Relazione

Progetto 2021/2022
Bacci Manuel - 0001040090 - manuel.bacci@studio.unibo.it

Introduzione

Le librerie “ncurses.h“ e “curses.h“, implementano le funzioni utili a gestire la grafica del programma attraverso un terminale di caratteri. Utilizzando queste funzioni diventa possibile schematizzare in codifica ASCII la grafica necessaria ad implementare un gioco Roguelike.

Il gioco in questione si sviluppa in una mappa senza traguardo, composta da infiniti livelli. Ciascun livello è associato ad una stanza differente, generata in modo casuale. All’interno di ogni stanza sono presenti diverse porte, ostacoli e nemici. La mappa viene esplorata stanza dopo stanza dal giocatore comandato da tastiera, che può proseguire nell’esplorazione o tornare indietro e trovare tutto come lo ha lasciato. Sparando dei proiettili, il giocatore deve sconfiggere tutti i nemici generati all’interno del livello in cui si trova. Se un nemico viene colpito da uno dei proiettili sparati dal giocatore, perde una quantità di punti vita pari al suo potenziale di danno. Ogni volta che il giocatore sconfigge un nemico, ossia quando i punti vita del suo bersaglio raggiungono lo zero, guadagna una determinata quantità di punti esperienza. Una volte che tutti i nemici sono stati sconfitti, le porte presenti all’interno della stanza vengono aperte, permettendo al giocatore di attraversarle per dirigersi verso un livello successivo non ancora esplorato, o verso un livello precedente. E’ inoltre possibile che l’ultimo nemico sconfitto generi un potere, con cui il giocatore può interagire per ripristinare completamente i suoi punti vita. Nel momento in cui il giocatore interagisce con questo potere, è anche probabile che compaia nel livello una nuova porta segreta, diretta verso una stanza nascosta in cui è presente un artefatto. Gli artefatti migliorano una statistica a caso fra gli attributi associati al giocatore: i suoi punti vita, il suo punteggio, il tempo di ricarica dei suoi attacchi, il suo potenziale di danno, la gittata o la velocità dei suoi proiettili. Se all’interno del livello il giocatore viene colpito da un proiettile o da un attacco a corpo sferrato da un nemico, perde una quantità di punti vita pari al suo potenziale di danno. Il giocatore possiede un numero limitato di cuori, ciascuno dei quali corrisponde a dieci punti vita, arrotondati per eccesso. Il gioco termina una volta che i cuori del giocatore scendono a zero.

Comandi

Premendo o tenendo premuti i tasti “WASD” è possibile movere l’icona del giocatore all’interno della mappa di gioco: con “W” o “w” verso l’alto, con “S” o “s” verso il basso, con “A” o “a” verso sinistra e con “D” o “d” verso destra. Premendo o tenendo premuti i tasti direzionali “↓↑←→” è invece possibile far si che il giocatore spari dei proiettili: con “↑” verso l’alto, con “↓” verso il basso, con “←” verso sinistra e con “→” verso destra.

Stanze

Le stanze sono implementate come oggetti della classe “class Room“. Ogni stanza è descritta da una finestra: “WINDOW* room“ il cui numero di righe: “int height“ e di colonne: “int width“ sono determinati casualmente in un intervallo limitato di valori. Il limite inferiore di questo intervallo corrisponde alla minima dimensione in grado di garantire che, anche nel peggiore dei casi, tutti gli elementi che verranno collocati all’interno della stanza saranno correttamente distanziati e impaginati nello spazio compreso fra i suoi lati. Il limite superiore corrisponde invece alla massima dimensione che consente di stampare interamente la stanza sfruttando il numero di righe e di colonne messe a disposizione dal terminale, facendo attenzione a non sovrapporla con gli altri elementi dell’interfaccia.

Il metodo costruttore della classe: “Room (int, int)“, determina casualmente le dimensioni della stanza utilizzando come limite superiore il numero di righe e di colonne del terminale fisico, passati come argomento della funzione. Genera poi un puntatore ad una nuova finestra delle dimensioni così ricavare e abilita la ricezione dell’Input immesso da tastiera nel terminale corrispondente. Imposta inoltre un intervallo di attesa fra ogni lettura dei valori rilevati alla pressione dei tasti. Quest’ultimo parametro determina la frequenza di aggiornamento di tutti gli elementi della grafica.

Oltre al metodo costruttore e ai metodi necessari a restituire le variabili protette: “void area (int&, int&)“ e “WINDOW* window()“, la classe dispone di alcuni metodi ausiliari. Il metodo “void draw()“ stampa all'interno della finestra un contorno che delimita i bordi della stanza, utilizzando il carattere '|' per i lati verticali, '-' per gli orizzontali e '+' per gli angoli. Il metodo “int input()“ rileva l'Input immesso da tastiera nel terminale corrispondente alla finestra. Il metodo “char get (int, int)“ verifica il carattere stampato alle coordinate indicate, rispetto al sistema della finestra. Il metodo “void empty (int&, int&)“ determina una posizione vuota casuale all’interno della stanza. Il metodo “void clear()“ cancella completamente il contenuto stampato nella finestra. Infine, il metodo “void refresh()“ stampa l'impostazione grafica attuale della finestra sullo schermo fisico del terminale.

Entità

Le entità con cui l’utente può interagire sono implementate come oggetti delle rispettive classi, tutte derivate dalla stessa base, ossia “class Entity”. Ogni entità è descritta dalla stanza a cui appartiene: “Room room", da una posizione all’interno di questa stanza: “int y, x", da un’icona che ne schematizza la grafica: “char icon” e da un insieme di statistiche e di stati: “Statistics stats, status”. L’insieme delle statistiche è specificato da una variabile della struttura dati “struct Statistics” ed elenca i valori associati alle caratteristiche principali dell’entità. L’insieme degli stati, implementato allo stesso modo, memorizza invece la variazione di questi attributi all’avanzare della partita.

La statistica “int health” indica i punti vita dell'entità e corrisponde alla quantità di danni che essa può sopportare prima di essere distrutta. L’attributo di stato corrispondente a questa statistica decrementa ogni volta che l’entità subisce un attacco. Quando il valore associato a questo stato diventa minore o uguale a zero, il giocatore viene sconfitto e il gioco termina. La statistica “Movement movement” descrive lo stile di movimento dell'entità e può assumere uno dei valori elencati dall’enumeratore “enum Movement”: “Casual” se essa si muove in modo casuale, “Following" se segue un bersaglio presente all’interno della sua area visiva o “Teleporting" se si teletrasporta in posizioni casuali della stanza, oppure “Stop" se l’entità è ferma o se non è necessario specificarne un movimento. La statistica “int speed” indica la velocità di movimento dell'entità e corrisponde al tempo che intercorre fra ogni suo passo ed il passo ad esso successivo. L’attributo di stato corrispondente a questa statistica decrementa ad ogni aggiornamento della grafica e viene ripristinato quando raggiunge lo zero. L’entità è in grado di muoversi solo nell’istante in cui il valore associato a questo stato è esattamente zero. Minore è il valore associato alla statistica, maggiore è la velocità di movimento effettiva dell’entità. La statistica “Attack attack” descrive lo stile di attacco dell'entità e può assumere uno dei valori elencati dall’enumeratore “enum Attack”: “Body” se essa attacca corpo a corpo, “Directional” se spara un proiettile nella direzione di un bersaglio presente all’interno della sua area visiva o “Plus" e “Cross” se spara quattro proiettili le cui traiettorie formano un più o una croce, oppure “Harmless” se l’entità è innocua o se non è necessario specificarne

un attacco. La statistica “int cooldown” indica il tempo di caricamento degli attacchi dell'entità e corrisponde al tempo che intercorre fra ogni suo attacco e l'attacco ad esso successivo. L’attributo di stato corrispondente a questa statistica decrementa ad ogni aggiornamento della grafica e viene ripristinato quando raggiunge lo zero. L’entità è in grado di attaccare solo nell’istante in cui il valore associato a questo stato è esattamente zero. Minore è il valore associato alla statistica, maggiore è la velocità di attacco effettiva dell’entità. La statistica “int damage” indica il potenziale di danno dell'entità e corrisponde alla quantità di punti vita che ogni suo attacco sottrae al bersaglio. La statistica “int shots” indica la velocità dei proiettili sparati dell'entità e corrisponde al tempo che intercorre fra ogni loro movimento e il movimento ad esso successivo. L’attributo di stato corrispondente a questa statistica decrementa ad ogni aggiornamento della grafica e viene ripristinato quando raggiunge lo zero. I proiettili sono in grado di avanzare solo nell’istante in cui il valore associato a questo stato è esattamente zero. Minore è il valore associato a questa statistica maggiore è la velocità di movimento effettiva dei proiettili sparati dell’entità. La statistica “int range” indica la gittata dei proiettili sparati dall'entità e corrisponde alla massima distanza che questi possono percorrere prima di essere distrutti. La statistica “int sight” indica l’area visiva dell'entità e corrisponde alla massima distanza entro la quale essa può interagire con il bersaglio. La statistica “int points” indica i punti esperienza dell'entità e corrisponde alla quantità di punti che il giocatore possiede o che può guadagnare sconfiggendo il bersaglio. La statistica “Direction direction” descrive la direzione di movimento dell'entità e può assumere uno dei valori elencati dall’enumeratore “enum Direction”: “North” se al suo passo successivo essa si muoverà verso l’alto, “South” se è diretta verso il basso, “East” se verso destra, “West” se verso sinistra o una delle possibili combinazioni di questi: “NorthEast”, “NorthWest”, “SouthEast” e “SouthWest”, oppure “Point” se l’entità deve stare ferma nel punto in cui è posizionata o se non è necessario specificarne una direzione. La statistica “Extention extention” descrive la posizione dell’estensione rispetto alla posizione dell’entità e può assumere uno dei valori elencati dall’enumeratore "enum Extention”: “Top” se si trova immediatamente sopra, “Bottom” se è sotto, “Right” se è a destra o “Left” se è a sinistra, oppure “Alone” se non è presente.

Il metodo costruttore della classe: “Entity (Room, char, int, int)” inizializza le caratteristiche principali dell’entità: la finestra a cui appartiene, la sua icona e la sua posizione in relazione al sistema della stanza in cui è collocata, a seconda dei valori indicati dalla classe derivata che utilizza questa classe base. Assegna poi alle statistiche associate all’entità dei valori neutri, che non implicano alcun movimento o danno, e assegna agli attributi che ne indicano lo stato i valori delle corrispondenti statistiche. Infine, stampa l’entità all’interno della stanza.

Oltre al metodo costruttore e ai metodi necessari a restituire le variabili protette: “void position (int&, int&)”, "Statistics statistics()” e “Statistics state()”, la classe dispone di alcuni metodi ausiliari. Il metodo “void extention (int&, int&, Direction)” determina la posizione occupata dall’estensione dell’entità o la posizione che essa occuperà muovendo il suo passo successivo nella direzione specificata. Il metodo “void next (int&, int&, Direction)” ritorna le coordinate corrispondenti al passo successivo dell’entità nella direzione specificata o nella direzione assegnata all’apposita statistica. Il metodo “void draw()” stampa alla posizione occupata dell’entità il carattere associato alla sua icona e ripete lo stesso procedimento anche per la sua estensione, qualora esista. Il metodo “void clear()” cancella i caratteri stampati alle posizioni occupate dall’entità e dalla sua estensione, sovrascrivendoli con un carattere bianco: ‘ ‘. I metodi “void move()” e “void move (int, int)” spostano l’entità alle coordinate indicate o alla posizione che essa occuperà muovendo il suo passo successivo nella direzione stabilita, qualora non siano presenti ostacoli. Il metodo “void decrease (Variant)” decrementa il valore assegnato all’attributo di stato indicato, qualora esso non abbia raggiunto il valore minimo. Il parametro associato all’attributo di stato può assumere uno dei valori elencati dall’enumeratore “enum Variant”: “Cooldown”, “Shots", “Speed", “Range” o “Health”. Il metodo “void reset (Variant)” ripristina l’attributo di stato specificato al valore della corrispondente statistica, qualora abbia raggiunto lo zero. Il metodo “void hit (int)” decrementa lo stato associato ai punti vita dell’entità di una quantità pari al valore indicato. Il metodo “void score (int)” incrementa il valore della statistica che indica i punti esperienza dell’entità della quantità specificata. Infine, il metodo “void direction (Direction)” modifica il valore assegnato alla statistica associata alla direzione dell’entità con la nuova direzione specificata.

Porte

Le porte sono implementate come oggetti della classe “class Door”, derivata da “class Entity”. Oltre alle caratteristiche comuni a tutte le entità, ogni porta è descritta da un orientamento: “Orientation orientation”. Questa variabile può assumere uno dei valori elencati dall’enumeratore “enum Orientation”: “Horizontal” se la porta è posizionata in orizzontale o “Vertical" se è verticale.

Il metodo costruttore della classe: “Door (Room)”, implementa la porta come un’entità, appartenente alla stanza indicata, la cui icona è ‘X’ e la cui posizione è irrilevante al momento dell’inizializzazione. Determina poi casualmente se la porta deve essere orientata in orizzontale o in verticale. Nel caso in cui la porta debba essere orientata in orizzontale, sceglie se collocarla sul lato superiore o inferiore della stanza e genera un’ascissa a cui posizionarne l’origine, facendo in modo che la porta sia correttamente rappresentata e distanziata nello spazio compreso fra i due lati verticali. Se la porta deve invece essere orientata in verticale, assegna alla sua origine la posizione indicata dall’ascissa di uno dei due lati verticali e da un’ordinata generata casualmente. Stampa infine la porta generata all’interno della stanza a cui appartiene. Il metodo costruttore “Door (Room, Door, bool)” implementa il metodo precedente con le funzionalità utili a generare una nuova porta, a seconda del valore booleano passato come parametro, o in modo speculare rispetto alla porta indicata, oppure in modo che le due porte non siano posizionate sullo stesso lato della stanza. Infine, il metodo costruttore “Door (Room, Door, Door)” permette di generare una nuova porta in modo che essa non interferisca con le altre due porte specificate, ossia con le porte di ingresso e uscita già posizionate all’interno della stanza.

Oltre ai metodi costruttori e al metodo necessario a restituire la variabile protetta: “Orientation inclination()", la classe dispone di alcuni metodi ausiliari. Il metodo “void draw()" stampa la porta come una sequenza di sei caratteri corrispondenti alla sua icona, se è orientata in orizzontale, o di tre se è verticale. Il metodo “void open()" sostituisce l’icona ‘X’ della porta con un carattere bianco: ‘ ‘, per farla apparire graficamente aperta.

Muri

I muri sono implementati come oggetti della classe “class Wall”, derivata da “class Entity”. Oltre alle caratteristiche comuni a tutte le entità, ogni muro è descritto dall’orientamento del suo muro principale: “Orientation orientation", dalla sezione a cui appartiene il suo muro secondario: “Side side" e dall’ascissa o dall’ordinata a cui hanno origine i due varchi perpendicolari che dividono questi muri: “int yp, xp”. La variabile associata all’orientamento può assumere uno dei valori elencati dall’enumeratore “enum Orientation”: “Horizontal” se il muro principale è posizionato in orizzontale o “Vertical” se è verticale, oppure “None" se non è presente. La sezione può invece assumere uno dei valori elencati dall’enumeratore “enum Side": “First” se il muro secondario è posizionato nella prima delle due sezioni in cui il muro principale suddivide la stanza o “Second” se è nella seconda, oppure “Zero" se non esiste.

Il metodo costruttore della classe: “Wall (Room)”, implementa il muro come un’entità, appartenente alla stanza indicata, la cui icona e la cui posizione sono irrilevanti al momento dell’inizializzazione. Determina poi casualmente l’orientamento del muro principale e la posizione del muro secondario in relazione alle sezioni in cui questo suddivide la stanza. Genera inoltre un'ordinata e un'ascissa alle quali collocare il muro principale e il muro secondario a seconda del loro orientamento. Questi valori sono determinati casualmente in modo da garantire che, anche nel caso pessimo,

fra i lati della stanza e i due muri perpendicolari vi sia lo spazio necessario a rappresentare correttamente gli altri eventuali elementi della grafica, come ad esempio delle porte. Infine, stampa la coppia di muri all'interno della stanza a cui appartengono.

Oltre al metodo costruttore, la classe dispone di alcuni metodi ausiliari. Il metodo "void draw()" stampa il muro principale e il muro secondario all'interno della stanza, qualora esistano. Se il muro principale è orientato in orizzontale, stampa una sequenza orizzontale di caratteri '-' che inizia e termina con un '+' e percorre la stanza dal lato sinistro al lato destro. L'ordinata a cui ha origine il varco che divide questo muro è determinata casualmente in modo che esso venga rappresentato correttamente, che sia opportunamente distanziato dai due muri verticali e che non interferisca con gli altri elementi della grafica, come ad esempio con l'origine del muro secondario. A partire da questa ordinata, sovrascrive il muro principale stampando una linea orizzontale di sei caratteri bianchi: ' '. Se il muro secondario è posizionato nella prima delle due sezioni in cui il muro principale suddivide la stanza, stampa una sequenza verticale di caratteri '|' che inizia e termina con un '+' e percorre la stanza dal lato superiore al muro principale. Se il muro secondario è invece collocato nelle seconda delle due sezioni, questa sequenza verticale deve percorrere la stanza dal muro principale al lato inferiore. In entrambi i casi, a partire da un'ascissa generata casualmente, sovrascrive il muro secondario stampando una linea verticale di tre caratteri bianchi: ' '. Nel caso in cui il muro principale sia invece orientato in verticale, stampa un muro verticale che percorre la stanza dal lato superiore al lato inferiore. Stampa poi un muro orizzontale che percorre la stanza dal lato sinistro al muro principale o dal muro principale al lato destro, a seconda della posizione del muro secondario. Genera infine due varchi perpendicolari che sovrascrivono una porzione di questi muri.

Nemici

I nemici sono implementati come oggetti della classe "class Enemy", derivata da "class Entity". Oltre alle caratteristiche comuni a tutte le entità, ogni nemico è descritto da una razza: "Breed breed". Questa variabile può assumere uno dei valori elencati dall'enumeratore "enum Breed": "Fly", "Mosquito", "Wasp", "Hornet", "Worm", "Mole", "Snake", "Snail" o "Mix".

Il metodo costruttore della classe: "Enemy (Room, int, int)", implementa il nemico come un'entità, appartenente alla stanza indicata, la cui icona è '@' e la cui posizione è specificata al momento dell'inizializzazione. Sceglie poi casualmente a quale delle razze disponibili appartiene il nemico e determina se esso presenta o meno un'estensione. In caso affermativo, stabilisce la sua direzione in relazione alla posizione dell'entità, in modo che essa non interferisca con gli altri elementi della grafica. Assegna inoltre alle statistiche associate al nemico i valori determinati dalla razza a cui appartiene e assegna agli attributi associati al suo stato attuale i valori delle corrispondenti statistiche. Infine, stampa il nemico all'interno della stanza a cui appartiene.

Poteri e Artefatti

Gli artefatti sono implementati come oggetti della classe "class Artifact", derivata da "class Entity". Oltre alle caratteristiche comuni a tutte le entità, ogni artefatto è descritto da una variabile booleana che indica se esso racchiude o meno un potere: "bool power".

Il metodo costruttore della classe: "Artifact (Room, Player, int, int, bool)", implementa l'artefatto come un'entità, appartenente alla stanza indicata, la cui icona è 'A' e la cui posizione è specificata al momento dell'inizializzazione. Assegna poi alle statistiche dell'artefatto i valori delle corrispondenti statistiche associate al giocatore indicato e incrementa o decrementa di un valore specifico una a caso fra queste, qualora essa non abbia raggiunto il limite massimo o minimo. A seconda del valore booleano passato come parametro, determina inoltre se l'artefatto racchiude o meno un potere. Infine, stampa l'artefatto all'interno della stanza a cui appartiene. Infine, determina casualmente la presenza e la direzione della sua estensione.

Oltre al metodo costruttore, la classe dispone di un metodo ausiliario. Il metodo "Player upgrade (Player)", qualora l'artefatto racchiuda un potere, ripristina completamente l'attributo di stato associato ai punti vita del giocatore. In caso contrario, modifica le statistiche del giocatore sostituendole con le statistiche migliorate dell'artefatto.

Giocatore

Il giocatore è implementato come un oggetto della classe "class Player", derivata da "class Entity". Oltre alle caratteristiche comuni a tutte le entità, il giocatore è descritto dalla condizioni attuali della sua transizione verso un nuovo livello: "Transition destination". La variabile associata alla transizione, implementata dalla struttura dati "struct Transition", è descritta dall'ultima destinazione raggiunta dal giocatore: "Destination last", dalla sua destinazione corrente: "Destination current", dalla porta attraversata: "Door door" e dall'ascissa o dall'ordinata del punto preciso in cui il giocatore sta attraversando la porta, in relazione alla posizione in cui essa ha origine: "int coordinate". Alle variabili associate all'ultima destinazione raggiunta e alla destinazione corrente del giocatore è assegnato uno dei valori elencati dall'enumeratore "enum Destination": "Next" se l'ultima porta che esso ha utilizzato o se la porta che sta utilizzando è diretta verso un livello successivo, "Previous" se il livello è precedente, "Secondary" se è secondario o "Special" se è speciale, oppure "Current" se il giocatore si trova attualmente nella stanza di partenza o se non sta attraversando nessuna porta.

Il metodo costruttore della classe: "Player (Room, int, int)", implementa il giocatore come un'entità, appartenente alla stanza indicata, la cui icona è 'P' e la cui posizione è specificata al momento dell'inizializzazione. Assegna poi alle statistiche del giocatore degli opportuni valori, tali da rendere il gioco equilibrato, e assegna agli attributi associati al suo stato attuale i valori delle corrispondenti statistiche.

Oltre al metodo costruttore e al metodo necessario a restituire la variabile protetta: "Transition passage(), la classe dispone di alcuni metodi ausiliari. Il metodo "void upgrade (Statistics)" sovrascrive le statistiche del giocatore con le nuove statistiche indicate. Il metodo "void transfer (Room, int, int)" trasferisce il giocatore dalla stanza associata al livello in cui si trova attualmente alla stanza indicata, corrispondente ad un livello successivo, precedente, secondario o speciale. Sposta poi il giocatore alle coordinate specificate. Infine, imposta l'ultima destinazione raggiunta dal giocatore alla direzione della porta che esso stava attraversando immediatamente prima di essere trasferito e indica che esso non sta attualmente attraversando nessuna porta. Il metodo "void transition (Destination, Door, int)" assegna i valori indicati agli attributi che descrivono la transizione del giocatore: la sua destinazione, la porta utilizzata e la coordinata relativa del punto preciso in cui l'ha attraversata.

Proiettili

I proiettili sono implementati come oggetti della classe "class Bullet", derivata da "class Entity". Oltre alle caratteristiche comuni a tutte le entità, ogni proiettile è descritto dal soggetto che lo ha sparato: "Subject shooter". Alla variabile associata al soggetto è assegnato uno dei valori elencati dall'enumeratore "enum Subject": "User" se a sparare il proiettile è stato il giocatore o "Opponent" se è stato un nemico.

Il metodo costruttore della classe: "Bullet (Room, Player, Direction, int, int)", implementa il proiettile come un'entità, appartenente alla stanza indicata, la cui icona è '.' e la cui posizione è specificata al momento dell'inizializzazione. Assegna poi alle statistiche e agli stati che indicano la velocità, il danno e la gittata del proiettile i valori delle corrispondenti statistiche associate al giocatore indicato. Imposta infine la direzione del proiettile al valore specificato e indica che è stato sparato dal giocatore. Il metodo costruttore "Bullet (Room, Enemy, Direction, int, int)" modifica il metodo precedente utilizzando le statistiche del nemico indicato anziché quelle del giocatore. Oltre al metodo costruttore, la classe dispone del metodo necessario a restituire la variabile protetta: "Subject shooter()".

Livelli

I diversi livelli del gioco sono implementati come oggetti della classe "class Level". Ogni livello è descritto da una stanza: "Room room", da una coppia di muri perpendicolari: "Wall wall", da tre porte: "Door in, out, secondary", da una lista di nemici: "Enemies* enemies", da una lista di proiettili: "Bullets* bullets" e da un artefatto: "Artifact artifact". La presenza dell'artefatto è specificata della variabile booleana "bool generated", il cui valore indica se nel livello è stato generato o meno un artefatto con cui il giocatore non ha ancora interagito. Le variabili booleane "bool first, third, special" Indicano rispettivamente se nel livello sono presenti: una sola porta, esattamente tre porte o un numero indifferente di porte di cui però una è speciale.

Il metodo costruttore della classe: "Level (int, int)", genera casualmente una nuova stanza su cui strutturare un livello che corrisponda al punto di partenza della partita, utilizzando il numero di righe e di colonne del terminale fisico, passati come argomento della funzione. All'interno di questa stanza posiziona poi una sola porta di uscita, già aperta, e nessun muro, nemico o artefatto. Il metodo costruttore "Level (Player, int, int)" implementa il metodo precedente utilizzando un giocatore già esistente, da trasferire all'interno del nuovo livello. Genera quindi una nuova stanza su cui strutturare un livello che corrisponda alla destinazione corrente verso cui è diretto il giocatore. In questo livello non sono inizialmente presenti porte, muri, nemici o artefatti. Se la porta che il giocatore sta attraversando è diretta verso un livello speciale non ancora esplorato, genera nella stanza un'unica porta di ingresso speciale, già aperta, e la posiziona in modo speculare rispetto alla porta attraversata, per dare un'impressione di continuità della mappa. Aggiunge poi alla stanza un artefatto che non racchiude alcun potere, e che quindi modifica le statistiche del giocatore. Se la porta che il giocatore sta attraversando è invece diretta verso un livello successivo o secondario non ancora esplorato, posiziona nella stanza una coppia di muri perpendicolari e due porte chiuse: una d'ingresso e una d'uscita. Vi è inoltre un 25% di probabilità che nella stanza venga generata anche una terza porta secondaria, inizialmente chiusa. Aggiunge infine al livello un numero di nemici compreso fra uno e quattro.

Il metodo "void draw()" stampa interamente il contenuto del livello, ossia la stanza ad esso associata e tutte le entità presenti al suo interno.

Il metodo "Player initialize()" genera un nuovo giocatore in una posizione determinata casualmente fra tutti gli spazi liberi presenti all'interno della stanza. Il metodo "Player initialize (Player)" implementa il metodo precedente utilizzando un giocatore già esistente. Determina quindi la porta attraverso cui il giocatore entra nella stanza in base ai valori assegnati agli attributi che ne specificano la transizione. Sposta poi il giocatore in prossimità di questa porta per dare l'impressione che esso entri nella stanza in modo speculare rispetto a come ha attraversato la porta per cambiare livello. Infine, trasferisce e stampa il giocatore all'interno del nuovo livello.

Il metodo "Player process (Player)" elabora l'Input immesso da tastiera nel terminale corrispondente alla finestra associata alla stanza su cui si struttura il livello attuale. Modifica quindi la direzione del giocatore in base a quale dei tasti "WASD" è stato premuto dall'utente: con "W" verso l'alto, con "S" verso il basso, con "D" verso destra e con "A" verso sinistra. Gestisce poi il comportamento del giocatore in conseguenza a questa modifica. Se l'utente preme invece uno dei tasti freccia "↓↑←→", se il tempo di ricarica degli attacchi del giocatore ha raggiunto lo zero, genera un nuovo proiettile all'interno della stanza, sparato dal giocatore nella direzione indicata dalla freccia. Dopo aver gestito il comportamento del giocatore, prosegue in ordine con la gestione dei proiettili e dei nemici presenti all'interno della stanza.

Il metodo "Player handle (Player)" gestisce il comportamento del giocatore a seconda dell'entità con cui esso si interfaccerà muovendo il suo passo successivo nella direzione prestabilita. Se il passo successivo del giocatore si sovrappone ad un artefatto, cancella l'artefatto dalla stanza. Se il livello non è speciale e al suo interno non sono già presenti tre porte, quando il giocatore interagisce con questo artefatto, con una probabilità del 50%, viene generata una terza porta speciale, già aperta, all'interno della stanza. In caso contrario, ripristina completamente i punti vita del giocatore o modifica le sue statistiche. Se il passo successivo del giocatore si sovrappone ad uno degli spazi vuoti che rappresentano una porta aperta, agli attributi associati alla transizione del giocatore vengono assegnati i valori che ne descrivono il trasferimento verso un livello che corrisponde alla direzione di questa porta. Questo trasferimento avverrà attraversando la porta indicata nel punto preciso in cui il giocatore muoverà il suo passo successivo. Sposta infine il giocatore nella posizione indicata dalle coordinate che ne specificano il passo successivo, qualora il movimento sia possibile.

Il metodo "void create (Bullets*, Player, Direction)" posiziona nella stanza un nuovo proiettile, aggiungendolo in coda alla lista di cui è fornita la testa, sparato dal giocatore nella direzione indicata e generato alle coordinate successive alla posizione del giocatore rispetto a questa direzione. Questo proiettile viene creato solo nel caso in cui sovrascriva uno spazio vuoto o al più già occupato da un altro proiettile. Il metodo "void create (Bullets*, Enemy, Direction)" modifica il metodo precedente utilizzando la posizione del nemico indicato anziché quella del giocatore. Presta inoltre attenzione a incrementare o decrementare correttamente le coordinate del proiettile qualora questo interferisca con l'estensione del nemico, in modo da dare eventualmente l'impressione che il proiettile sia stato sparato dall'estensione stessa.

Il metodo "void erase (Bullets*, Bullets*)" scorre l'apposita lista dalla testa indicata alla coda, per trovare un proiettile che corrisponda a quello fornito. Qualora lo trovi, lo elimina dalla lista e lo cancella dalla stanza.

Il metodo "Player handle (Bullets*, Player)" gestisce il comportamento di ciascuno dei proiettili elencati nell'apposita lista, di cui è fornita la testa, a seconda dell'entità con cui esso si interfaccerà muovendo il suo passo successivo nella direzione prestabilita. Qualora, scorrendo la lista dalla testa alla coda, lo stato che indica la velocità del proiettile considerato abbia raggiunto lo zero, procede a gestirne il movimento. Se il proiettile è stato sparato da un nemico e il suo passo successivo si sovrappone al giocatore, decrementa lo stato associato ai punti vita del giocatore del potenziale di danno del proiettile, ossia del nemico che lo ha sparato. Se il proiettile è stato sparato dal giocatore e il suo passo successivo si sovrappone ad un nemico, procede con la gestione della salute del bersaglio. Infine, sia in questi casi, sia quando il passo successivo del proiettile si sovrappone ad uno spazio vuoto o occupato da un'altro proiettile ma lo stato che indica la sua gittata ha raggiunto lo zero, cancella il proiettile dalla stanza e lo elimina dalla lista.

Il metodo "Player create (Enemies*) posiziona nella stanza e aggiunge in coda alla lista di cui è indicata la testa un nuovo nemico, generato in una posizione determinata casualmente fra tutti gli spazi liberi presenti all'interno della stanza.

Il metodo "Player erase (Enemies*, Bullet, Player)" scorre l'apposita lista dalla testa fornita alla coda per verificare se le coordinate a cui è posizionato il nemico considerato o la sua estensione corrispondono al passo successivo del proiettile indicato. In caso affermativo, decrementa lo stato

associato ai suoi punti vita del potenziale di danno del proiettile, ossia del giocatore che lo ha sparato. Se i punti vita del nemico hanno raggiunto lo zero, incrementa la statistica associata ai punti esperienza del giocatore del valore assegnato alla sua statistica corrispondente. Elimina poi il nemico dalla lista e lo cancella dalla stanza. Se nella lista non è più presente alcun nemico, apre tutte le porte posizionate all'interno della stanza. Infine, con una probabilità di circa 33%, genera alla posizione dell'ultimo nemico eliminato un nuovo artefatto che racchiude un potere.

Il metodo "Player handle (Enemies*, Player)" gestisce il comportamento di ciascuno dei nemici elencati nell'apposita lista a seconda dell'entità con cui si interfaccerà muovendo il suo passo successivo nella direzione prestabilita. Scorrendo la lista dalla testa indicata alla coda, verifica per ciascuno dei nemici se nei suoi dintorni è presente il giocatore, ad una distanza inferiore al valore assegnato alla statistica associata alla sua vista. Se il giocatore è all'interno dell'area visiva del nemico, verifica se la traiettoria che li collega è intercettata da un qualche ostacolo. In caso negativo, se lo stato che indica la velocità del nemico considerato ha raggiunto lo zero, procede a gestirne il movimento. Se il valore della statistica che descrive lo stile di movimento del nemico indica che esso si muove teletrasportandosi, determina casualmente una posizione vuota all'interno della stanza in cui collocarlo, in modo che la sua estensione non si sovrapponga ad altri elementi della grafica. Se lo stile di movimento del nemico indica invece che esso si muove casualmente, modifica la sua direzione con una direzione determinata in modo casuale. Infine, se lo stile di movimento del nemico indica che esso si muove seguendo il giocatore, modifica la sua direzione a seconda della sua posizione in relazione a quella del giocatore. Muove poi il nemico verso la sua posizione successiva nella direzione così determinata. Se lo stato che indica il tempo di caricamento degli attacchi del nemico considerato ha raggiunto lo zero, procede anche a gestirne l'attacco. Se il valore della statistica associata allo stile di attacco del nemico indica che esso attacca sparando quattro proiettili le cui traiettorie formano graficamente un più, genera 4 nuovi proiettili, sparati dal nemico nelle quattro direzioni orizzontali e verticali: Nord, Sud, Est e Ovest. Se lo stile di attacco del nemico indica invece che esso attacca sparando quattro proiettili le cui traiettorie formano graficamente una croce, genera 4 nuovi proiettili, sparati dal nemico nelle quattro direzioni diagonali: Nord-Est, Nord-Ovest, Sud-Est e Sud-Ovest. Se invece attacca mirando al giocatore, genera un nuovo proiettile la cui direzione dipende della posizione del nemico in relazione a quella del giocatore. Verifica inoltre se il nemico considerato o la sua estensione sono a contatto con il giocatore e sono quindi in grado di sferrargli un attacco corpo a corpo.

Gioco

Il gioco in se è implementato come un oggetto della classe "class Game". E' descritto dal giocatore gestito dall'utente: "Player player", dalla lista dei livelli esplorati dal giocatore: "Levels* levels", dall'insieme delle statistiche e degli stati attuali del giocatore: "Statistics stats, state", e dal numero di righe e di colonne del terminale su cui viene eseguito: "int width, height". L'insieme dei livelli esplorati è gestito da un albero binario bidirezionale. Ogni nodo di quest'albero corrisponde ad una variabile dell'apposita struttura a quattro campi: "struct Levels". Il campo "Level level" descrive il livello corrente, ossia contiene il valore del nodo. Il campo "Levels* previous" memorizza il puntatore al livello precedente, ossia al nodo padre. Il campo "Levels* next" punta al livello successivo, ossia al nodo figlio nel sotto-albero sinistro. Infine, il campo "Levels* secondary" punta al livello secondario, ossia al nodo figlio nel sotto-albero destro.

Il metodo costruttore della classe: "Game (int, int)", genera un nuovo livello che corrisponde al punto di partenza del gioco e lo aggiunge in testa all'apposita lista, impostandolo come radice dell'albero binario in cui sono elencati tutti i livelli esplorati dal giocatore. Questo livello non è attualmente associato a nessun altro livello precedente, successivo o secondario. Genera poi un nuovo giocatore e lo inizializza all'interno del primo livello. Infine, stampa l'interfaccia grafica attuale sullo schermo fisico del terminale.

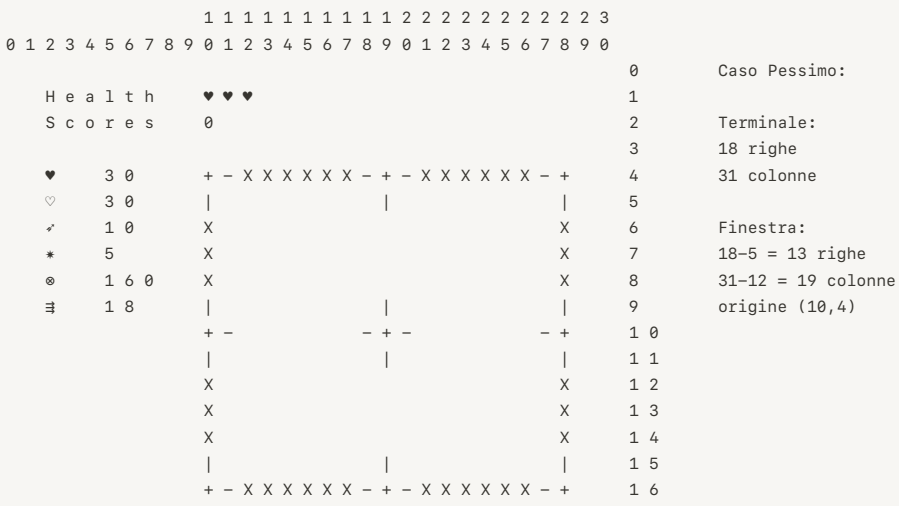
Il metodo "void hud()" aggiorna l'interfaccia grafica del gioco. Stampa nella corretta posizione del terminale gli indicatori associati alle principali caratteristiche del giocatore, seguiti dai valori assegnati ai corrispondenti stati o alle corrispondenti statistiche. Questi indicatori sono: "Health" per la salute residua, "Scores" per i punti esperienza, "❤" per i punti vita massimi, "♡" per i punti vita attuali, "➶" per la gittata dei proiettili, "✷" per il potenziale di danno, "◷" per il tempo di caricamento degli attacchi e "⇶" per la velocità dei proiettili.

Il metodo "int process()" gestisce il comportamento del livello attuale. Se il numero di righe o di colonne del terminale è inferiore al valore determinato all'inizio del gioco, ossia se l'utente ha diminuito le dimensioni della finestra nel corso dell'esecuzione, termina il processo indicando che si è verificato un errore. Se il valore assegnato all'attributo di stato che indica la salute del giocatore è uguale o inferiore a zero, termina l'esecuzione indicando che l'utente è stato sconfitto. Nel caso in cui i valori assegnati agli attributi che descrivono la transizione del giocatore indicano che esso è attualmente diretto verso un livello successivo, passa alla gestione del livello associato al nodo figlio nel sotto-albero sinistro del nodo corrispondente al livello attuale. Se il giocatore è invece diretto verso un livello precedente, passa alla gestione del nodo padre. Infine, se il giocatore è diretto verso un livello secondario o speciale, passa alla gestione del nodo figlio nel sotto-albero destro. Genera i livelli corrispondenti a questi nodi qualora essi non siano già stati esplorati e trasferisce il giocatore inizializzandolo nuovamente al loro interno. Se uno dei valori attualmente assegnati agli attributi di stato o alle statistiche del giocatore ha subito una variazione rispetto ai valori memorizzati in precedenza, aggiorna l'interfaccia grafica in modo da concretizzare questo cambiamento. Indica infine che l'esecuzione sta procedendo in modo corretto.

Il metodo "Levels* create (Levels*, Destination)" aggiunge all'albero di cui è fornita la radice un nuovo livello, che corrisponda alla destinazione attuale del giocatore. Se il giocatore è diretto verso un livello successivo non ancora esplorato, genera un nuovo livello e lo associa al nodo figlio nel sotto-albero sinistro del nodo corrispondente al livello attuale. Se il giocatore è invece diretto verso un livello secondario o speciale, associa il nuovo livello al nodo figlio nel sotto-albero destro del nodo corrispondente al livello attuale.

Main

Il metodo "int main()", ad ogni iterazione, ossia ad ogni aggiornamento della grafica, processa il comportamento del gioco e analizza il suo stato. Imposta inoltre il terminale nella corretta modalità per l'esecuzione del programma, impedendo all'utente di eseguire il gioco qualora le dimensioni del terminale fisico non siano sufficienti. Stampa infine a schermo i messaggi utili ad avvisare l'utente del cambio di stato del gioco, qualora si sia verificato un errore o qualora il giocatore sia stato sconfitto.



Fly

health	10
movement	Casual
speed	40
attack	Harmless
cooldown	0
damage	0
shots	0
range	0
sight	16
points	2

Mosquito

health	15
movement	Following
speed	60
attack	Body
cooldown	120
damage	2
shots	0
range	0
sight	14
points	6

Wasp

health	10
movement	Casual
speed	80
attack	Directional
cooldown	200
damage	4
shots	16
range	8
sight	8
points	8

Hornet

health	30
movement	Stop
speed	0
attack	Cross or Plus
cooldown	140
damage	6
shots	20
range	12
sight	20
points	10

Worm

health	5
movement	Teleport
speed	500
attack	Harmless
cooldown	0
damage	0
shots	0
range	0
sight	20
points	4

Mole

health	25
movement	Teleport
speed	500
attack	Cross or Plus
cooldown	220
damage	10
shots	22
range	10
sight	8
points	16

Snake

health	35
movement	Following
speed	100
attack	Directional
cooldown	180
damage	8
shots	18
range	14
sight	12
points	12

Snail

health	40
movement	Stop
speed	0
attack	Cross or Plus
cooldown	160
damage	12
shots	0
range	0
sight	12
points	14

Mix

health	5, 10, 15, 20, 25, 30, 35, 40
movement	Casual, Following, Teleport, Stop
speed	40, 60, 80, 100, 500
attack	Cross, Plus, Directional, Body, Harmless
cooldown	120, 140, 160, 180, 200, 220
damage	2, 4, 6, 8, 10, 12, 0
shots	18, 20, 22, 24
range	8, 10, 12, 14
sight	8, 12, 16, 20
points	2, 4, 6, 8, 10, 12, 14, 16

Player

health	30
cooldown	160
damage	5
shots	18
range	10
points	0