RELAZIONE PROGETTO PROGRAMMAZIONE

In questo progetto è stato usato il linguaggio di programmazione C++, e in particolare tutti i concetti fondamentali appresi durante il corso di programmazione oltre che la libreria ncurses per realizzare l'interfaccia a linea di comando. Il lavoro è stato gestito usando un approccio misto, ossia inizialmente abbiamo sviluppato "lo scheletro" del gioco per conto nostro a seguito di aver ragionato tutti insiemi sui dettagli implementativi, in seguito per realizzare tutto il resto abbiamo usato Google Meet e una estensione dell'IDE Visual studio Code, live share, per lavorare simultaneamente sullo stesso codice. Per salvare i progressi fatti nella realizzazione del progetto abbiamo usato github al fine di avere tutto il codice insieme in un'unica repository privata. Di seguito verranno presentate tutte le funzionalità e i meccanismi implementati da uno o più membri del gruppo.

Introduzione: Il gioco "1 vs (all)phabet" è un gioco platform in cui il giocatore controlla un personaggio che deve raccogliere le monete "@" per superare i vari livelli; nel frattempo, il giocatore deve evitare i mostri che cercano di impedire il suo progresso. Sia il player che i mostri sono implementati mediante l'utilizzo di classi, in particolare, essendo che entrambi presentano tratti comuni, è stato scelto di implementarli come sottoclassi della classe Character, della quale presenteremo e discuteremo in seguito il prototipo dei metodi e gli attributi.

Inizio: Il gioco, appena eseguito, mostrerà a video una prima schermata iniziale dove sarà possibile scegliere se si vuole effettuare il login a un account precedentemente registrato, se crearne uno nuovo, o se uscire dal gioco.

Nel caso si ricada in una fra le prime due scelte verranno richiamati rispettivamente da un oggetto di classe Player:

- login(username, password), che si occupa di accedere alla cartella archivio, che a sua volta contiene la cartella con i dati e i livelli dei vari giocatori che hanno effettuato precedentemente l'accesso al gioco. A questo punto controlla che l'utente esista e/o la password fornita da quest'ultimo combaci con quella dell'account alla quale si vuole effettuare l'accesso; in caso una delle due condizioni non sia soddisfatta verrà mostrato a video il conseguente messaggio di errore, altrimenti se non si verificano errori sarà possibile effettuare l'accesso a quell'account e avere quindi le risorse necessarie per continuare la partita dal punto in cui si aveva lasciato esattamente con i medesimi potenziamenti, punteggio e monete.
- signin(username, password), che si occupa di creare un nuovo account, cioè di accedere alla cartella "Archivio" e creare la cartella col nome dell'utente (se questa già non esiste) e avvia il gioco partendo dal primo livello; se l'account che si desidera creare esiste già verrà mostrato a video un errore e ci sarà la possibilità di crearne uno valido.

In ogni caso, successivamente sarà generata una mappa random oppure verrà caricata da file e il player ricomincerà da livello nella quale si trovava prima di chiudere il gioco, oppure nel caso scelga "Sign In" partirà dal primo livello.

Avendo prima nominato la classe Player, è doveroso fornire dettagli riguardo quest'ultima, questa classe è una sottoclasse della classe Character, ossia la classe che gestisce tutti i personaggi chiave del gioco (i mostri e il player). Di seguito si riportano gli attributi e i prototipi dei metodi della classe Player:

- Attributi:
    - nick[20] (char[ ]), ossia il nome del player;
    - psw[20] (char[ ]), cioè la password scelta dall'utente;
    - money (int), che rappresenta il numero di monete del giocatore;
    - current_level (int), il livello corrente in cui si trova il player;

    - score (int), il punteggio accumulato del player progredendo nel gioco.

- Prototipo Metodi
    - Due costruttori, di cui uno di default e uno che prende in input username, password e look e li assegna ai rispettivi attributi;
    - int fight(int m_hp, int m_atk, int m_def)…
    - char* getNick(), che ritorna l'attributo nick[ ];
    - int getMoney(), che ritorna l'attributo money;
    - void pay(int p), che in caso di acquisti, decurta il parametro p dal valore dell'attributo money;
    - void takeMoney(int value), che aggiunge il value al valore attuale dell'attributo money;
    - arnd check_oround(Map &m)…
    - int choice_menu(), che si occupa di creare la schermata iniziale del gioco, la quale mostra in primo piano il nome e di seguito in colonna le 3 opzioni descritte al punto "Inizio", ossia "Login, SignIn e Exit", ritornando a loro volta un apposito valore in base alla scelta presa (rispettivamente 0, 1, 2)
    - void getCredentials(char username[], char password[]), che mediante una semplice interfaccia grafica chiede in input le credenziali username e password;
    - void saveStats(), dedita al salvataggio delle statistiche del player sul file di testo apposito nell'archivio;
    - void setCurrentLevel(int l), che setta l'attributo current_level con il parametro l;
    - int getCurrentLevel(), che ritorna l'attributo current_level;
    - void incScore(int score), che incrementa l'attributo score con l'omonimo valore passato tramite parametro;
    - int getScore(), che ritorna l'attributo score;
    - void setScore(int score), che setta l'attributo score con il parametro dato;

Mappa: Gli attributi della classe sono l'altezza e la larghezza (height e width) della matrice di gioco, la matrice di caratteri matrix e il numero di monete (coins).
Tutti i metodi riguardanti la mappa e la rilevazione di quegli elementi che si trovano su di essa, vengono implementati nella classe Map, suddivisa nei file Map.hpp e Map.cpp.
Questa classe è caratterizzata dalla presenza di due costruttori.
Il primo costruttore, prende in input il livello del gioco, le dimensioni della mappa e crea una nuova mappa. Questo costruttore viene chiamato quando avviene la creazione di un nuovo livello. Il costruttore si occupa anche della generazione delle monete che si troveranno sparse casualmente nella mappa, numero di monete = 5 + (livello/2), in questo modo il numero di monete da raccogliere aumenta di 1 ogni due livelli, aumentando gradualmente la difficoltà di gioco (il giocatore deve raccogliere tutte le monete per accedere al livello successivo). Prima della generazione delle monete, vengono generati i bordi della mappa, poi viene la generazione delle mura interne. Questo elemento della creazione della mappa è fondamentale poiché insieme alle monete in posizioni sempre differenti, è ciò che rende ogni livello diverso ed unico rispetto agli altri. Il numero delle mura interne = larghezza + altezza della mappa di gioco. Il primo muro di una sequenza di mura è generato in una posizione casuale, gli altri muri vengono generati in posizioni adiacenti, in direzioni casuali, questo processo continua fino a quando si è raggiunta la lunghezza della sequenza o quando la posizione scelta per il muro successivo è troppo vicina ai bordi o è già occupata da un altro muro. Una volta generate tutte le mura, e tutte le monete, vengono posizionati sulla mappa i caratteri che formano i portali d'accesso al livello precedente e successivo.
Il secondo costruttore invece genera la mappa leggendo da file il metodo readMap.
La stringa contenente il nome del file è l'unico parametro che viene passato al costruttore

Un altro metodo di fondamentale importanza è il metodo writeMap, che genera un file di testo contenente la mappa nell'esatto stato in cui la si ha lasciata prima di uscire dal gioco.
In questo modo in caso di spegnimento del pc, tutti i progressi del giocatore vengono salvati su file. Ci sono il getter e il setter per ottenere ed impostare il numero di monete della mappa. Di notevole importanza sono una serie di metodi che ritornano valori booleani relativi alla presenza di certi elementi in una determinata posizione:

- isInside(int x,int y) → metodo che controlla la presenza se la posizione x,y è all'interno delle mura
- isEmpty(int x, int y) →ritorna true se il carattere alla posizione x,y è una moneta, è vuoto o se è un proiettile (true anche per proiettili e monete poiché non sono soggetti alle collisioni dei proiettili sparati dal protagonista e dai nemici)
- isMoney(int x, int y) → ritorna true se il carattere alla posizione x,y è una moneta
- isMonster(int x, int y) → ritorna true se alla posizione x,y c'è un mostro (si verifica che Map.matrix[y][x] sia una lettera maiuscola)
- isTurret(int x, int y) → ritorna true se alla posizione x,y c'è una torretta (si verifica che Map.matrix[y][x] sia una lettera minuscola)

L'ultimo metodo di qui è importante parlare è il metodo clean, esso viene chiamato al momento dell'uscita dal gioco, serve per rimuovere dalla mappa i proiettili prima del salvataggio su file.

Personaggi: 1 vs (all)phabet presenta 2 tipologie di personaggi: player e mostri. Questi sono stati implementati a partire dalla superclasse Character, la quale include attributi e metodi fondamentali ad ogni personaggio per la stampa sulla mappa, il movimento, l'attacco, e l'interazione con gli oggetti presenti nel gioco.
Gli attributi della classe Character sono i seguenti:

x (int)	Coordinata x
y (int)	Coordinata y
hp (double)	Punti vita
atk (int)	Valore di attacco
def (int)	Valore di difesa
look (char)	Carattere con il quale il personaggio viene visualizzato sulla mappa

I punti vita vengono decrementati a seguito di una collisione con un proiettile, oppure in caso di contatto con un personaggio avversario. Nei casi appena citati la quantità di punti vita persi dipende dal valore di difesa (def) del personaggio che subisce il danno, e dal valore di attacco (atk) dell'avversario.

Attacco, difesa e punti vita possono essere incrementati per il player attraverso lo shop, mentre per i mostri aumentano gradualmente all'aumentare del livello.
Il giocatore può accedere allo shop in qualsiasi momento premendo "1" sulla tastiera.

I mostri all'inizio di ogni partita vengono distribuiti sulla mappa in modo casuale, e possono essere di due tipologie: mostri comuni o torrette.

I mostri comuni si muovono in direzione del player, tuttavia nel metodo move, che definisce il loro movimento, non vengono tenuti in considerazione eventuali muri che possono trovarsi tra il giocatore e il mostro.
Possono infliggere danno al player sparando dei proiettili oppure entrando in contatto con questo.
Le torrette non si muovono, e provocano danno al player solamente sparando.

La sottoclasse Monster presenta quindi un attributo di tipo booleano “tur”, con cui vengono distinti i mostri comuni dalle torrette (se il mostro è una torretta, tur = 1).
I mostri comuni sono identificati con caratteri da ‘A’ a ‘Z’, mentre le torrette da ‘a’ a ‘z’; queste lettere rappresentano la percentuale di vita dei mostri, dunque quando hanno piena vità saranno rappresentati da una ‘A’ (‘a’ per la torretta) e man mano che vengono colpiti o entrano in contatto con il player diventano tutte le lettere intermedie fino ad arrivare alla ‘Z’ (o ‘z’); quando vengono uccisi verranno cancellati dalla mappa e il giocatore acquisirà punteggio e monete come ricompensa.

1 vs (all)phabet non prevede un limite per il numero di mostri che possono essere presenti nella stessa mappa, difatti è stata scelta una struttura di tipo lista per la gestione delle diverse istanze di Monster.
La struct “mlist” definisce il tipo degli elementi da cui è composta la lista di mostri.
Quest’ultima e le funzioni utili alla gestione della lista sono dichiarate e definite nei file Monster.hpp e Monster.cpp , a seguito del codice relativo alla classe Monster.

Tutti i personaggi sono in grado di sparare dei proiettili per decrementare i punti vita dell’avversario.
L’oggetto proiettile è stato implementato nella classe “Bullet” e, così come per i mostri, è stata utilizzata una struttura di tipo lista per la gestione delle istanze.
La struct “blist” definisce il tipo degli elementi da cui è composta la lista di proiettili. Quest’ultima e le funzioni utili alla gestione della lista sono dichiarate e definite nei file Bullet.hpp e Bullet.cpp , a seguito del codice relativo alla classe.
Ogni volta che un personaggio spara, viene invocato il metodo “fire” di Character, che crea una nuova istanza di tipo Bullet e la inserisce all’interno della lista proiettili.
Così come per il danno da contatto, il danno provocato da un proiettile è calcolato in base al valore di attacco del personaggio che spara e al valore di difesa del personaggio colpito.
Il giocatore può sparare nelle quattro direzioni con i tasti ‘w’, ‘a’, ‘s’, ‘d’, mentre i mostri sparano in direzione del player quando si trovano allineati con questo.

Al fine di poter riconoscere monete o avversari che si trovano nelle quattro posizioni intorno al giocatore, è stato implementato il metodo “check_around” nella classe Player.
Tale metodo viene invocato costantemente nel game loop e restituisce una variabile di tipo struct, all’interno della quale sono contenuti quattro interi, ognuno dei quali si riferisce a una delle celle intorno a player, e assume un valore diverso a seconda dell’oggetto che si trova in quella posizione (0 = cella vuota, 1 = mostro, 2 = moneta).

Salvataggio su File: Tutte le caratteristiche del Player, al momento della creazione dell’account e al momento dell’uscita dal gioco vengono salvate nella cartella “Archivio”; in particolare al suo interno verranno create le cartelle con il nome dei Player, e al suo interno troveremo i file che conservano i dettagli delle mappe dei vari livelli del gioco (nei file denominati “Level*.txt” dove al posto di “*” si sostituisce il numero dell’i-esimo livello) e il backup dei dati del player, quali la password, le sue statistiche di attacco, difesa e vita (hp), il punteggio e monete. Inoltre il player potrà chiudere il gioco e riaprirlo o tornare indietro nei livelli e trovare TUTTO ESATTAMENTE COME LO HA

LASCIATO (monete, mostri, muri, etc.) incluse statistiche (sia del player che dei mostri), potenziamenti, punteggio e monete.

Shop: In ogni momento che si desidera, durante il gioco, sarà possibile per il player premere il tasto ‘1’ per essere portato sulla schermata dello shop, nella quale potrà spendere le monete ottenute superando livelli e ammazzando i mostri; in particolare, il codice che gestisce grafica e funzionalità dello shop si trova in una funzione nel main:

- void shop(Player &p), la quale prende come parametro un oggetto di tipo Player, che rappresenta il giocatore che invoca la funzione; se questo deciderà di potenziarsi vedrà decurtare le monete dal suo saldo e acquisirà i potenziamenti richiesti.

Gestione della partita: la gestione della partita sul singolo livello è gestita dalla classe Game. Il primo metodo di cui bisogna parlare è il costruttore:

- void Game(char filePath[], int atk, int def, int livello), qui viene creato il livello di gioco, viene subito fatto un controllo su filePath, per verificare che il livello non esista già, se esite avviene la lettura da file del livello e la vita dei mostri viene calcolata in base alla lettera da cui sono rappresentati. Il passaggio al costruttore dei valori di attacco e difesa del protagonista e del livello permettono di rendere la difficoltà del gioco sempre proporzionata e crescente all'aumentare dei livelli.
- int run(Player &p), in questo metodo risiede il game loop, che viene temporizzato tramite una serie di vari contatori che quando raggiungono un preciso valore permettono il verificarsi delle varie azioni del gioco (movimento mostri, spari, input da tastiera, danni da contatto). Da questo metodo vengono chiamati tutti quei metodi che eseguono l'aggiornamento della matrice della classe Map e tutti quei metodi che stampano a video gli elementi del gioco dopo gli aggiornamenti . In questo loop avviene anche il controllo delle liste dinamiche: quando un proiettile ha una collisione, esso viene rimosso dalla lista, quando c'è uno sparo viene inserito nella lista. La medesima cosa avviene per i mostri, quando la vita di un mostro scende sotto 0, esso viene tolto dalla lista dei mostri. Alla fine del metodo ci sono una serie di controlli sulle condizioni del gioco che stabiliscono l'esito del livello. Esistono 7 esiti diversi:
  exit, in_game, win, lose, go_to_next, go_to_prev, go_to_shot.
- I metodi nominati qui sopra sono metodi privati della classe, ai quali si aggiungono i metodi che stampano a video le statistiche di gioco, i comandi e la grafica del game over
- I parametri della classe sono il numero di mostri(int), il numero di torrette(int), la mappa (Map) e la lista dei mostri (in cui sono contenute anche le torrette).

Dinamiche di gioco: il cuore del gioco, il punto in cui risiede la logica principale che consente al gioco di funzionare è il main. Al suo interno infatti si inizia creando un oggetto di tipo Player, il quale dopo essere passato attraverso login o signIn inizierà a prendere forma; se il player effettua login (quindi aveva dei game precedenti), verrà creata la bi-lista dei livelli a partire dai file, altrimenti verrà creata man mano che il player avanza negli stati del gioco. Qui, in base ai valori di ritorno del Game.run() si stabiliscono i vari stati del gioco, ossia se il player desidera andare allo shop, se muore, se vince e deve andare al livelli successivo, se desidera tornare al precedente, etc…
Se il player perde verranno cancellati tutti i livelli e si ripartirà da un nuovo livello in cui i mostri avranno dei valori atk/def proporzionati ai potenziamenti del player. Prima di uscire dal gioco questo effettua un salvataggio su file di testo degli attributi del player e dei livelli affrontati tenendo conto del livello dalla quale è uscito.

Membri:

Fiorellino Andrea, matricola: 0001089150, email: andrea.fiorellino@studio.unibo.it
Po Leonardo, matricola: 0001069156, email: leonardo.po@studio.unibo.it
Silvestri Luca, matricola: 0001080369, email: luca.silvestri9@studio.unibo.it