PROGETTO – ESAME DI PROGRAMMAZIONE – A.A. 2022-2023

nWarrior Game

Scelte progettuali per lo sviluppo di un gioco platform 2D sviluppato in C++ utilizzando la libreria ncurses

Partecipanti:
Mat. n° 0001078029 - Leonardo Pinna

Il gioco nWarrior Game

Il gioco nWarrior Game è un gioco platform 2D senza una narrazione sottostante sviluppato in C++ utilizzando la libreria *ncurses*, e prevede un giocatore singolo che affronta livelli di gioco infiniti di difficoltà sempre crescente: in ogni livello sono presenti un numero di nemici in funzione del livello raggiunto nella partita e dalle statistiche del giocatore, una disposizione di elementi distruttibili (muri) che per ogni livello è scelta da un set di livelli predefiniti. Il giocatore e i nemici possono sparare nella direzione in cui sono rivolti. Solo il giocatore può distruggere i muri, e questo garantisce sempre che il giocatore non resti bloccato in un livello. È presente un sistema di punteggio e, in caso di nuovo record, il punteggio massimo raggiunto è salvato dal gioco.

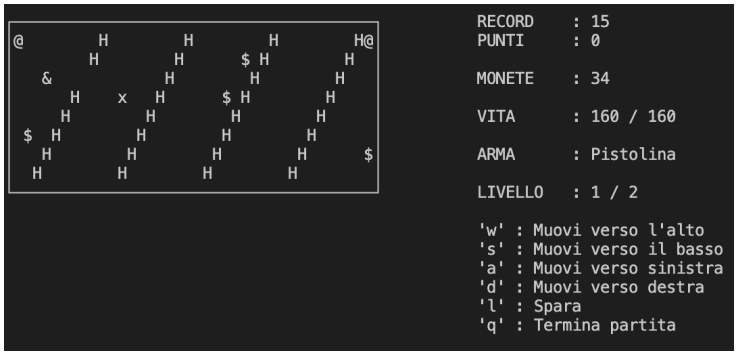

Figura 1: Il gioco nWarrior Game

Sono presenti 2 “porte” in ogni livello: una porta a sinistra permette di ritornare al livello precedente, con muri e nemici non distrutti, una porta a destra permettere di passare al livello successivo, che se non ancora raggiunto viene generato, oppure se già raggiunto mantiene lo stato di quando era stato lasciato in precedenza.

In ogni livello sono presenti un numero di elementi bonus relativi a vita extra e soldi, che possono essere ottenuti dal giocatore entrando in collisione con essi.

È presente un negozio, dove poter acquistare diverse armi usando la valuta di gioco, e dove è possibile acquistare vita massima extra per il giocatore. Ogni arma ha un valore di danno ed una portata del proiettile differenti.

Infine, è possibile salvare i dati di gioco in un file e caricare i dati di gioco dal file di salvataggio.

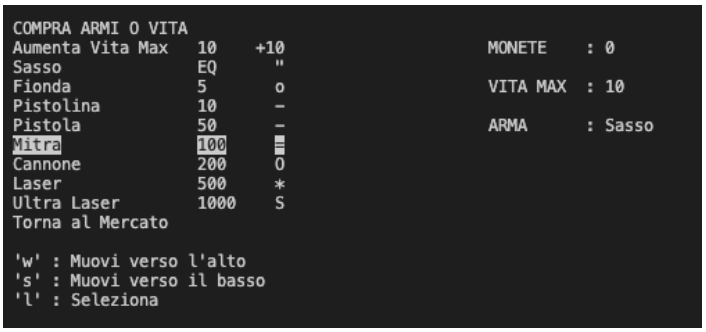

Figura 2: Negozio di armi e vita

Il flusso del gioco

Classe Game

Il gioco è lanciato dal file "*main.cpp*", che crea un oggetto di tipo ***Game*** e esegue il metodo *Game::run()*.
La classe *Game* è inizializzata e sono inizializzati i menù di gioco, il controller del giocatore (vedi la sezione *PlayerManager*) e un puntatore ad un oggetto *WINDOW* della libreria *ncurses*.

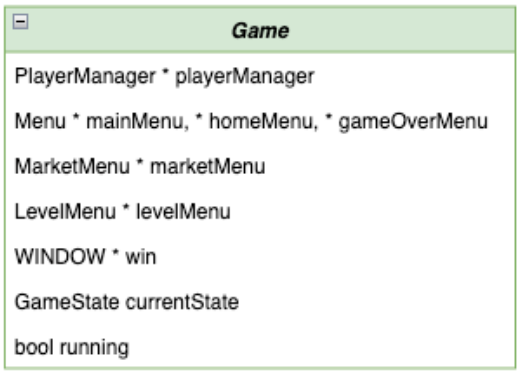

Figura 3: Classe Game con attributi

La classe *Game* ha funzioni di alto livello per il gioco:
- Gestisce il flusso di gioco attraverso gli stati di gioco (vedi la sezione relativa agli stati di gioco) e seleziona il menu da utilizzare
- Crea e gestisce il controllore del giocatore (*PlayerManager*)
- Gestisce il salvataggio e il caricamento dei dati da un file esterno

Ogni menu di gioco gestisce gli input del giocatore, esegue il rendering degli elementi a schermo e ritorna un valore di tipo *GameState* che è salvato nell'attributo *currentState* ed è utilizzato dall'oggetto *Game* nel ciclo successivo per decidere quale menu lanciare. Il loop di gioco è controllato dal booleano *running*, che diventa *false* quando è terminato il gioco (ad esempio, selezionando *Esci* da un menu).

Stati di gioco

Figura 4: Classe enumerabile GameState

Gli stati di gioco di ***GameState*** sono utilizzati dalla classe *Game* per definire quale menu lanciare nel ciclo corrente.
Segue una breve descrizione degli stati di gioco:
- *MainMenu*: è lanciato il menu iniziale

- *Home*: è lanciato il menu home
- *Market*: è lanciato il menu del mercato
- *Playing*: è lanciato il menu del gioco vero e proprio
- *GameOver*: è lanciato il menu di game over
- *SaveGame*: è lanciato il metodo *saveGame()* di Game per salvare i dati della partita corrente
- *LoadGame*: è lanciato il metodo *loadGame()* di Game per caricare i dati da un file esterno, se trovato
- *ExitGame*: l'attributo di *Game running* è posto false e il ciclo si interrompe, completando il programma

Classe PlayerManager e dati salvati

La classe ***PlayerManager*** controlla le statistiche del giocatore, gestisce gli acquisti di armi e vita massima utilizzando la valuta di gioco e gestisce l'equipaggiamento del giocatore (singola arma equipaggiata).

PlayerManager si interfaccia con i menu *Mercato* in fase di acquisto di nuove armi o equipaggiamento di una arma acquistata e con il menu *In-Game* per l'inizializzazione del giocatore usando i dati relativi alla vita massima e all'arma equipaggiata al momento dell'avvio della partita.

Infine, PlayerManager gestisce il punteggio record raggiunto dal giocatore, aggiornandolo durante il game over di una partita in caso di nuovo record raggiunto.

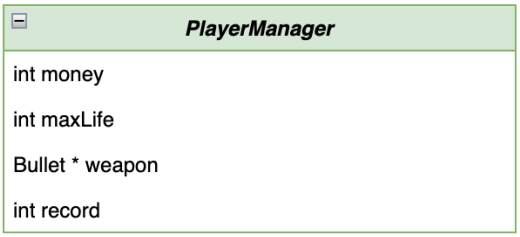

Figura 5: Classe PlayerManager con attributi

PlayerManager quindi si occupa della memorizzazione dei dati della partita corrente, e in caso di caricamento di una partita salvata i dati di *PlayerManager* sono aggiornati per rispecchiare i dati della partita caricata.

Gli unici dati necessari per salvare una partita sono contenuti in *PlayerManager* e gestiti da tale classe:

- Denaro corrente
- Vita massima del giocatore
- Per ogni arma:
    - È acquistata? (booleano)
    - È equipaggiata? (booleano)
- Il record di punteggio massimo raggiunto dalla creazione del file di salvataggio.

Parametri e costanti globali

Uno degli obiettivi da raggiungere è quello di rendere il gioco adattabile a seconda di certi parametri che pur cambiando, permettono di mantenere la continuità e la correttezza del gioco.

Per rendere il gioco controllabile e modificabile agilmente, è stato creato un file contenente i parametri di gioco importanti:

- Sono presenti due parametri *HEIGHT* and *WIDTH* che sono le dimensioni della finestra di un livello
- Un parametro regola la velocità di gioco
- Un parametro regola il numero massimo di nemici a schermo
- Un parametro regola il numero di monete raccolte per ogni elemento *MoneyBonus* e il numero di tali elementi per livello
- Un parametro regola il numero di vite massime necessario per aumentare il livello iniziale della partita

Questi parametri possono essere cambiati per modificare le funzionalità di gioco, senza comprometterne il funzionamento corretto.

Dati del gioco

I dati di gioco sono salvati in file specifici che raccolgono ad esempio le varie tipologie di nemici e le rispettive armi, i vari tipi di muro, le armi del giocatore. Il programma è stato studiato per permettere la creazione di nuovi oggetti e l'aggiunta di tali oggetti nel gioco minimizzando le modifiche al codice all'interno delle altre classi.

Progettazione e sviluppo di menu ed elementi di gioco

Gli elementi delle fasi di gioco

Classi degli elementi di base

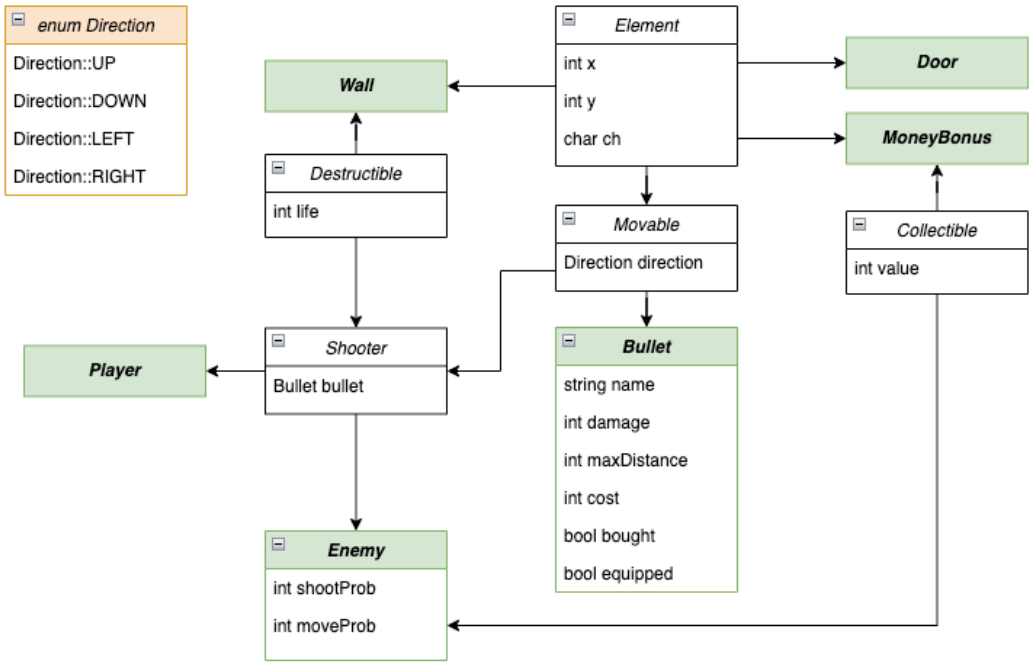

Figura 6: Schema relazionale tra le classi degli elementi di un livello di gioco con rappresentazione degli attributi

Lo sviluppo degli elementi di gioco è stato completato a partire dalla costruzione di classi base che racchiudono attributi comuni agli elementi di gioco. La classe ***Element*** rappresenta un elemento generico che presenta 3 caratteristiche fondamentali: le coordinate x e y di posizione e il carattere di rappresentazione dell'elemento a schermo. Inoltre, ogni elemento ha i metodi ***outOfBounds()*** e ***collision(Element e)*** che valutano se l'elemento è all'interno del livello di gioco e se c'è una collisione con un altro elemento. Le altre due classi base sviluppate sono la classe ***Collectible*** che contiene un attributo di valore e una classe ***Destructible*** che contiene un attributo di vita residua. Dalle 3 classi base, è stato possibile costruire tutti gli elementi di gioco presenti in un livello. L'elemento più semplice è un elemento fisso in una posizione che non ha altre funzionalità, utilizzato come "porta" nel gioco.

Dalla classe *Element* è stata derivata una classe ***Movable***, che rappresenta un elemento in grado di muoversi lungo una direzione, definita dall'enumerabile **Direction**. Dalla classe *Movable* deriva la classe ***Bullet***, che rappresenta un proiettile o arma, utilizzata sia dal giocatore che dai nemici. Per questo motivo, è stata creata la classe ***Shooter***, un *Movable* e *Destructible* che equipaggia un arma (*Bullet*). La classe *Shooter* è quella che origina il giocatore. Per i nemici, è stato necessario lo sviluppo di una ulteriore classe ***Enemy*** che compone la classe *Shooter* e la classe *Collectible*, e possiede due attributi extra che regolano la probabilità di movimento e sparo di un particolare nemico.

Infine, per rappresentare i muri di un livello, è stata creata la classe ***Wall*** come composizione semplice delle classi *Destructible* e *Element*: in effetti, un "muro" di gioco ha una vita, una posizione e un carattere che lo rappresenta.

Classe Level: il livello di gioco

Gli elementi di base descritti sono utilizzati dalla classe ***Level***, che rappresenta un singolo livello di gioco e si occupa della creazione dei muri nel livello, dei nemici e della valutazione delle posizioni di gioco per permettere gli spostamenti degli elementi all'interno del livello. Le dimensioni del livello sono gestite da costanti globali (vedi sezione: parametri e costanti globali).

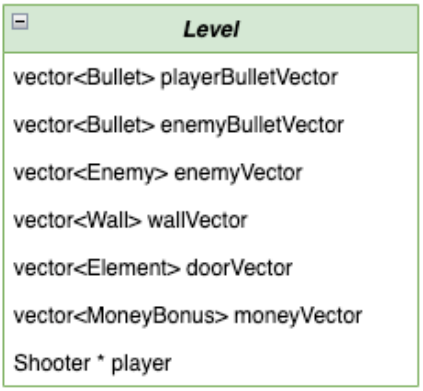

Figura 7: Classe Livello con attributi

Ogni livello di gioco è caratterizzato da:

- Un giocatore (*Shooter * player*). È utilizzato un puntatore perchè il player è gestito dal menu di livello (vedi sezione relativa ai menù del gioco) e non viene creato un nuovo *Shooter* player per ogni nuovo livello.
- Un vettore di proiettili sparati dal giocatore *playerBulletVector*
- Un vettore di proiettili sparati dai nemici *enemyBulletVector*
- Un vettore di nemici presenti *enemyVector*
- Un vettore di muri *wallVector*, ridimensionato in caso di distruzione di uno degli elementi di tipo *Wall*
- Un vettore di due portali che permettono al giocatore di passare da un livello all'altro
- Un vettore di bonus presenti nel livello *moneyVector*

Ogni livello quindi contiene diversi vettori di elementi che lo compongono, mentre la logica di gioco è gestita dal menù di livello (sezione relativa ai menù di gioco).

Il costruttore di *Level* prende in input un intero che rappresenta il livello di difficoltà, e un vettore di vettori di interi che contiene un insieme di coordinate nelle quali inserire gli elementi *Wall* (muri).

I menù di gioco

Classe Menu

Il gioco presenta 5 menù: 3 menù semplici, (Game Over, Main Menu e Home) e due menù che gestiscono logiche più complesse rispetto ai menu di base (Mercato e In-Game).

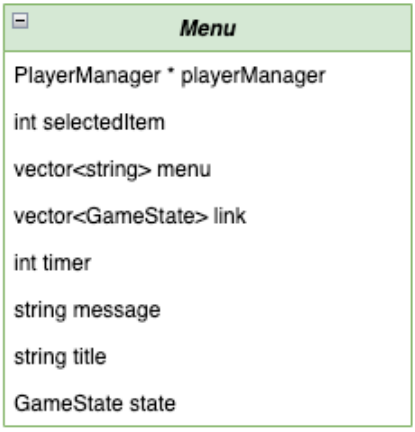

Figura 8: Classe Menu con attributi

I 3 menu semplici sono oggetti della classe ***Menu***, che è caratterizzata da un vettore di stringhe (testi delle voci di un menu) e da un corrispondente vettore di stati di gioco che sono attivati quando la voce del menu è selezionata (vedi sezione relativa agli stati di gioco). Il menu presenta un timer e un messaggio (stringa) che gestiscono la comparsa di messaggi temporanei in un menu. Infine ogni menu ha un attributo di tipo *GameState* per gestire il flusso di gioco (vedi relativa sezione).

La classe M*enu* si occupa di gestire l'input dell'utente e di effettuare il rendering degli elementi del menu, che sono le voci del menu e i prompt dei tasti. La voce del menu attualmente selezionata è definita dall'attributo *selectedItem*.

Classe Mercato

La classe ***Mercato*** è un menu che gestisce l'acquisto di armi e vita extra, e inoltre gestisce l'equipaggiamento di armi già acquistate e, oltre ad un normale menu, mostra le statistiche del giocatore (vita, arma e denaro). Pertanto, è stato necessario modificare i metodi di gestione degli input (*handleInput()*) e di rendering dell'input (*render()*) della classe *Menu* originaria. Non sono presenti ulteriori attributi.

Classe LevelMenu

La classe ***LevelMenu*** è un menu complesso, e svolge multiple funzioni:

- Riceve l'input dell'utente per muovere il cursore del giocatore, sparare o uscire dal gioco
- Gestisce la logica di gioco aggiornando le posizioni dei nemici, dei proiettili e valutando le collisioni degli elementi
- Elimina eventuali elementi distruttibili che hanno vita minore o uguale a 0 dai relativi vettori
- Gestisce la creazione di nuovi livelli e lo spostamento tra un livello e altro del giocatore quando avviene la collisione con una *Door*
- Esegue il rendering del livello attuale, dei dati di gioco (vita, monete, livello raggiunto) e delle istruzioni dei comandi per l'utente.
- Aggiorna il punteggio della partita attuale e se viene raggiunto un nuovo record, aggiorna il record della classe *PlayerManager*.

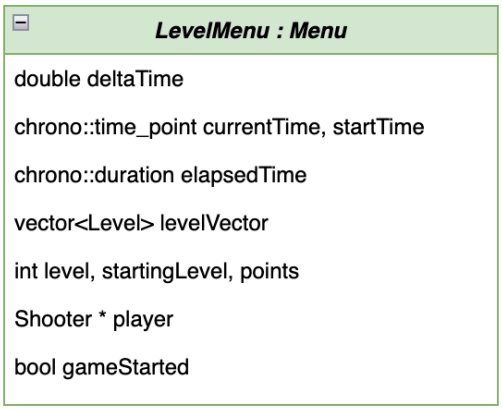

Figura 9: Classe LevelMenu con attributi

È presente un booleano che verifica se il gioco è iniziato oppure no, per inizializzare i dati in caso di gioco appena avviato.

Sono presenti 3 interi: *level* che rappresenta il livello raggiunto a partire da 0, *startingLevel* che permette al gioco di stabilire quale sia il livello di partenza in funzione delle statistiche raggiunge dal giocatore e *points* che contiene il punteggio raggiunto dal giocatore nella partita corrente. *startingLevel* è il parametro in input delle classi *Level*.

L'attributo più importante per il *LevelMenu* è il vettore *levelVector* di oggetti di tipo *Level*, che rappresenta una catena di livelli di gioco, e ciascun livello è caratterizzato dagli elementi descritti nella sezione relativa alla classe Level.

Quando il giocatore entra in collisione con la porta di avanzamento del livello più avanzato raggiunto, viene inizializzato e aggiunto un nuovo livello al vettore *levelVector*. Quando invece un giocatore entra in un livello già raggiunto, è mostrato tale livello con le caratteristiche preservate (disposizione dei muri, nemici rimasti, bonus rimasti).

Quando si raggiunge il Game Over, *gameStarted* è posto in *false* e sono eliminati i puntatori ai vettori di ogni livello e i puntatori ai livelli di *levelVector*. Lo stato di gioco passa a *GameOver* e il controllore *Game* esegue la logica del manu di game over come descritto nella sezione *Menu*.