenti viene gestita in game_flow() attraverso la funzione srand()).

File box.cpp/box.hpp:
In questo file si trovano tutte le funzioni che vengono utilizzate per disegnare a schermo ciò che si può vedere nel gioco.
In esso è centrale la caratteristica di ereditarietà delle classi: la classe BOX è la classe padre necessaria per disegnare sul terminale il quadrato generale sia di gioco che di menu; la classe MENU è figlia di BOX ed aggiunge la funzione choice() che permette la possibilità di scelta tra le varie voci di menu, il suo funzionamento è il seguente: viene inizializzato un ciclo che termina solo quando il giocatore ha effettuato una scelta tra 3 opzioni disponibili e viene restituito un valore intero, da 1 a 3, in base all'opzione selezionata.
MARKET è classe figlia di MENU ed aggiunge attraverso draw_market() abbellimenti estetici al market.
MAP è classe figlia di BOX ed è responsabile di disegnare la mappa in base ai parametri inseriti nell'istanziamento della classe: ladder_on (implica la presenza di scale nella mappa), holes_on (implica la presenza di buchi nella mappa), two_holes (sono presenti due buchi nella mappa), two_ladders (sono presenti due scale nella mappa), cloud (vengono disegnate delle nuvole nella mappa), day (se 'true' viene disegnato un sole per indicare il giorno, altrimenti viene disegnata la luna per indicare la notte), x1_hole_start (indica dove inizia il primo buco della mappa), x1_hole_finish (indica dove finisce il primo buco della mappa), x2_hole_start (indica dove inizia il secondo buco della mappa), x2_hole_finish (indica dove inizia il secondo buco della mappa).

File DataManager.cpp/.hpp:
Il file contiene al suo interno le funzioni atte alla gestione dei dati di salvataggio progressi.
La principale, save_data(), si occupa di salvare i dati del giocatore e dei nemici.
Preso il puntatore all'oggetto di classe player corrente (P), il "seed" della mappa (seed), lo score del player(game_scr), il nome del file in cui salvare i dati (fileName) ed un puntatore alla superclasse del nemico che lo salva indipendentemente dalla sua tipologia), viene controllato se il file passato per parametro è preesistente. In tal caso il file viene scritto da zero, così accade, altrimenti vengono riscritti solo i dati del giocatore, copiati quelli dei nemici, e scritti/modificati quelli del nemico corrente (se si entra in una nuova stanza, se si

salva con un nemico già visitato presente, la linea sarà sovrascritta). La funzione che si occupa di salvare i dati dei nemici è newEnemyDataSave(), seguito di enemyDataSave(), lasciata come commento.
obtain_data() si occupa di prendere uno specifico dato dal file di salvataggio. Prendendo come parametri in input due valori interi, il primo necessario per entrambe le ricerche, ed il secondo solo per quella di dati appartenenti ai nemici e non al giocatore, la funzione controlla se il secondo dato è stato passato come parametro o se corrisponde al valore di default: in questo caso, solo i dati del player vengono selezionati sulla base del primo valore passato, altrimenti viene considerato il secondo per distinguere il livello nel quale si trova il nemico interessato ed il primo per capire quale dato recuperare.
ChangeData_basic() serve a modificare uno specifico valore corrispondente ai dati del giocatore. Non prende in considerazione i dati dei nemici.
CopyFile() si occupa di creare una copia del file in input, creando un file con nome corrispondente al secondo parametro passato.

File player.cpp/player.hpp:
Questi file racchiudono il funzionamento della classe player. I metodi contenuti servono a stampare il player a schermo, muoverlo, generare il proiettile e ricevere danno.
I metodi mvup(), mvleft(), mvright() e mvdown() servono tutti a stampare il giocatore più in alto, a destra, a sinistra o in basso. mvleft() e mvright() inoltre controllano se sono presenti scale nella posizione in cui si sta cercando di stampare il giocatore e in caso positivo, lo fanno salire attraverso la funzione stairsup().
mvdown() controlla se ci sono scale, ma a differenza degli altri due metodi, permette al player di scendere attraverso strairsdown().
jump() è crea l'animazione del salto e comunica direttamente con shoot(). Questo metodo genera un proiettile ('o'), che attraversa la mappa fino a che non arriva al bordo, oppure fino a che non arriva a colpire un ‘bulletterrain’, controllato dall'omonimo
metodo. bulletterrain() ritorna un booleano true solo se il carattere preso in input è considerabile terreno.
Per il proiettile, il terreno sono tutti i caratteri associabili ai nemici, oltre a quelli che il giocatore considererebbe terreno (#, -, |, {}, []).
jump() e shoot() comunicano tra loro per poter assicurare al giocatore di poter saltare mentre il proiettile viene disegnato.
leftright() è il metodo che controlla l'input del giocatore e richiama i metodi di movimento appropriati, inoltre richiama il metodo lifeshow() che, oltre a mostrare le vite rimanenti a schermo, controlla se il giocatore è caduto nel vuoto o se è stato colpito dal proiettile; in caso positivo, gli sottrae una vita e lo rigenera nel punto iniziale (se caduto) o sul posto dove è stato danneggiato (se colpito da un proiettile).
Il metodo move(), oltre che richiamare lifeshow, richiama il metodo gravity e ritorna l'input del giocatore.
Il metodo gravity(), comunica con il metodo isterrain(), che controlla se il carattere al di sotto del player sia considerabile terreno o meno. In caso negativo, utilizza il metodo mvdown() fino a che isterrain() non ritorna true, in questo modo si assicura che il giocatore sia sempre sottoposto alla gravità.
Il metodo display() stampa il personaggio, mentre il metodo playeroutput(), in base all'input che riceve, ritorna il valore di una variabile, utile per per operazioni di salvataggio o di debug.

File basicenemy.cpp/basicenemy.hpp:
In questi file si trovano la classe basicenemy e la sua sottoclasse jumpingenemy.
La classe padre è responsabile per la creazione dei nemici base del gioco e condivide alcuni metodi con la classe player: mvdown(), mvright(), mvleft(), display(), shoot(), gravity(), isterrain() e enemyoutput(), quest'ultimo funziona in modo similare a playeroutput().
playerfinder() controlla fino a nove spazi di distanza se è presente il giocatore: rilascia 0 se è a sinistra, 1 se è a destra e 2 se non è stato trovato.
Il metodo takedamage() è simile a lifeshow(), ma non mostra la vita a schermo: quando riceve danno il nemico non torna dove è stato generato e, se cade nel vuoto, viene considerato morto istantaneamente.
Il metodo behaviour() contiene l'IA del nemico. Come prima cosa controlla se sta venendo disegnato un proiettile e, in caso positivo, continua a disegnarlo; successivamente, richiama playerfinder() che, se non ritorna 2 in output, allora c'è una probabilità del 50% che spari.
In caso ritorni false c'è una possibilità su cinque che decida di muoversi a destra e una su cinque che si muova a sinistra.
Inoltre jumpingenemy, controlla se è su un bordo richiamando il metodo isterrain() e in caso positivo, per evitare di cadere, si muove nella direzione opposta.
La sottoclasse di basicenemy, jumpingenemy, contiene anche le funzioni mvup() e jump(), similari a quelle contenute nella classe player.
Inoltre modifica per ereditarietà il metodo playerfinder(), estendendolo fino a quindici caratteri di distanza.
bulletfinder() funziona in modo simile a playerfinder() ma controlla la presenza di un proiettile a tre spazi di distanza.
Anche behaviour() viene notevolmente cambiato tramite ereditarietà: una nuova variabile di questa sottoclasse è difficulty, che divide alcuni valori nel metodo behaviour per rendere più o meno probabile che certe azioni (come la decisione di sparare) possano accadere; un'caratteristica di behaviour è che richiama bulletfinder() per trovare proiettili a tre caratteri di distanza e in base a una data probabilità, decide di saltarli. Il salto può però anche essere controproducente per il jumpingenemy poiché il giocatore può indurre quest'ultimo a saltare all'interno di buchi nel terreno, uccidendolo istantaneamente.