UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

Dipartimento di Informatica

Documentazione progetto per Metodi Avanzati di Programmazione

creare un'avventura grafica in Java

RELATORE: Marco Prencipe
MATRICOLA: 747001

ANNO ACCADEMICO 2022 / 2023

Indice

1 Introduzione

Il gioco è stato sviluppato in Java ed è un'avventura coinvolgente in 2D. Ambientato in un mondo affascinante, il giocatore assume il ruolo di un coraggioso personaggio che esplora una mappa di gioco limitata ma ricca di sfide e opportunità.
Il cuore del gioco ruota attorno al movimento del personaggio attraverso diverse ambientazioni, inclusa una mappa principale, la casa del mercante e un pericoloso dungeon. Il giocatore ha il compito di esplorare queste tre diverse location e affrontare una serie di mostri pericolosi, interagire con NPC (personaggi non giocanti) e acquistare oggetti indispensabili dal mercato per migliorare le proprie capacità.
La mappa di gioco presenta una varietà di ostacoli che impediscono l'accesso a determinate zone. Tuttavia, il giocatore ha la possibilità di rompere questi ostacoli utilizzando attrezzi speciali che può ottenere nel corso del gioco. Questo richiede una pianificazione strategica e l'uso oculato delle risorse disponibili.
Il culmine dell'avventura si raggiunge quando il giocatore affronta il mostro finale nel dungeon. Una volta sconfitto, il gioco si considera completato con successo. Tuttavia, lungo il percorso, il giocatore potrebbe trovare ostacoli che bloccheranno il suo percorso.

2 Pacchetto main

2.1 Main.java

Il main è il punto d'ingresso di un'applicazione Java con interfaccia grafica (GUI).

2.1.1 public static void main(String[] args)

Crea una finestra *"JFrame"* chiamata window è aggiunge le proprietà di *"GamePanel"* ad essa, la finestra è impostata in modo che l'applicazione termina la sua esecuzione quando la finestra viene chiusa, disabilita di ridimensionare la finestra *'window.setResizable(false);'*, crea un instanza di *GamePanel* chiamata *gamepanel* che permette di accedere e manipolare l'oggetto *'GamePanel'* nel resto del codice.
Successivamente mediante il metodo *add(Component comp)* della classe *JFrame* si aggiunge il componente grafico alla finestra *window*, il programma carica il file di configurazione dal file *config.txt* utilizzando il metodo loadConfig(), ridimensoina la finestra in base ai suoi compomenti, la rende visibile tramite *'window.setVisible(true);'* e infine chiama i metodi *setupGame()* e *startGameThread()* per avviare il thread di gioco.

2.2 AssettSetter.java

La classe è usata per impostare nella mappa di gioco oggetti di gioco, NPC, mostri e infine case interattive. Viene dichiarata una variabile di istanza gp di tipo GamePanel e crea una connessione tra le due classi in modo da poter manipolare l'oggetto GamePanel tramite la variabile gp.
Gli oggetti di gioco, gli NPC, i mostri etc. estendono tutti la classe *Entity* ereditando quindi tutti i suoi membri.

2.2.1 Costruttore

Si assegna il valore del parametro gp alla variabile di istanza gp della classe AssetSetter.java, quindi si consente alla classe di memorizzare e accedere all'oggetto GamePanel attraverso la variabile gp.

2.2.2 metodo setObject

Nel metodo setObject() vengono assegnati nuovi oggetti in un array bidimensionale 'obj' definito nella classe GamePanel come array pubblico di Entità e successivamente viene definita la posizione nella mappa tramite l'assegnazione di un valore alla variabile 'worldX' e 'worldY' dell'oggetto 'obj' specificati dagli indici 'mapNum' e 'i'. I nuovi oggetti vengono creati utilizzando i propri costuttori delle rispettive classi che richiedono un oggetto di tipo gp.

2.2.3 metodo setNPC

Nel metodo setNPC() vengono assegnati nuovi NPC in un array bidimensionale 'npc' definito nella classe GamePanel come array pubblico di Entità e successivamente viene definita la posizione nella mappa tramite l'assegnazione di un valore alla variabile 'worldX' e 'worldY' dell'oggetto 'npc' specificati dagli indici 'mapNum' e 'i'. I nuovi NPC vengono creati utilizzando i propri costuttori delle rispettive classi che richiedono un oggetto di tipo gp.

2.2.4 metodo setBoss

Il metodo posiziona il Boss del gioco nella mappa, in modo che anche quando il metodo setMonster viene richiamato in update() per rigenerare i mostri presenti sulla mappa non resetti il Boss.

2.2.5 metodo setMonster

Il metodo inizializza, posizionandoli quindi nella mappa, i mostri del gioco nelle diverse aree. Si dichiarano le variabili mapNum e il contatore i che indicano rispettivamente la mappa dove sono posizionati i mostri e l'indice dell'array. Le coordinate sono specificate moltiplicando gp.tileSize ovvero la grandezza di una casella per il numero desiderato nelle coordinate worldX e worldY.

2.2.6 metodo setInteractiveTile

Nel metodo setInteractiveTile() vengono assegnati nuove case interattive nell'array bidimensionale 'iTile' definito nella classe GamePanel come array pubblico di Entità ma a differenza degli altri metodi passiamo direttamente al costruttore della classe 'col' e 'row' per posizionare l'oggetto nella mappa di gioco.

2.3 CollisionChecker.java

La classe è utilizzata per gestire il contatto di Entità con case solide, Entità con Entità, Player con case, Player con oggetti ogni oggetto in gioco ha una area solida inizializzta come pubblica nella classe *Entity* specificata utilizzando la classe *Rectangle*. La classe *Rectangle* fa parte del package java.awt e rappresenta un rettangolo nel sistema di coordinate 2D. Il rettangolo viene creato utilizzando il costruttore *'Rectangle(int x, int y, int width, int height)'*, che accetta quattro argomenti interi: le coordinate *'x'* e *'y'* dell'angolo superiore sinistro del rettangolo, e la *'width'* (larghezza) e *'height'* (altezza) del rettangolo.

2.3.1 Costruttore

Si assegna il valore del parametro gp alla variabile di istanza gp della classe CollisionChecker.java, quindi si consente alla classe di memorizzare e accedere all'oggetto GamePanel attraverso la variabile gp.

2.3.2 metodo checkTile

Il metodo calcola le coordinate dei lati sinistro e destro dell'oggetto entity rispetto al mondo di gioco utilizzando la posizione dell'oggetto entity, insieme alle coordinate x e larghezza del suo solidArea. Fa lo stesso per i lati superiore e inferiore utilizzando la posizione dell'oggetto entity insieme alle coordinate y e altezza del suo solidArea. Calcola gli indici di colonna e riga delle case chesi trovano nei lati sinistro e destro, e nei lati superiore e inferiore

dell'oggetto entity, dividendo le coordinate per la dimensione della piastrella (gp.tileSize). In base alla direzione dell'oggetto entity (up, down, left o right). Ad esempio, se l'oggetto si muove verso l'alto, sottrae la velocità (entity.speed) dalla coordinata y del lato superiore e calcola gli indici di riga per quella posizione aggiornata. Recupera i numeri delle case corrispondenti ai lati sinistro e destro, e ai lati superiore e inferiore dell'oggetto entity, dalla mappa delle case (gp.tileM.mapTileNum) usando gli indici di riga e colonna calcolati. Verifica se le case corrispondenti ai numeri delle case contengono una collisione (collision == true) nell'oggetto gp.tileM.tile. Se una delle case ha una collisione, imposta entity.collisionOn su true, indicando che si è verificata una collisione. Nell'immagine è raffigurata la collisione della solidArea del giocatore, quadratino in rosso più scuro, con la solidArea dell'albero, quadrato rappresentato in chiaro, per una questione di facilità nei movimenti la parte solida del Player è più piccola del resto in modo che possa passare tranquillamente tra due caselle come una porta ad esempio.

2.3.3 metodo checkObject

Il metodo prende un oggetto di tipo Entity e un valore booleano player, in modo che solo il giocatore possa raccogliere gli oggetti, come argomenti e verifica se l'oggetto entity entra in collisione con altri oggetti nella mappa di gioco.
Inizializza una variabile index con il valore 999, che sarà utilizzato per tenere traccia dell'indice dell'oggetto con cui si verifica una collisione (se player è true). Esegue un ciclo for per ogni elemento nell'array gp.obj[1][1]. Verifica se l'elemento corrente nell'array gp.obj[gp.currentMap][i] non è nullo (cioè, è presente un oggetto in quella posizione), calcola le posizioni delle aree solide dell'oggetto entity e dell'oggetto gp.obj[gp.currentMap][i] aggiungendo le rispettive posizioni nel mondo (worldX e worldY) alle coordinate delle rispettive aree solide (solidArea.x e solidArea.y). In base alla direzione dell'oggetto entity, sposta le coordinate dell'area solida (entity.solidArea.x e entity.solidArea.y) in modo da simulare il movimento dell'oggetto. Verifica se l'area solida di entity interseca l'area solida di gp.obj[gp.currentMap][i] utilizzando il metodo 'intersects' della classe *Rectangle*. Se c'è una collisione e gp.obj[gp.currentMap][i].collision è true, imposta entity.collisionOn su true per indicare una collisione. Se player è true, assegna il valore corrente dell'indice i a index per tener traccia dell'oggetto con cui si verifica la collisione. Ripristina le posizioni dell'area solida di entity e gp.obj[gp.currentMap][i] ai valori predefiniti. Restituisce il valore di index che indica l'indice dell'oggetto con cui si è verificata la collisione (999 se non c'è stata collisione o se player è false).

2.3.4 metodo checkEntity

Il criterio utilizzato in questo metodo è simile al metodo precedente checkEntity tiene traccia di un indice index che rappresenta l'indice dell'oggetto con cui si verifica una collisione. Questo indice viene restituito alla fine del metodo. checkObject restituisce l'indice index solo se player è vero. In

[1] [1] indica che si utilizza il secondo indice dell'array bidimensionale altrimenti avrebbe utilizzato un indice con grandezza inferiore

base all'index ricevuto dal metodo quindi se il contatto è avvenuto tra giocatore e mostro, giocatore e NPC, giocatore e 'proiettile' agirà di conseguenza.

2.3.5 metodo checkPlayer

Il metodo controlla se un oggetto entity di tipo Entity ha un contatto con il giocatore gp.player. Viene dichiarata una variabile booleana contactPlayer con valore iniziale false per indicare se c'è stato un contatto con il giocatore. Le posizioni delle aree solide dell'oggetto entity e del giocatore gp.player vengono calcolate aggiungendo le coordinate delle rispettive aree solide alle coordinate mondiali. In base alla direzione dell'oggetto entity, le coordinate dell'area solida vengono spostate di conseguenza utilizzando lo statement switch. Viene eseguito un controllo di intersezione tra le due aree solide utilizzando il metodo intersects della classe 'Rectangle'. Se le due aree si intersecano, significa che c'è un contatto. Se c'è stato un contatto, viene impostato il flag collisionOn dell'oggetto entity su true e la variabile contactPlayer su true. Alla fine, le posizioni delle aree solide vengono ripristinate ai valori predefiniti. Infine, il metodo restituisce il valore della variabile contactPlayer, che indicherà se c'è stato un contatto con il giocatore.
Il seguente metodo è utilizzato ad esempio per gestire i danni sul giocatore da parte dei mostri.

2.4 Config.java

La classe gestisce il di configurazione che contiene i valori delle preferenze del giocatore: volume musica, volume effetti audio. Il file è un file di testo chiamato "config.txt"

2.4.1 Costruttore

Si assegna il valore del parametro gp alla variabile di istanza gp della classe Config.java, quindi si consente alla classe di memorizzare e accedere all'oggetto GamePanel attraverso la variabile gp.

2.4.2 metodo saveConfig

Viene creato un oggetto BufferedWriter per scrivere nel file. Viene utilizzata la classe FileWriter per creare un oggetto che rappresenta il file "config.txt". All'interno del blocco try, vengono scritti i valori per il salvataggio dei dati nel file. 'String.valueOf(gp.music.volumeScale)' converte il valore del volume della musica in una stringa e lo scrive nel file utilizzando il BufferedWriter. Successivamente, viene scritto un carattere di nuova linea utilizzando il metodo newLine del BufferedWriter. Questo assicura che i valori successivi vengano scritti su una nuova riga nel file, lo stesso viene eseguito per 'String.valueOf(gp.se.volumeScale)' in modo da salvare i valori degli effetti audio. Sia 'music' che 'se' sono variabili della classe *Sound.java* instanziate nella classe GamePanel.java. Infine il BufferedWriter viene chiuso. Se si verificano eccezioni durante la lettura o la conversione dei valori, viene stampata la traccia dello stack delle eccezioni.

2.4.3 metodo loadConfig

Il metodo legge dal file "config.txt" e salva le informazioni lette in una stringa s, converte il valore letto in intero tramite 'Integer.parseInt(s)' e li assegna a 'gp.se.volumeScale' e 'gp.music.volumeScale'. Infine il BufferedReader viene chiuso. Se si verificano eccezioni durante la lettura o la conversione dei valori, viene stampata la traccia dello stack delle eccezioni.

2.5 EntityGenerator.java

2.5.1 Costruttore

Al costruttore viene passato GamePanel gp.

2.5.2 metodo getObject

Il metodo accetta una stringa itemName come parametro, che rappresenta il nome dell'oggetto che si desidera generare. All'interno del metodo, viene utilizzata un'istruzione switch per confrontare il nome dell'oggetto con diversi casi. In ogni caso corrispondente al nome dell'oggetto, viene creato un nuovo oggetto di tipo Entity corrispondente utilizzando il costruttore appropriato e viene assegnato alla variabile obj.

2.6 EventHandler.java

La classe EventHandler sembra essere responsabile per la gestione degli eventi nel gioco.
Nella classe vengono dichiarate variabili come GamePanel, eventRect[][][] matrice tridimensionale per la gestione della posizione degli eventi da attivare dove il primo indice si riferisce alla mappa dove accade l'evento il secondo indice la colonna e il terzo la riga.

2.6.1 Costruttore

Nel costruttore della classe viene quindi creato l'array tridimensionale eventRect, che ha dimensioni gp.maxMap, gp.maxWorldCol e gp.maxWorldRow. Questo array viene utilizzato per memorizzare gli oggetti EventRect che rappresentano i rettangoli degli eventi nel gioco.

2.6.2 metodo setDialogue

Il metodo viene utilizzato per inizializzare una struttura di dialoghi all'interno del gioco, in modo che possano essere utilizzati successivamente durante l'esecuzione del gioco.

2.6.3 metodo checkEvent

Il metodo viene utilizzato per controllare se si verificano eventi all'interno del gioco. Calcola la distanza tra la posizione corrente del giocatore (gp.player.worldX e gp.player.worldY) e le coordinate precedenti dell'evento (previousEventX e previousEventY). La distanza viene calcolata sia lungo l'asse x che lungo l'asse y, e viene quindi presa la distanza massima tra le due. Se la distanza supera la dimensione di una singola tile (gp.tileSize), viene impostata la variabile canTouchEvent a true, indicando che è possibile attivare eventi. Successivamente, viene eseguito un blocco di condizioni if-else per controllare se si verificano specifici eventi. Ogni condizione chiama il metodo hit() con determinati parametri e verifica se il risultato è true. Se la condizione è soddisfatta, viene chiamato un metodo specifico per gestire l'evento corrispondente. Ad esempio, se hit(0, colonna, riga, "direzione") restituisce true, viene chiamato il metodo successivo, allo stesso modo, ci sono altre condizioni che controllano le collisioni con specifiche coordinate e direzioni, e chiamano i metodi appropriati per gestire gli eventi corrispondenti.

2.6.4 metodo hit

Il metodo viene utilizzato per capire quando il giocatore tocca una determinata casella in una posizione determinata da mappa, colonna e riga in una direzione specifica, si controlla se l'indice della mappa corrisponde alla mappa. Vengono calcolate le posizioni della solid area del giocatore e dell'evento aggiungendo le rispettive coordinate (worldX e worldY). Vengono calcolate le posizioni dell'oggetto evento specifico moltiplicando le coordinate colonna e riga per la dimensione di una casella 'gp.tileSize'. Viene controllata l'intersezione tra l'area solida del giocatore e l'oggetto evento. Se la collisione avviene e l'evento non è stato ancora completato (eventDone == false), si procede con il controllo della direzione richiesta. Se la direzione richiesta corrisponde alla direzione attuale del giocatore (gp.player.direction.contentEquals(reqDirection)) oppure se la direzione richiesta è "any", la variabile hit viene impostata a true. Vengono memorizzate le coordinate precedenti

dell'evento (previousEventX e previousEventY) con le coordinate correnti del giocatore infine vengono ripristinate le posizioni originali dell'area solida del giocatore e dell'oggetto evento.
I metodi successivi sono eventi che si attivano in base a cosa il giocatore fa.

2.6.5 metodo damagePit

Il metodo gestisce l'evento in cui il giocatore cade in una buca e subisce danni.

2.6.6 metodo healingPit

Il metodo gestisce l'evento in cui il giocatore beve da una sorgente d'acqua e recupera vita. L'evento per essere "attivato" necessita della pressione del tasto Enter da parte del giocatore *if (gp.keyH.enterPressed == true)* dove enterPressed è una variabile booleana che tiene diventa true se Enter è premuto, la pressione dei tasti è gestita nella classe *KeyHandler.java*, se la vita del Player è inferiore alla sua vita massima viene eseguito il blocco di codice all'interno dell'if statement quindi, si cambia lo stato di gioco in dialogueState passato come parametro, si cancella l'animazione dell'attacco dato che sia l'attacco che l'interazione condividono lo stesso tasto, viene mostrato a schermo, tramite una finestra di dialogo creata in un metodo della classe *UI.java*, il testo "Hai bevuto dell'acqua..." e si incrementa la vita attuale del Player di 1.

2.6.7 metodo interactiveSignOne

Il metodo gestisce l'evento dell'interazione con un cartello nella mappa.

2.6.8 metodo interactiveSignTwo

Il metodo gestisce l'evento dell'interazione con un cartello nella mappa.

2.6.9 metodo detectEnterBossZone

Il metodo ferma la musica corrente di gioco e ne avvia una per il particolare evento di Boss Battle che viene attivata nel momento in cui si varca una certa soglia nel dungeon.

2.6.10 metodo teleport

Il metodo gestisce l'evento di spostamento tra le mappe di gioco, i valori map, col e row vengono assegnati alle variabili tempMap, tempCol e tempRow rispettivamente, questi valori rappresentano la destinazione del teletrasporto del giocatore. La variabile canTouchEvent viene impostata su false impedendo al giocatore di attivare altri eventi di tocco durante il teletrasporto.

2.7 EventRect.java

La classe fornisce un modo per definire e gestire rettangoli di evento nel gioco, consentendo di controllare l'interazione del giocatore con determinate aree e il completamento degli eventi associati ad esse.

2.8 GamePanel.java

La classe estende *JPanel* e implementa *Runnable*. Rappresenta il pannello di gioco principale in cui viene visualizzato il gioco. La classe eredita tutte le funzionalità di base fornite dalla classe *JPanel*. *JPanel* è una classe Swing che fornisce un componente per disegnare e gestire elementi grafici all'interno di un contenitore e implementa l'interfaccia *Runnable*, significa che la classe può essere eseguita come un thread separato. L'interfaccia Runnable definisce un singolo metodo *run()*, che contiene il codice che verrà eseguito all'interno del thread. Questo consente di gestire in modo

efficiente le operazioni di rendering e aggiornamento del gioco in un thread separato rispetto al thread principale dell'applicazione.
Nella classe vengono inizializzate delle variabili utili per l'impostazione dello schermo di gioco:

- originalTileSize indica la grandezza originale di una casa di gioco ovvero 16 pixel per 16 pixel;
- scale indica il fattore di scala utilizzato per ingrandire le dimensioni della casa;
- tileSize rappresenta la dimensione finale di una cella ingrandita
- maxScreenCol e maxScreenRow indicano il numero massimo di colonne e righe visibili sullo schermo;
- screenWidth e screenHeight indicano la larghezza e l'altezza totali dello schermo in base alla dimensione del tile e al numero di colonne e righe visibili;

le impostazioni del mondo:

- maxWorldCol e maxWorldRow rappresentano il numero massimo di colonne e righe del mondo di gioco.
- maxMap rappresenta il numero massimo di mappe disponibili nel gioco.
- currentMap indica l'indice della mappa corrente.

gli oggetti di gioco:

- tileM è un'istanza di TileManager che gestisce le case del gioco;
- keyH è un'istanza di KeyHandler che gestisce gli input da tastiera;
- music e se sono istanze di Sound che gestiscono la riproduzione della musica e degli effetti sonori;
- cChecker è un'istanza di CollisionChecker che gestisce la verifica delle collisioni tra gli oggetti nel gioco;
- aSetter è un'istanza di AssetSetter che gestisce il caricamento delle risorse grafiche;
- ui è un'istanza di UI che gestisce l'interfaccia utente;
- eHandler è un'istanza di EventHandler che gestisce gli eventi nel gioco;
- config è un'istanza di Config che gestisce le impostazioni di configurazione del gioco;
- pFinder è un'istanza di PathFinder che gestisce la ricerca del percorso di NPC e mostri nel gioco;
- eManager è un'istanza della classe EnvironmentManager e viene inizializzata passando l'oggetto this come argomento al costruttore. L'oggetto EnvironmentManager gestisce l'ambiente di gioco, inclusi la luce, l'atmosfera e gli effetti ambientali.
- map è un'istanza della classe Map e viene inizializzata passando l'oggetto this come argomento al costruttore. L'oggetto Map rappresenta la mini mappa di gioco e gestisce la generazione e l'aggiornamento della mini mappa.
- saveLoad è un'istanza della classe SaveLoad e viene inizializzata passando l'oggetto this come argomento al costruttore. L'oggetto SaveLoad gestisce il salvataggio e il caricamento dei dati di gioco.

- eGenerator è un'istanza della classe EntityGenerator e viene inizializzata passando l'oggetto this come argomento al costruttore. L'oggetto EntityGenerator gestisce la generazione di entità di gioco come oggetti, NPC e mostri.

variabili per il controllo del gioco:

- gameThread è un thread che esegue il ciclo di gioco;
- player è un'istanza di Player che rappresenta il personaggio controllato dal giocatore;
- obj, npc, mon, iTile e projectile sono matrici di entità che rappresentano gli oggetti, i personaggi non giocanti, i mostri, i tile interattivi e i proiettili nel gioco;
- particleList è una lista di entità che rappresentano le particelle nel gioco;
- entityList è una lista di entità generica che contiene tutte le entità nel gioco;

infine inizializza gli stati di gioco chiamati *gameState* che può assumere i seguenti valori:

- titleState per la schermata dei titoli di gioco;
- playState per lo stato di gioco "attivo";
- pauseState per la schermata di pausa;
- dialogueState per lo stato di visualizzazione delle finestre di dialogo;
- characterState per lo stato di gestione del personaggio quali statistiche e inventario;
- optionsState per la schermata delle opzioni;
- gameOverState per lo stato di fine del gioco;
- transitionState per lo stato di transizione;
- tradeState per la schermata di scambio con il mercante;
- sleepState per la transizione quando si dorme:
- mapState per la visualizzazione della mini mappa;

in base ad ogni gameState è possibile utilizzare determinati tasti definiti nella classe KeyHandler.java e stampare a schermo elementi di gioco basati sullo stato attuale del gioco definiti nella classe *UI.java*.

2.8.1 Costruttore

Viene impostata la grandezza della finestra di gioco utilizzando un oggetto Dimension[2] che riceve come parametri altezza e larghezza di tipo intero quindi screenWidth e screenHeigth calcolati precedentemente, imposta il colore di background su nero, abilita il double buffering sul pannello di gioco[3], agiunge un oggetto KeyListener al pannello di gioco. Il KeyListener viene rappresentato dall'oggetto keyH, che è un'istanza della classe KeyHandler.java. Questo permette al pannello di ascoltare gli eventi di input da tastiera e gestirli tramite il KeyHandler e infine imposta il pannello di gioco come "focusable" ciò significa che il pannello può ricevere gli eventi di input da tastiera quando ha il focus.

[2] Dimension è una classe fornita dal pacchetto java.awt che rappresenta una dimensione in termini di larghezza e altezza. Viene utilizzata per specificare le dimensioni di un componente o di un'area all'interno di un'applicazione grafica.

[3] Il double buffering è una tecnica che consente di ridurre lo sfarfallio (flickering) durante il rendering delle immagini sullo schermo.

2.8.2 metodo resetGame

Il metodo reimposta diversi aspetti del gioco, come la posizione del giocatore, lo stato del giocatore, gli NPC, i mostri, gli oggetti e i tile interattivi. Se restart è true, reimposta anche i valori predefiniti del giocatore e il ciclo giorno-notte del gioco.

2.8.3 metodo setupGame

In sintesi, il metodo setupGame() inizializza il gioco impostando gli oggetti, gli NPC, le entità e le piastrelle interattive. Inoltre, imposta lo stato di gioco iniziale a titleState.

2.8.4 metodo startGameThread

Il metodo viene utilizzato per creare un nuovo thread di gioco e avvia l'esecuzione del metodo run() nella classe GamePanel.

2.8.5 metodo run

Il metodo run() è un metodo essenziale quando si implementa l'interfaccia Runnable in una classe. Contiene il codice che viene eseguito nel thread di gioco. Di seguito sono elencate le azioni eseguite all'interno del metodo:

- double drawInterval = 1000000000/FPS: Viene calcolato l'intervallo di tempo in nanosecondi tra ogni frame di gioco, basato sul valore di FPS (frame per secondo);[4]
- double delta = 0: viene inizializzata la variabile delta che tiene traccia dell'accumulo di tempo tra i frame di gioco;
- long lastTime = System.nanoTime(): viene registrato il tempo corrente in nanosecondi come lastTime. Questo viene utilizzato per calcolare il tempo trascorso tra i frame di gioco;
- long currentTime: viene dichiarata la variabile currentTime per memorizzare il tempo corrente;
- long timer = 0: viene inizializzata la variabile timer che tiene traccia del tempo trascorso per calcolare gli FPS;
- ciclo principale del gioco: il ciclo viene eseguito finché il thread di gioco non è null. All'interno del ciclo:
  - viene calcolato il tempo trascorso tra l'ultimo frame e il frame corrente e viene aggiunto a delta;
  - timer tiene traccia del tempo trascorso dal momento in cui il ciclo è iniziato;
  - se delta raggiunge o supera 1, viene chiamato il metodo update() per aggiornare la logica di gioco e il metodo repaint() per ridisegnare il pannello di gioco;
  - delta viene decrementato di 1 e drawCount viene incrementato di 1;
  - se timer raggiunge o supera 1 secondo (1.000.000.000 nanosecondi), viene stampato il numero di frame disegnati (drawCount) e i contatori vengono reimpostati;

2.8.6 metodo update

Il metodo si occupa di gestire gli aggiornamenti degli elementi del gioco in base allo stato corrente, mantenendo il gioco in movimento e reattivo agli input e alle interazioni.

[4] Il metodo run() controlla l'aggiornamento e il ridisegno del gioco a una frequenza determinata da FPS, mantenendo un intervallo costante tra i frame. In questo modo, il gioco viene eseguito in modo fluido e gli FPS vengono monitorati per misurare le prestazioni. Il gioco è progettato per aggiornarsi 60 volte al secondo.

2.8.7 metodo paintComponent

Nel metodo, prima di tutto viene chiamato il metodo super.paintComponent(g) per effettuare il disegno di base del pannello. Successivamente, viene creato un oggetto Graphics2D g2 da Graphics g per consentire l'utilizzo di funzionalità avanzate di disegno. Nello stato titleState (stato del menu di selezione), viene disegnato il menu tramite il metodo ui.draw(g2).
Negli altri stati di gioco, viene disegnata la mappa di gioco utilizzando l'oggetto tileM Tile Manager e il metodo tileM.draw(g2), vengono disegnate le caselle interattive di gioco attraverso un ciclo for che scorre l'array iTile e chiama il metodo draw(g2) per gli oggetti non nulli, aggiunge gli oggetti di gioco (giocatore, NPC, oggetti, mostri, proiettili, particelle) alla lista entityList definita in GamePanel.java, la entityList viene ordinata in base all'altezza nel mondo degli oggetti,[5] viene eseguito un ciclo for per disegnare gli oggetti presenti nella entityList, chiamando il metodo draw(g2) per ogni oggetto successivamente la entityList viene svuotata, viene disegnata l'interfaccia di gioco tramite il metodo ui.draw(g2).

2.8.8 metodo playMusic

Il metodo avvia la riproduzione di una traccia musicale specificata dall'indice i. La classe music viene utilizzata per impostare il file audio da riprodurre, quindi viene chiamato il metodo play() per avviare la riproduzione e loop() per riprodurre la traccia in loop continuo.

2.8.9 metodo stopMusic

Il metodo interrompe la riproduzione della musica in corso. Chiama il metodo stop() sulla classe music per fermare la riproduzione.

2.8.10 metodo playSE

Il metodo playSE(int i) avvia la riproduzione di un effetto sonoro specificato dall'indice i. La classe se viene utilizzata per impostare il file audio da riprodurre, quindi viene chiamato il metodo play() per avviare la riproduzione dell'effetto sonoro.

2.9 KeyHandler.java

La classe implementa l'interfaccia KeyListener[6] per gestire gli eventi legati alla pressione dei tasti sulla tastiera, sono dichiarati GamePanel e le variabili booleane per la gestione dei tasti premuti quali upPressed, downPressed, leftPressed, rightPressed che indicano se i rispettivi tasti freccia su, giù, sinistra e destra sono stati premuti, enterPressed per il tasto Invio e shotPressed per il tasto Spazio.

2.9.1 Costruttore

Si assegna il valore del parametro gp alla variabile di istanza gp della classe KeyHandler.java, quindi si consente alla classe di memorizzare e accedere all'oggetto GamePanel attraverso la variabile gp.

2.9.2 metodo keyTyped

Generato automaticamente ma non utilizzato.

[5] La chiamata a Collections.sort(entityList, new Comparator<Entity>() ...) ordina la lista entityList in base al valore di worldY delle entità. L'implementazione del metodo compare() nel Comparator confronta due oggetti Entity (e1 e e2) confrontando i loro valori worldY utilizzando il metodo Integer.compare().

[6] KeyListener è un'interfaccia in Java appartenente al package java.awt.event. Questa interfaccia definisce i metodi che devono essere implementati per gestire gli eventi generati dalla pressione dei tasti sulla tastiera.

2.9.3 metodo keyPressed

Il metodo gestisce i comandi di gioco che possono essere utilizzati in base alla stato di gioco attuale ad esempio, se lo stato di gioco è gp.titleState, viene chiamato il metodo titleState() per gestire la pressione del tasto durante lo stato di selezione. In modo simile, vengono chiamati altri metodi come playState(), pauseState(), dialogueState(), characterState(), optionsState(), gameOverState() e tradeState() per gestire i comandi associati ai rispettivi stati di gioco.

2.9.4 metodo titleState

Il metodo viene chiamato quando il gioco è nello stato di selezione del titolo (gp.titleState) e viene premuto un tasto sulla tastiera. Il parametro code ricevuto rappresenta il codice del tasto premuto. Vengono eseguite diverse azioni in base al tasto premuto. Ecco una descrizione di cosa accade per ciascun tasto:

- se il tasto premuto è "W" (KeyEvent.VK_W), viene decrementato il valore della variabile gp.ui.selectedNum, che rappresenta il numero di selezione nell'interfaccia, viene anche riprodotto un effetto sonoro utilizzando il metodo gp.playSE(9)[7]. Se il valore di gp.ui.selectedNum diventa inferiore a 0, viene impostato a 2 per tornare alla selezione più in basso;
- se il tasto premuto è "S" (KeyEvent.VK_S), viene incrementato il valore della variabile gp.ui.selectedNum. Viene riprodotto un effetto sonoro utilizzando il metodo gp.playSE(9). Se il valore di gp.ui.selectedNum diventa superiore a 2, viene impostato a 0 per tornare alla selezione più in alto;
- se il tasto premuto è "Invio" (KeyEvent.VK_ENTER), vengono eseguite azioni in base al valore di gp.ui.selectedNum:
  - se 0, lo stato di gioco viene impostato a "stato di gioco attivo" e viene riprodotta la musica di gioco;
  - se 1, carica uan partita salvata;
  - se 2, viene terminato il programma chiamando System.exit(0).

2.9.5 metodo playState

Il metodo gestisce gli eventi di pressione dei tasti durante lo stato di gioco attivo (playState). A seconda del codice del tasto premuto, vengono eseguite diverse azioni:

- "W" (KeyEvent.VK_W), la variabile upPressed viene impostata su true;
- "S" (KeyEvent.VK_S), la variabile downPressed viene impostata su true;
- "A" (KeyEvent.VK_A), la variabile leftPressed viene impostata su true;
- "D" (KeyEvent.VK_D), la variabile rightPressed viene impostata su true;
- "P" (KeyEvent.VK_P), lo stato di gioco viene impostato come stato di pausa (pauseState);
- "Enter" (KeyEvent.VK_ENTER), la variabile enterPressed viene impostata su true;
- "E" (KeyEvent.VK_E), lo stato di gioco viene impostato su (characterState) e si resetta la posizione del cursore dell'inventario gestito nella classe UI.java.

[7] 9 è l'intero ricevuto come parametro per selezionare il file audio da riprodurre in questo caso si riferisce all'8 effetto audio dichiarato nella classe Sound.java, gp.playSE(9) è un suono che indica lo spostamento di un cursore.

- "Spazio" (KeyEvent.VK_SPACE), la variabile shotPressed viene impostata su true;
- "O" (KeyEvent.VK_O), lo stato di gioco viene impostato come stato delle opzioni (optionsState);
- "M" (KeyEvent.VK_M), lo stato di gioco viene impostato come stato della mappa (mapState);

2.9.6 metodo pauseState

Il metodo alla pressione del tasto "P" reimposta lo stato di gioco da pausa a play.

2.9.7 metodo dialogueState

Il metodo alla pressione del tasto "Invio" passa al dialogo successivo.

2.9.8 metodo characterState

Il metodo gestisce gli eventi di pressione dei tasti durante lo stato del personaggio (characterState). A seconda del codice del tasto premuto, vengono eseguite diverse azioni:

- se il tasto premuto è il tasto "E" (KeyEvent.VK_E), lo stato di gioco viene impostato come stato di gioco attivo (playState), tornando al controllo del personaggio;
- se il tasto premuto è il tasto "Enter" (KeyEvent.VK_ENTER), viene chiamato il metodo selectItem() dell'oggetto gp.player, che seleziona l'oggetto corrente nel inventario del giocatore.

2.9.9 metodo playerInventoryControl e traderInventoryControl

I due metodi forniscono ulteriori azioni alla pressione di determinati tasti rispettivamente per il controllo dell'inventario del Player e del mercante.

2.9.10 metodo optionsState

Il metodo gestisce gli eventi di pressione dei tasti durante lo stato delle opzioni:

- "O" (KeyEvent.VK_O), il gioco torna allo stato di gioco attivo (playState).
- "ENTER" (KeyEvent.VK_ENTER), la variabile enterPressed viene impostata su true e seleziona quell'opzione.

Viene inizializzata una variabile maxSelectedNum in base al valore di gp.ui.subState[8]. Questa variabile determina il numero massimo di opzioni selezionabili. All'interno del menù opzioni alla pressione di:

- "W" (KeyEvent.VK_W), il numero di opzione selezionato viene decrementato, se il numero di opzione diventa inferiore a 0, viene impostato a maxSelectedNum.
- "S" (KeyEvent.VK_S), il numero di opzione selezionato viene incrementato, se il numero di opzione diventa superiore a maxSelectedNum, viene impostato a 0.
- "A" (KeyEvent.VK_A) e gp.ui.subState è 0, vengono eseguite ulteriori azioni in base al numero di opzione selezionato:
  - se il numero di opzione selezionato è 0 e il volume della musica (gp.music.volumeScale) è maggiore di 0, il volume della musica viene decrementato;
  - se il numero di opzione selezionato è 1 e il volume degli effetti sonori (gp.se.volumeScale) è maggiore di 0, il volume degli effetti sonori viene decrementato.

[8] subState è una variabile di tipo int il cui scopo è quello di selezionare sottomenu differenti in base alla scelta del giocatore

2.9.11 metodo gameOverState

Il metodo gestisce gli eventi di pressione dei tasti durante lo stato di game over quindi dopo che la vita del Player è 0. A seconda del codice del tasto premuto, vengono eseguite diverse azioni:

- "W" (KeyEvent.VK_W), il numero di opzione selezionato viene decrementato. Se il numero di opzione diventa inferiore a 0, viene impostato a 0;
- "S" (KeyEvent.VK_S), il numero di opzione selezionato viene incrementato. Se il numero di opzione diventa superiore a 1, viene impostato a 1;
- "ENTER" (KeyEvent.VK_ENTER), viene eseguito un controllo sul numero di opzione selezionato:
  - se il numero di opzione selezionato è 0, il gioco passa allo stato di gioco attivo (playState), viene chiamato il metodo resetGame(), e viene riprodotta la musica di gioco;
  - se il numero di opzione selezionato è 1, il gioco passa allo stato di titolo (titleState), viene chiamato il metodo resetGame(), e viene riprodotta la musica di gioco.

2.9.12 metodo tradeState

l metodo gestisce gli eventi di pressione dei tasti durante lo stato di scambio con il mercante. A seconda del codice del tasto premuto e dello stato di sottoselezione (subState), vengono eseguite diverse azioni:

- "ENTER" (KeyEvent.VK_ENTER), viene impostato il flag enterPressed a true, utilizzato per selezionare i diversi (subState);
- se lo stato di sottoselezione è 0, vengono gestiti i comandi relativi alla selezione delle opzioni di scambio:
  - "W" (KeyEvent.VK_W), il numero di opzione selezionato viene decrementato. Se il numero di opzione diventa inferiore a 0, viene impostato a 0;
  - "S" (KeyEvent.VK_S), il numero di opzione selezionato viene incrementato. Se il numero di opzione diventa superiore a 2, viene impostato a 2.
- se lo stato di sottoselezione è 1, è possibile utilizzare i comandi relativi al controllo dell'inventario del commerciante (traderInventoryControl(code)). Inoltre, se il tasto premuto è il tasto "ESC" (KeyEvent.VK_ESCAPE), lo stato di sottoselezione viene impostato a 0;
- se lo stato di sottoselezione è 2, vè possibile utilizzare i comandi relativi al controllo dell'inventario del giocatore (playerInventoryControl(code)). Inoltre, se il tasto premuto è il tasto "ESC" (KeyEvent.VK_ESCAPE), lo stato di sottoselezione viene impostato a 0.

2.9.13 metodo mapState

Il metodo viene chiamato quando il gioco è nello stato di visualizzazione della mini mappa di gioco, alla pressione del tasto "M" il gioco ritorna allo stato attivo playState.

2.9.14 metodo keyReleased

Il metodo gestisce gli eventi di rilascio dei tasti. Viene controllato il codice del tasto rilasciato e, a seconda di esso, vengono impostati i flag corrispondenti a false per indicare che il tasto non è più premuto. Questo permette di monitorare lo stato dei tasti e reagire adeguatamente quando vengono premuti o rilasciati durante il gioco.

2.10 Sound.java

La classe gestisce la riproduzione dei suoni nel gioco. Viene creato un oggetto clip,[9] URL soundURL[]: È un array di oggetti URL che contiene gli indirizzi dei file audio utilizzati nel gioco,[10] FloatControl fc è un oggetto FloatControl utilizzato per controllare il volume del suono[11]. Infine le variabili volumeScale di tipo int indica il livello attuale del volume, impostato di default a 3, e la variabile volume di tipo float rappresenta il valore effettivo del volume che varia da -80f (il minimo) a 6f (il massimo).

2.10.1 Costruttore

In questo costruttore, vengono inizializzate le risorse audio (soundURL) utilizzate per riprodurre i vari suoni nel gioco. Le righe di codice assegnano alle variabili soundURL le risorse audio corrispondenti ai file WAV specificati. Ad esempio, soundURL[0] viene impostato con la risorsa audio del file "MainTheme.wav", soundURL[1] con il file "coin.wav", soundURL[2] con il file "powerup.wav", e così via. Una volta che le risorse audio sono state caricate, possono essere utilizzate per creare istanze di oggetti Clip e riprodurre i suoni desiderati nel gioco.

2.10.2 metodo setFile

Il metodo viene utilizzato per impostare il file audio da riprodurre nell'oggetto clip della classe Sound. Il metodo prende come parametro un indice i che rappresenta la posizione del file audio nell'array soundURL. ramite il metodo AudioSystem.getAudioInputStream(soundURL[i]), che carica effettivamente il file audio dalla risorsa specificata si ottiene un oggetto 'ais' AudioInputStream, viene creato un oggetto Clip tramite AudioSystem.getClip() e viene aperto il file audio tramite il metodo clip.open(ais). Infine per verificare e impostare il volume del clip in base al valore corrente di volumeScale viene chiamato il metodo checkVolume() .

2.10.3 metodo play

Il metodo play() avvia la riproduzione del file audio chiamando il metodo clip.start(). Questo farà sì che il file audio venga riprodotto dall'inizio.

2.10.4 metodo loop

Il metodo loop() imposta il file audio in modalità di riproduzione continua, utilizzando il valore Clip.LOOP_CONTINUOUSLY come parametro del metodo clip.loop(). Questo farà sì che il file audio venga riprodotto in loop finché non viene chiamato il metodo stop().

2.10.5 metodo stop

Infine, il metodo stop() interrompe la riproduzione del file audio chiamando il metodo clip.stop(). La riproduzione può essere successivamente ripresa chiamando il metodo play().

2.10.6 metodo checkVolume

Il metodo assegna un valore a float volume in base al valore di volumeScale.

[9] Per utilizzare la classe Clip, è necessario importare il pacchetto javax.sound.sampled, che contiene le classi e le interfacce per la gestione dei suoni campionati.

[10] URL è una classe definita nell'API di Java che rappresenta un Uniform Resource Locator, ovvero un indirizzo di una risorsa specifica, come ad esempio un file, una pagina web o un servizio web.

[11] Un oggetto FloatControl rappresenta un controllo di tipo float che può essere applicato a un componente audio, come ad esempio il volume, il bilanciamento o la velocità di riproduzione. È possibile utilizzare un oggetto FloatControl per ottenere e impostare il valore di una caratteristica float specifica di un componente audio.

2.11 UI.java

La classe è responsabile per la gestione dell'interfaccia utente nel contesto del gioco. Ecco una spiegazione delle sue variabili e metodi principali:

- gp:variabile di tipo GamePanel che rappresenta il pannello di gioco a cui è associata l'interfaccia utente;
- g2: oggetto Graphics2D utilizzato per disegnare elementi grafici sull'interfaccia utente;
- mainFont: font utilizzato per il testo nell'interfaccia utente in questo caso è stato utilizzato come font "VCR_OSD_MONO_1.001.ttf";
- heart_full, heart_half, heart_blank, mana_full, mana_blank, coin: Immagini utilizzate per rappresentare graficamente cuori, cristalli di mana e monete nell'interfaccia utente;
- message : lista di messaggi da mostrare graficamente;
- messageCounter: lista dei contatori associati ai messaggi nell'interfaccia utente;
- subState: indica i diversi sotto menu selezionabili in base allo stato di gioco se il metodo in questione ne gestisce alcuni.

Infine si impostano le variabili per il posizionamento dei cursori per l'inventario del Player e del mercante a 0, 0.

2.11.1 Costruttore

Inizializza le risorse grafiche necessarie per l'interfaccia utente, come font e immagini, in modo che possano essere utilizzate successivamente per disegnare l'HUD (Heads-Up Display) nel gioco:

- InputStream is = getClass().getResourceAsStream("/fonts/VCR_OSD_MONO_1.001.ttf");: legge il file del font specificato (VCR_OSD_MONO_1.001.ttf) come stream di input, la risorsa del font viene caricata dalla directory "fonts" all'interno del progetto;
- mainFont = Font.createFont(Font.TRUETYPE_FONT, is);: crea un oggetto Font utilizzando il file del font letto nell'input stream. La costante Font.TRUETYPE_FONT specifica che il file del font è di tipo TrueType.[12]
- Entity heart = new OBJ_Heart(gp);: Questa riga crea un oggetto Entity di tipo "heart" utilizzando la classe OBJ_Heart. Questa classe sembra essere responsabile di generare le immagini dei cuori per l'HUD.
- heart_full = heart.image;: Questa riga assegna l'immagine del cuore pieno all'attributo heart_full dell'oggetto UI.
- heart_half = heart.image2;: Questa riga assegna l'immagine del cuore a metà all'attributo heart_half dell'oggetto UI.
- heart_blank = heart.image3;: Questa riga assegna l'immagine del cuore vuoto all'attributo heart_blank dell'oggetto UI.
- Simili alle righe sopra, le righe successive creano e assegnano le immagini dei cristalli di mana e delle monete.

[12] Un font TrueType è un formato di font flessibile, scalabile e ampiamente compatibile che consente di ottenere testo di alta qualità su schermi e dispositivi di diverse risoluzioni.

2.11.2 metodo addMessage

Il metodo aggiunge un messaggio alla lista dei messaggi. Prende in input una stringa text che rappresenta il testo del messaggio da aggiungere. Il metodo crea un nuovo messaggio nella lista message e inizializza il suo contatore a 0 nella lista messageCounter.

2.11.3 metodo draw

Inizialmente, il metodo imposta il riferimento g2 al parametro fornito, consentendo di utilizzare il contesto grafico all'interno del metodo. Successivamente, il metodo controlla lo stato del gioco (gp.gameState) e disegna gli elementi corrispondenti all'interno delle rispettive condizioni:

- gp.titleState: disegna la schermata dei titoli;
- gp.playState: disegna la barra della vita del giocatore e i messaggi;
- gp.pauseState: disegna la schermata di pausa;
- gp.dialogueState: disegna la schermata dei dialoghi;
- gp.characterState: disegna la schermata delle statistiche del giocatore e dell'inventario;
- gp.optionsState: disegna la schermata delle opzioni di gioco;
- gp.gameOverState: disegna la schermata di game over;
- gp.transitionState: disegna l'effetto di transizione tra le mappe;
- gp.tradeState: disegna la schermata di scambio con il mercante;
- gp.sleepState: disegna l'effetto di transizione tra la notte e il giorno se si dorme.

Ogni metodo di disegno corrispondente a uno specifico stato viene chiamato all'interno delle condizioni per gestire il disegno degli elementi corrispondenti a quello stato specifico.

2.11.4 metodo drawPlayerLife

Il metodo drawPlayerLife viene utilizzato per disegnare la rappresentazione grafica della vita e del mana del giocatore nell'interfaccia utente. All'inizio del metodo, vengono inizializzate le variabili x e y con le coordinate iniziali in cui verranno disegnati gli elementi grafici. La variabile i viene inizializzata a 0. Viene eseguito un ciclo while per disegnare i cuori che rappresentano la vita massima disponibile del giocatore rappresentata dall'immagine heart_blank (cuore vuoto) nella posizione (x, y) definita. Nel ciclo successivo dopo aver resettato posizione e contatore si disegna con heart_half e heart_full la vita del giocatore (mezzo cuore indica 1 di vita, quindi 3 cuori pieni indicano 6 di vita). Dopo aver disegnato la vita del giocatore, vengono disegnati i cristalli di mana. Vengono eseguiti due cicli while simili a quelli precedenti per disegnare i cristalli di mana vuoti e pieni corrispondenti alla mana massima e alla mana attuale del giocatore.

2.11.5 metodo drawMonsterLife

Il metodo drawMonsterLife() disegna le barre della vita dei mostri. Per ogni mostro presente nella matrice mon nella mappa corrente, controlla se il mostro è visibile nella telecamera. Se il mostro ha la barra della vita attiva (e non è un boss), viene calcolata la dimensione della barra della vita in base alla vita attuale del mostro rispetto alla sua vita massima. Viene quindi disegnato un rettangolo nero come sfondo della barra della vita e un rettangolo rosso rappresentante la vita attuale del mostro. Inoltre, viene tenuto conto del tempo trascorso per la visualizzazione della barra della vita dei mostri e, se supera un certo limite, la barra viene disattivata. Se il mostro è un boss, viene disegnata una barra della vita più grande nella parte superiore dello schermo, insieme al nome del mostro.

2.11.6 metodo drawMessage

Il metodo disegna i messaggi sulla schermata di gioco. I messaggi vengono mostrati in sequenza con un effetto di scorrimento verso il basso e vengono rimossi dopo un certo periodo di tempo sono inoltre cancellati dalle 2 liste. Questo consente di visualizzare i messaggi temporanei come informazioni di gioco.

2.11.7 metodo drawTitleScreen

Il metodo è utilizzato per disegnare la schermata dei titoli. All'inizio del metodo, viene impostato il colore di sfondo utilizzando l'oggetto Color con i valori RGB (131, 209, 53, 210) e viene riempito un rettangolo con le dimensioni della schermata. Viene impostata la dimensione del carattere del testo a 80 punti e il tipo di carattere viene impostato in grassetto, viene quindi scritta una stringa di testo "Adventure Game" e vengono calcolate le coordinate (x, y) per posizionare il testo al centro della schermata utilizzando il metodo getTextX. Successivamente, vengono disegnate le ombre del testo utilizzando il colore nero come ombra e viene disegnato il testo principale utilizzando il colore bianco. Viene quindi calcolata la posizione (x, y) per disegnare l'immagine del personaggio al centro della schermata utilizzando le dimensioni del tile. Successivamente, vengono disegnate le opzioni del menu. Per ogni opzione, viene impostata la dimensione del carattere del testo a 30 punti e il tipo di carattere viene impostato in grassetto. Vengono calcolate le coordinate (x, y) per posizionare il testo in base alla posizione precedente. Vengono disegnate le ombre e il testo delle opzioni utilizzando i colori nero e bianco. Se l'opzione corrente è quella selezionata (selectedNum), viene disegnato un simbolo ">" per evidenziare l'opzione.

2.11.8 metodo drawPauseScreen

Il metodo disegna la schermata di pausa del gioco. Inizialmente, viene definito il testo da visualizzare, che è "PAUSA", viene impostato il font per il testo utilizzando il metodo deriveFont() della classe Font. Viene specificato un font di dimensione 50 con stile plain (senza grassetto o corsivo). La posizione del testo viene calcolata utilizzando il metodo getTextX() per ottenere la coordinata x in base alla larghezza del testo mentre per la coordinata y viene impostata a metà dell'altezza dello schermo. Il colore del testo viene impostato su nero per ottenere un effetto ombra e successivamente su bianco spostandolo in diagonale di 3 pixel.

2.11.9 metodo drawDialogueScreen

Il metodo viene utilizzato per disegnare la finestra di dialogo durante una conversazione nel gioco. Vengono definiti i valori per le coordinate x e y, e le dimensioni width e height della finestra di dialogo. Questi valori sono calcolati in base alla dimensione del tile (gp.tileSize) e vengono utilizzati per posizionare e dimensionare correttamente la finestra di dialogo. Viene chiamato il metodo drawSubWindow() per disegnare una sottofinestra con le coordinate e le dimensioni specificate. Viene impostato il font del testo utilizzando il metodo deriveFont() per ottenere una versione con uno stile specifico (in questo caso Font.PLAIN) e una dimensione di 25F. Le coordinate x e y vengono aggiornate per posizionare il testo all'interno della finestra di dialogo. Viene assegnato il valore del dialogo corrente (currentDialogue) utilizzando il dialogo specifico dal set di dialoghi dell'entità (npc.dialogues) e l'indice corrente del dialogo (npc.dialogueIndex). Se il tasto "Enter" (gp.keyH.enterPressed) viene premuto durante lo stato di dialogo (gp.gameState == gp.dialogueState), viene incrementato l'indice del dialogo corrente (npc.dialogueIndex) e il flag del tasto "Enter" viene impostato su false per evitare il rilevamento continuo dell'input. l testo del dialogo viene diviso in righe separate utilizzando il carattere di nuova linea (\n) come delimitatore. Per ogni riga, il testo viene disegnato sulla finestra di dialogo utilizzando il metodo drawString() con le coordinate x e y. Dopo ogni riga, la coordinata y viene incrementata di 40 per posizionare la riga successiva correttamente.

2.11.10 metodo drawCharacterScreen

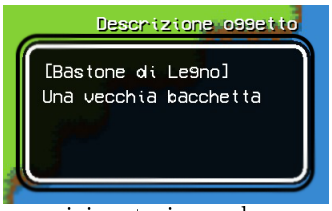
Il metodo disegna la schermata delle statistiche del giocatore. Viene definita la posizione e le dimensioni della finestra delle statistiche del giocatore utilizzando le variabili frameX, frameY, frameWidth e frameHeight. Successivamente, viene chiamato il metodo drawSubWindow() per disegnare la finestra delle statistiche. Vengono impostati il colore e il font per il testo utilizzando i metodi setColor() e setFont() della classe Graphics2D. Viene utilizzato un font di dimensione 20 con stile plain (senza grassetto o corsivo). Vengono definiti le coordinate textX e textY per posizionare il testo all'interno della finestra delle statistiche. La variabile lineHeight rappresenta l'altezza di ogni riga di testo. Successivamente, viene utilizzato il metodo drawString() per disegnare il testo delle statistiche del giocatore. Ogni riga di testo rappresenta una specifica statistica (come il livello, la vita, il mana, la forza, ecc.). Il testo viene disegnato alle coordinate (textX, textY). Dopo ogni riga di testo, la coordinata textY viene incrementata di lineHeight per spostare il testo alla riga successiva. Per allineare i valori delle statistiche a destra, viene calcolata la coordinata textX utilizzando il metodo getTextXtoRight() che restituisce la posizione x corretta per allineare il testo a destra rispetto al valore. Infine, vengono disegnate le immagini dell'arma, dello scudo e dell'attrezzo attualmente in uso dal giocatore utilizzando il metodo drawImage() della classe Graphics2D. Le immagini vengono posizionate alle coordinate appropriate all'interno della finestra delle statistiche.

2.11.11 metodo drawInventoryScreen

Il metodo disegna la schermata dell'inventario per un determinato personaggio (entity). La schermata può essere per il giocatore (gp.player) o per un mercante, riceve in come parametro l'entità e se mostrare o meno il cursore in base alla variabile booleana cursor che può essere true o false. Vengono definite le variabili per le coordinate del frame dell'inventario (frameX, frameY, frameWidth, frameHeight) e le variabili per le coordinate dei singoli slot dell'inventario (slotXstart, slotYstart, slotX, slotY, slotSize), se l'entità è il giocatore (gp.player), vengono impostate le coordinate per la finestra dell'inventario del giocatore e i contatori dei slot (slotColPlayer, slotRowPlayer) altrimenti, se l'entità è un mercante, vengono impostate le coordinate per la finestra dell'inventario del mercante e i contatori dei slot (slotColTrader, slotRowTrader). Viene disegnato il frame dell'inventario chiamando il metodo drawSubWindow con le coordinate calcolate. Vengono disegnati gli oggetti presenti nell'inventario dell'entità e:

- viene disegnata l'immagine dell'oggetto nell'oggetto nell'ordine in cui sono presenti nell'inventario, per ogni oggetto nell'inventario, viene controllato se l'oggetto corrisponde a quello attualmente equipaggiato (arma, scudo o attrezzo) dell'entità gli oggetti equipaggiati sono segnati in rosso;
- la variabile slotX tiene traccia della coordinata x corrente per il disegno degli oggetti, mentre slotY tiene traccia della coordinata y corrente. Quando si raggiunge la fine di una riga di oggetti, le coordinate vengono riportate all'inizio della riga successiva.

Se il parametro cursor è true, viene disegnato il cursore sull'oggetto selezionato nell'inventario ovvero un rettangolo con spessore 3 attorno all'oggetto selezionato per evidenziarlo. Più in basso viene disegnata la finestra di descrizione dell'oggetto selezionato.Le variabili dFrameX, dFrameY, dFrameWidth e dFrameHeight definiscono le coordinate e le dimensioni della finestra di descrizione.



La variabile textX indica la coordinata x del testo di descrizione, mentre textY indica la coordinata y. Viene impostato il font per il testo di descrizione, viene chiamato il metodo drawSubWindow per disegnare il frame di descrizione, viene ottenuto l'indice dell'oggetto selezionato utilizzando il metodo getItemIndex, se l'indice è valido (inferiore alla dimensione dell'inventario), vengono disegnate le linee di descrizione dell'oggetto nella finestra di descrizione. Ogni linea viene posizionata in modo incrementale in base alla coordinata y.

2.11.12 metodo drawGameOverScreen

l metodo disegna la schermata di Game Over. Viene impostato il colore di sfondo dell'intera schermata con un colore nero trasparente e definite le variabili x e y per le coordinate correnti e text per il testo da disegnare. Viene impostato il font a 48. Viene disegnato il testo "GAME OVER" sia in nero (con una leggera ombreggiatura) che in bianco per creare un effetto visivo. Viene disegnato un rettangolo nero sottolineando il testo del titolo. Viene disegnata un'immagine del personaggio morto al centro dello schermo. Viene impostato il font a 28 punti per il testo "RIPROVA" e "INDIETRO" in base all'opzione selezionata a sinistra dei 2 testi viene disegnato il simbolo ">" in base al selectedNum scelto si otterranno risultati differenti.

2.11.13 metodo drawTransition

Il metodo gestisce l'effetto di transizione rendendo gradualmente lo schermo nero e poi ripristinando lo stato di gioco con la nuova mappa e le nuove coordinate del giocatore.

2.11.14 metodo drawTradeScreen

Il metodo gestisce tramite uno switch i 3 subState per comprare e vendere dal commerciante.

2.11.15 metodo trade_select

Vengono definiti i valori delle variabili x, y, height e width per posizionare la finestra di selezione.

Dopodiché, vengono impostate le coordinate x e y per posizionare il testo "Compra" all'interno della finestra. In base al valore di selectedNum viene disegnato il simbolo ">" per indicare la selezione. Se il tasto Invio (enterPressed) è premuto, viene impostato subState a 1 per passare alla schermata di acquisto. Lo stesso procedimento è applicato per i testi "Vendi" e "Esci". Se il tasto Invio è premuto, viene ripristinato selectedNum a 0, viene impostato lo stato di gioco (gp.gameState) a gp.dialogueState, viene assegnata una stringa di dialogo corrente (currentDialogue) e viene chiamato il metodo drawDialogueScreen per visualizzare la schermata di dialogo.

2.11.16 metodo trade_buy

Il metodo gestisce la schermata di acquisto durante il commercio. Innanzitutto, vengono chiamati i metodi drawInventoryScreen per disegnare lo schermo dell'inventario del giocatore e del mercante, rispettivamente.

Dopodiché, vengono impostate le coordinate x e y, la larghezza e l'altezza per disegnare la finestra dei comandi. All'interno della finestra, viene mostrato il testo "[ESC] Indietro". Subito dopo, vengono definite le coordinate x e y, la larghezza e l'altezza per disegnare la finestra delle monete del giocatore. Viene visualizzata l'immagine delle monete e viene indicato il numero di monete possedute dal giocatore. Successivamente, si verifica se l'indice dell'oggetto selezionato (itemIndex) è inferiore alla dimensione dell'inventario del mercante (npc.inventory.size()). Se questa condizione è verificata, viene disegnata la finestra dei prezzi e

viene mostrata l'immagine delle monete che indica il prezzo dell'oggetto selezionato. Nel caso in cui il tasto Invio (enterPressed) sia premuto, viene controllato se il prezzo dell'oggetto selezionato supera il numero di monete del giocatore e se il giocatore ha spazio nell'inventario per poter contenere l'oggetto, se il giocatore ha abbastanza monete per acquistare l'oggetto selezionato e non ha raggiunto (maxInventorySize) quindi non ha l'inventario pieno potrà acquistare viene quindi sottratto il prezzo dell'oggetto selezionato dalle monete del giocatore (gp.player.coin) e l'oggetto viene aggiunto all'inventario del giocatore (gp.player.inventory). In caso contrario verranno mostrati messaggi di errore per indicare sia la "non sufficienza di monete" sia "l'aver raggiunto il limite di oggetti trasportabili, viene quindi cambiato lo stato di gioco in (gp.dialogueState) e stampati i messaggi nelle box di dialogo. In sintesi, il metodo trade_buy gestisce la schermata di acquisto durante il commercio, mostrando le opzioni di acquisto, i prezzi, le monete del giocatore e gestendo l'aggiunta dell'oggetto selezionato all'inventario del giocatore.

2.11.17 metodo trade_sell

Il metodo si occupa della schermata di vendita durante una negoziazione. Inizialmente, disegna lo schermo dell'inventario del giocatore, mostrando gli oggetti posseduti. Successivamente, vengono posizionate e dimensionate le finestre dei comandi, dei prezzi di vendita e delle monete del giocatore. All'interno di tali finestre, vengono visualizzati il testo "[ESC] Indietro" e il numero di monete possedute dal giocatore e quante monete guadagnerebbe il giocatore se vendesse l'oggetto. Viene quindi calcolato l'indice dell'oggetto selezionato nel suo inventario usando il metodo getItemIndex. Se l'oggetto può essere venduto, viene disegnata la finestra dei prezzi, mostrando il prezzo di vendita dell'oggetto. Se il giocatore preme il tasto "enter", viene verificato se l'oggetto selezionato è equipaggiato, in tal caso viene mostrato un messaggio di avviso e l'oggetto non può essere venduto. Altrimenti, l'oggetto viene rimosso dall'inventario del giocatore e il suo prezzo di vendita viene aggiunto alle monete del giocatore. Alla fine del metodo, la schermata di gioco passa allo stato di dialogo e viene assegnato un messaggio di dialogo corrente che avvisa il giocatore che non può vendere un oggetto equipaggiato. In questo modo, il metodo gestisce l'interazione del giocatore durante la vendita di oggetti in modo fluido e intuitivo.

2.11.18 metodo drawOptionScreen

Il metodo drawOptionsScreen() viene utilizzato per disegnare la schermata delle opzioni di gioco. All'interno del metodo, viene impostato il colore del testo su bianco e la dimensione del carattere su 20. Viene quindi definita la posizione e le dimensioni della finestra delle opzioni, utilizzando valori basati sul valore di gp.tileSize. Successivamente, viene utilizzato uno switch statement per determinare quale sotto-stato delle opzioni deve essere disegnato. Sono gestiti tre sotto-stati:

- nel caso 0, viene chiamato il metodo options_top() per disegnare la finestra principale della schermata delle opzioni;
- nel caso 1, viene chiamato il metodo options_controll() per disegnare la schermata dei comandi del gioco;
- nel caso 2, viene chiamato il metodo options_endGame() per disegnare le opzioni di fine gioco della schermata delle opzioni.

Infine, viene impostato gp.keyH.enterPressed su false per indicare che il tasto Enter non è stato premuto in modo da evitare che parta l'animazione dell'attacco.

2.11.19 metodo options_top

Il metodo viene utilizzato per disegnare le opzioni principali della schermata delle opzioni. Viene quindi disegnato il testo "OPZIONI" centrato in alto. Successivamente, vengono disegnate le opzioni "Musica", "Effetti", "Comandi", "Esci" e "Indietro", con il simbolo ">" posizionato a sinistra dell'opzione selezionata (selectedNum). Se l'opzione "Comandi" viene selezionata e viene premuto il tasto Enter, il sotto-stato viene impostato su 1 e l'opzione selezionata viene ripristinata a 0. Se l'opzione "Esci" viene selezionata e viene premuto il tasto Enter, il sotto-stato viene impostato su 2 e l'opzione selezionata viene ripristinata a 0. Se l'opzione "Indietro" viene selezionata e viene premuto il tasto Enter, lo stato del gioco viene impostato su gp.playState e l'opzione selezionata viene ripristinata a 0. Dal lato opposto della finestra vengono disegnate le barre del volume per la musica e gli effetti. Il rettangolo che indica il volume attuale della musica di gioco e degli effetti sonori è regolato in base al valore di volumeScale. Infine, viene chiamato il metodo saveConfig() per salvare la configurazione delle opzioni.

2.11.20 metodo options_controll

Il metodo viene utilizzato per disegnare le opzioni dei controlli nella schermata delle opzioni. All'interno del metodo, vengono definiti le variabili per le posizioni del testo. Viene quindi disegnato il testo "CONTROLLI" utilizzando la posizione calcolata dal metodo getTextX() e la coordinata Y della finestra. Successivamente, vengono disegnate le opzioni per i comandi di movimento, conferma/attacco, abilità, inventario, pausa e opzioni. I tasti assegnati a ciascun comando vengono disegnati accanto al testo corrispondente. Infine, viene disegnata l'opzione "Indietro" accanto al testo corrispondente. Se l'opzione "Indietro" viene selezionata e viene premuto il tasto Enter, il sotto-stato viene impostato su 0 e l'opzione selezionata viene impostata a 2 ad indicare l'ultimo menu selezionato.

2.11.21 metodo options_endGame

Il metodo viene utilizzato per disegnare le opzioni di chiusura del gioco nella schermata delle opzioni. All'interno del metodo, vengono definiti le variabili per le posizioni del testo. Viene quindi disegnato il testo "Chiudere il gioco?" utilizzando la posizione calcolata dal metodo getTextX() e la coordinata Y della finestra. Successivamente, vengono disegnate le opzioni "SI" e "NO":

- "SI" il gioco viene chiuso chiamando il metodo System.exit(0)[13];
- "NO" si ritorna al menu opzioni principare.

2.11.22 metodo drawSleepScreen

Il metodo gestisce la transizione del gioco durante il periodo di riposo, modificando l'opacità del filtro di illuminazione per creare l'effetto visivo desiderato e ripristinando lo stato di gioco normale al termine della transizione.

2.11.23 metodo drawSubWindow

Il metodo disegna una finestra di sottocategoria con un colore di riempimento scuro e leggermente trasparente e un bordo bianco intorno ad essa. Questo può essere utilizzato per creare effetti visivi e separare visivamente diverse sezioni dell'interfaccia utente. Prende in input le coordinate (x, y) del punto iniziale della finestra, la larghezza e l'altezza della finestra. All'interno del metodo, viene creata una variabile c di tipo Color per rappresentare il colore di riempimento della finestra è usato un colore nero semi-trasparente con i valori (0, 0, 0, 230). Questo significa che la finestra avrà un aspetto scuro e leggermente trasparente. Successivamente, il coloredi g2 viene impostato su c e viene utilizzato il metodo fillRoundRect per riempire un rettangolo arrotondato con le coordinate (x, y), larghezza e altezza specificate, i valori 35 indicano il raggio di curvatura degli angoli del rettangolo. Dopo aver riempito il rettangolo, viene creato un nuovo colore c di colore bianco con i valori (255, 255, 255). Il colore di g2 viene quindi impostato con questo nuovo colore e con uno spessore di linea di 4 utilizzando il metodo setStroke. Questo definisce un bordo bianco con uno spessore di linea di 4 intorno alla finestra. Infine, viene utilizzato il metodo drawRoundRect per disegnare il bordo del rettangolo arrotondato con le coordinate (x + 5, y + 5), larghezza e altezza ridotte rispetto alla finestra di base.

2.11.24 metodo getItemIndex

Il metodo viene utilizzato per calcolare l'indice di un oggetto all'interno dell'inventario, dato la colonna e la riga in cui l'oggetto è posizionato. Prende in input due parametri interi: slotCol rappresenta la colonna dell'oggetto e slotRow rappresenta la riga dell'oggetto. All'interno del metodo, l'indice dell'oggetto viene calcolato moltiplicando la riga per 5 (dato che ci sono 5 colonne in totale) e sommando la colonna. Questo calcolo tiene conto della disposizione degli oggetti nell'inventario e restituisce l'indice corrispondente all'oggetto desiderato. Infine, l'indice viene restituito come risultato del metodo.

2.11.25 metodo getTextX

All'interno del metodo, viene calcolata la lunghezza del testo utilizzando il metodo getStringBounds della classe FontMetrics[14]. Questo metodo restituisce un oggetto Rectangle2D che rappresenta il rettangolo di delimitazione del testo. Viene quindi estratta la larghezza del rettangolo di

[13] 'System.exit(0);' è una chiamata al metodo exit() della classe System che viene utilizzata per terminare l'esecuzione del programma con un codice di uscita specificato.

[14] FontMetrics è una classe in Java che fornisce informazioni sulla metrica dei caratteri di un determinato Font. Essa contiene metodi per ottenere informazioni come larghezza, altezza, spaziatura e altre proprietà dei caratteri all'interno di un Font.

delimitazione utilizzando il metodo getWidth(). Successivamente, viene calcolata la coordinata x per posizionare il testo al centro della schermata. Viene diviso gp.screenWidth per 2 per ottenere il punto medio orizzontale della schermata e sottratta la metà della lunghezza del testo per centrarlo correttamente. Infine, la coordinata x viene restituita come risultato del metodo.

2.11.26 metodo getTextXtoRight

Il metodo restituisce la coordinata x per allineare il testo a destra rispetto a una posizione di riferimento tailX[15]. Viene calcolata la lunghezza del testo utilizzando il metodo getStringBounds() della classe FontMetrics. Successivamente, viene sottratta la lunghezza del testo dalla coordinata tailX per ottenere la coordinata x corretta per allineare il testo a destra. Infine, viene restituita la coordinata x calcolata.

2.12 UtilityTool.java

La classe contiene metodi utili per il ridimensionamento delle immagini.

2.12.1 metodo scaleImage

Il metodo permette di ridimensionare un'immagine di tipo BufferedImage[16] alle dimensioni desiderate utilizzando le tecniche standard di ridimensionamento delle immagini mantenendo il tipo dell'immagine originale.

3 Pacchetto ai

3.1 Node.java

La classe ha la funzione di creare degli oggetti nodo da utilizzare in un algoritmo di ricerca del percorso più rapido chiamato A* Pathfinder per rendere il combattimento più avvincente. Sono definiti i seguenti membri della classe:

- parent: un riferimento al nodo genitore di questo nodo nel percorso.
- col: la colonna del nodo nella griglia o nel sistema di coordinate.
- row: la riga del nodo nella griglia o nel sistema di coordinate.
- gCost: il costo del percorso dal nodo di partenza a questo nodo.
- hCost: il costo dal nodo corrente al nodo di destinazione.
- fCost: la somma del gCost e hCost, che rappresenta una stima del costo totale per attraversare il percorso dal nodo di partenza al nodo di destinazione passando per questo nodo.
- solid: un flag booleano che indica se il nodo è solido o non attraversabile.
- open: un flag booleano che indica se il nodo è stato visitato e considerato nella ricerca del percorso.
- checked: un flag booleano che indica se il nodo è stato verificato o esplorato durante l'esecuzione dell'algoritmo di ricerca.

[15] La variabile tailX è calcolata come la coordinata x del bordo destro del frame meno un offset deciso.
[16] BufferedImage è una classe fornita dalla libreria Java java.awt.image. Rappresenta un'immagine raster, ovvero un'immagine composta da pixel, che può essere utilizzata per manipolare e gestire immagini in Java.

3.1.1 Costruttore

Il costruttore della classe accetta due parametri col e row e inizializza i membri col e row del nodo corrente con i valori forniti.

3.2 PathFinder.java

La classe contiene gli oggetti e i metodi utilizzati per implementare nel gioco un algoritmo di ricerca per il miglior percorso. Nella classe:

- Node[][] node: e una matrice bidimensionale di oggetti Node che rappresenta la griglia o il sistema di nodi utilizzato per trovare il percorso;
- openList: è un ArrayList di oggetti Node che rappresenta la lista di nodi che devono ancora essere esplorati o valutati;
- pathList: è un ArrayList di oggetti Node che rappresenta il percorso trovato, ovvero l'elenco dei nodi dal nodo di partenza al nodo di destinazione;
- startNode: riferimento al nodo di partenza.
- goalNode: riferimento al nodo di destinazione o obiettivo.
- currentNode: riferimento al nodo corrente che viene esaminato o valutato durante la ricerca del percorso.
- goalReached: flag che indica se il nodo di destinazione è stato raggiunto o se è stato trovato un percorso valido.
- step: int che tiene traccia del numero di passaggi o iterazioni eseguite durante la ricerca del percorso.

3.2.1 Costruttore

Nel costruttore della classe si assegna l'oggetto gp passato come parametro al campo gp dell'istanza corrente di PathFinder. In questo modo, l'oggetto PathFinder può accedere e interagire con le proprietà e i metodi del GamePanel attraverso il campo gp, inoltre chiama il metodo instantiateNodes().

3.2.2 metodo instantiateNodes

Il metodo inizializza i nodi in tutta la mappa creando una griglia con le corrispondenti coordinate

3.2.3 metodo resetNodes

Il metodo reimposta le variabili booleane open, checked e solid a false per ogni nodo generato.

3.2.4 metodo setNodes

Il metodo è responsabile per l'impostazione dei nodi di partenza e destinazione, nonché la configurazione dei nodi solidi in base alla mappa di gioco. All'inizio, vengono resettati tutti i nodi esistenti. Successivamente, il nodo di partenza viene impostato utilizzando le coordinate fornite come argomenti, mentre il nodo corrente viene inizializzato come il nodo di partenza. Il nodo di destinazione viene impostato di conseguenza. Viene creato un elenco aperto (openList) contenente il nodo corrente. Successivamente, viene iterato su tutti i nodi nella griglia utilizzando le variabili di colonna e riga. Durante questo processo, vengono identificati e impostati come solidi i nodi

corrispondenti ai tile con collisione nella mappa di gioco. Inoltre, vengono rilevati e impostati come solidi i nodi corrispondenti ai tile distruttibili. Per ciascun nodo, vengono calcolati i costi gCost, hCost e fCost. Infine, vengono eseguite le operazioni necessarie per preparare l'algoritmo di ricerca del percorso. In sintesi, il metodo setNodes inizializza i nodi di partenza e destinazione e configura i nodi solidi in base alla mappa di gioco, fornendo così le informazioni necessarie per la successiva ricerca del percorso.

3.2.5 metodo getCost

Il metodo è responsabile per il calcolo dei costi gCost, hCost e fCost di un nodo specifico. Utilizzando le coordinate del nodo corrente e dei nodi di partenza e destinazione, vengono calcolate le distanze orizzontali (xDistance) e verticali (yDistance) tra il nodo corrente e il nodo di partenza, nonché tra il nodo corrente e il nodo di destinazione. Il costo gCost viene quindi calcolato sommando le distanze orizzontali e verticali tra il nodo corrente e il nodo di partenza. Il costo hCost viene calcolato in modo simile, sommando le distanze orizzontali e verticali tra il nodo corrente e il nodo di destinazione. Infine, il costo totale fCost viene calcolato sommando gCost e hCost.

3.2.6 metodo search

Il metodo implementa l'algoritmo di ricerca del percorso. Utilizzando un ciclo while, l'algoritmo continua a eseguire fino a quando (goalReached) è falsa e il numero di iterazioni (step) è inferiore a 500. All'interno del ciclo, vengono esaminati i nodi adiacenti al nodo corrente, i nodi adiacenti vengono successivamente passati al metodo openNode che li aggiunge all'elenco dei nodi aperti se soddisfano determinate condizioni, viene selezionato il nodo con il costo fCost più basso dall'elenco dei nodi aperti. Nel caso in cui ci siano più nodi con lo stesso costo fCost, viene selezionato quello con il costo gCost più basso. Se non ci sono nodi nell'elenco dei nodi aperti, l'algoritmo si interrompe. Il nodo corrente viene quindi aggiornato con il nodo selezionato e viene verificato se corrisponde al nodo di destinazione. In caso affermativo, viene impostata la variabile goalReached su true e viene chiamato il metodo trackThePath per tracciare il percorso dal nodo di destinazione al nodo di partenza. Infine, viene incrementato il numero di iterazioni step. Il metodo restituisce il valore di goalReached, indicando se è stato raggiunto il nodo di destinazione durante la ricerca del percorso.

3.2.7 metodo openNode

Il metodo viene utilizzato per aprire un nodo durante la ricerca del percorso. Viene controllato se il nodo non è ancora stato aperto (node.open == false), non è stato già verificato (node.checked == false) e non è un nodo solido (node.solid == false). In tal caso, il nodo viene aperto impostando node.open su true, viene assegnato il nodo corrente come genitore (node.parent = currentNode) e infine viene aggiunto all'elenco dei nodi aperti (openList.add(node)).

3.2.8 metodo trackThePath

Il metodo ù viene utilizzato per tracciare il percorso una volta raggiunto il nodo obiettivo durante la ricerca. Viene creato un nodo temporaneo current e viene iniziato un ciclo while che continua finché il nodo corrente non diventa il nodo di partenza (current != startNode). All'interno del ciclo, il nodo corrente viene aggiunto all'inizio della lista del percorso (pathList.add(0, current)) e viene aggiornato il nodo corrente con il suo genitore (current = current.parent). In questo modo, il percorso viene tracciato all'indietro dal nodo obiettivo fino al nodo di partenza. Alla fine del ciclo, la lista del percorso conterrà i nodi in ordine dal nodo di partenza al nodo obiettivo.

4 Pacchetto data

4.1 DataStorage.java

Il codice fornito definisce la classe DataStorage, che implementa l'interfaccia Serializable per consentire la serializzazione degli oggetti di questa classe. La classe DataStorage è utilizzata per memorizzare i dati di gioco e le informazioni sullo stato del giocatore, dell'inventario e degli oggetti sulla mappa come:

- level: Rappresenta il livello del giocatore.
- maxLife: Rappresenta il valore massimo della vita del giocatore.
- life: Rappresenta il valore corrente della vita del giocatore.
- maxMana: Rappresenta il valore massimo del mana del giocatore.
- mana: Rappresenta il valore corrente del mana del giocatore.
- strength: Rappresenta la forza del giocatore.
- dexterity: Rappresenta la destrezza del giocatore.
- exp: Rappresenta l'esperienza del giocatore.
- nextLevelExp: Rappresenta l'esperienza necessaria per passare al livello successivo.
- coin: Rappresenta la quantità di monete possedute dal giocatore.
- itemNames: ArrayList che contiene i nomi degli oggetti nell'inventario del giocatore.
- itemAmmounts: ArrayList che contiene le quantità degli oggetti nell'inventario del giocatore.
- currentWeaponSlot: Rappresenta lo slot corrente dell'arma equipaggiata dal giocatore.
- currentShieldSlot: Rappresenta lo slot corrente dello scudo equipaggiato dal giocatore.
- currentToolSlot: Rappresenta lo slot corrente dello strumento equipaggiato dal giocatore.
- mapObjectNames: Matrice di stringhe che contiene i nomi degli oggetti sulla mappa.
- mapObjectWorldX: Matrice di interi che contiene le coordinate X degli oggetti sulla mappa.
- mapObjectWorldY: Matrice di interi che contiene le coordinate Y degli oggetti sulla mappa.
- mapObjectLootName: Matrice di stringhe che contiene i nomi degli oggetti ottenibili dagli oggetti sulla mappa.
- mapObjectOpened: Matrice di booleani che indica se gli oggetti sulla mappa sono stati aperti o meno.

4.2 SaveLoad.java

Classe incaricata al caricamento e al salvataggio delle informazioni.

4.2.1 Costruttore

Viene passato GamePanel.

4.2.2 metodo save

Il metodo salva le informazioni di gioco attuali, inclusi i dati del giocatore, dell'inventario e degli oggetti sulla mappa, nel file di salvataggio "save.dat".[17]

4.2.3 metodo load

Il metodo carica i dati di gioco salvati nel file "save.dat" e li ripristina nel GamePanel, inclusi i dati del giocatore, dell'inventario e degli oggetti sulla mappa.[18]

5 Pacchetto entity

5.1 Entity.java

La classe Entity rappresenta un'entità nel gioco e contiene diverse variabili e attributi che definiscono le caratteristiche dell'entità. Alcune delle principali variabili e attributi della classe includono:

- variabili per la posizione nel mondo (worldX e worldY) e le dimensioni dell'area solida (solidAreaDefaultX e solidAreaDefaultY);
- immagini utilizzate per la rappresentazione grafica dei personaggi in diverse direzioni e azioni;
- si specificano le aree solide e le aree solide di attacco come oggetti Rectangle;
- un inventario rappresentato da una lista di entità (inventory) con una dimensione massima (maxInventorySize).
- variabili booleane che indicano lo stato dell'entità, come collisioni, invincibilità, attacchi, vita, etc;
- contatori utilizzati per vari scopi, come il conteggio degli sprite, il conteggio della vita rimanente, ecc;
- array di stringhe (dialogues) per gestire i dialoghi dell'entità;
- attributi del personaggio come il nome, la velocità, la vita massima, il livello, la forza, la destrezza, l'attacco, la difesa, l'esperienza, ecc;
- attributi degli oggetti come il valore, il valore di attacco, il valore di difesa, la descrizione, il costo di utilizzo, il prezzo, ecc.
- costanti per definire i diversi tipi di entità.

5.1.1 Costruttore

Viene inizializzato gp in modo da poter utilizzare tutti i metodi della classe GamePanel.

5.1.2 metodi getsScreenX, getScreenY

I metod restituiscono le coordinate altezza e lunghezza dello schermo partendo dal giocatore.

[17] ObjectOutputStream è una classe in Java che consente di scrivere oggetti su un output stream. Viene utilizzata per la serializzazione degli oggetti, cioè per convertire gli oggetti in una sequenza di byte che possono essere scritti su un file o trasmessi su una rete.

[18] ObjectInputStream è una classe in Java che consente di leggere oggetti da un input stream. Viene utilizzata per deserializzare gli oggetti, cioè per convertire una sequenza di byte letta da un file.

5.1.3 metodi getLeftX, getRightX, getTopY, getBottomY, getCol, getRow

Metodi getter per ottenere la posizione del giocatore all'interno della mappa.

5.1.4 metodi getCenterX, getCenterY

I metodi sono utilizzati per ottenere le coordinate del punto centrale dell'oggetto rispetto al suo sistema di coordinate. In questo modo il punto di riferimento per raggiungere un "goal", punto di arrivo, non sarà più l'angolo in alto a sinistra.

5.1.5 metodi getXdistance, getYdistance, getTileDistance

Questi metodi vengono utilizzati per calcolare le distanze orizzontali, verticali e in termini di tile tra l'oggetto corrente e un altro oggetto specificato come parametro target.

5.1.6 metodi getGoalCol, getGoalRow

Questi metodi vengono utilizzati per calcolare la colonna e la riga obiettivo (goal) in base alla posizione dell'oggetto target rispetto alla matrice dei tile, utilizzando la dimensione del tile.

5.1.7 metodo setLoot

Il metodo è sovrascritto nelle classi che lo utilizzano come le casse per impostare il loro bottino.

5.1.8 metodo setAction

Il metodo imposta delle azioni che qualsiasi entità può compiere, viene effettuato l'overload del metodo nelle specifiche classi che lo utilizzano.

5.1.9 metodo damageReaction

Il metodo imposta delle azioni che qualsiasi entità esegue dopo essere stata colpita, viene effettuato l'overload del metodo nelle specifiche classi che lo utilizzano.

5.1.10 metodo speak

Il metodo si occupa di gestire il dialogo corrente e di aggiornare la direzione dell'entità che sta parlando per fornire una rappresentazione visiva coerente con la direzione del giocatore.

5.1.11 metodo turnToPlayer

Il metodo consente all' entità corrente di girarsi in direzione opposta rispetto al giocatore, rendendo possibile una reazione o un'interazione specifica con il giocatore stesso.

5.1.12 metodo startDialogue

Il metodo prepara il gioco per l'avvio di un dialogo con un'entità specifica, impostando lo stato del gioco e fornendo le informazioni necessarie all'interfaccia utente per gestire correttamente il dialogo.

5.1.13 metodo interact

Gestisce gli eventi che accadono nel caso si interagisce con oggetti che sovraccaricano il metodo, come porte e casse.

5.1.14 metodo use

Il metodo imposta delle azioni al momento dell'utilizzo dell'entità, viene effettuato l'overload del metodo nelle classi interessate.

5.1.15 metodo checkDrop

Il metodo imposta dei drop rate ovvero la possibilità che un mostro lasci un determinato oggetto a terra, si effettua l'overload del metodo nelle classi interessate.

5.1.16 metodo dropItem

Il metodo trova uno spazio disponibile nell'array degli oggetti gp.obj[][][] e vi inserisce l'oggetto droppato, impostandone anche le coordinate sulla mappa. Questo consente all'oggetto droppato di essere visualizzato e interagibile nella posizione in cui l'entità corrente l'ha lasciato cadere.

5.1.17 metodi getParticleColor, getParticleSize, getParticleSpeed, getParticleMaxLife

I metodi vengono sovrascritti nella classe interessata per restituire i valori corretti relativi alle particelle da generare.

5.1.18 metodo generateParticle

Il metodo genera delle particelle utilizzando un generatore (generator) e una destinazione (target). Per generare le particelle, vengono utilizzati i valori specifici del generatore attraverso i metodi getParticleColor(), getParticleSize(), getParticleSpeed() e getParticleMaxLife(). All'interno del metodo, vengono ottenuti il colore, la dimensione, la velocità e la durata massima delle particelle dal generatore. Successivamente, vengono create quattro istanze di Particle utilizzando i valori ottenuti e posizioni relative (-2, -1), (-2, 1), (2, -1) e (2, 1). Infine, le nuove particelle vengono aggiunte alla lista delle particelle (gp.particleList). In questo modo, le particelle generate avranno le caratteristiche specifiche del generatore, come il colore, la dimensione, la velocità e la durata massima, consentendo di personalizzare il comportamento delle particelle in base al generatore utilizzato.

5.1.19 metodo checkCollision

Il metodo controlla la presenza di collisioni per l'entità corrente. Inizialmente, imposta la variabile collisionOn su false. Successivamente, vengono eseguiti controlli per le collisioni con i tile, gli oggetti, le entità (come NPC, mostri e oggetti interattivi) e infine con il giocatore. Se l'entità corrente è di tipo "mon" (mostro) e si verifica un contatto con il giocatore, viene inflitto danno al giocatore utilizzando il valore di attacco dell'entità corrente.

5.1.20 metodo update

Il metodo update() rappresenta il cuore dell'aggiornamento di un oggetto o personaggio nel contesto del gioco. Esso gestisce diversi aspetti cruciali per il corretto funzionamento del personaggio. Innanzitutto, viene controllato se l'oggetto è soggetto a knockback, cioè se è stato spinto all'indietro a causa di una collisione. Viene verificata la presenza di una collisione attraverso il metodo checkCollision(). Se la collisione è confermata, vengono reimpostati i parametri relativi al knockback e il personaggio riprende la sua velocità predefinita. In caso contrario, l'oggetto viene spostato nella direzione del knockback precedentemente definita. Successivamente, viene controllato se l'oggetto è in fase di attacco. Se sì, viene eseguita l'azione di attacco richiamando il metodo attacking(), che gestisce gli aspetti specifici dell'attacco. Se l'oggetto non è né soggetto a knockback né in fase di attacco, si procede con il comportamento predefinito. Viene impostata l'azione dell'oggetto attraverso il metodo setAction(), seguito dalla verifica di collisioni tramite checkCollision(). Se non

viene rilevata alcuna collisione, l'oggetto si muove nella direzione definita da direction. L'animazione è un elemento fondamentale per rendere il personaggio più dinamico e coinvolgente. All'interno del blocco dedicato all'animazione, viene incrementato il contatore spriteCounter. Quando il contatore supera un certo valore, il numero di sprite spriteNum viene alternato tra due opzioni, generando così un effetto di animazione alternata. Alcuni aspetti importanti riguardano l'invincibilità dei mostri, il timer dei colpi e lo stordimento del mostro. Se l'invincibilità è attiva, viene incrementato un contatore fino a raggiungere un valore massimo, dopodiché l'invincibilità viene disattivata. Il timer dei colpi viene incrementato fino a un valore massimo di 30. Lo stordimento del mostro segue un meccanismo simile all'invincibilità, con un contatore che viene incrementato fino a superare un valore limite, a quel punto lo stordimento viene rimosso. Infine, viene gestita la rinascita dei mostri di tipo slime. Un contatore viene incrementato periodicamente e quando raggiunge un valore massimo, viene chiamato il metodo setMonster() per rigenerare i mostri di tipo slime sullo scenario di gioco.

5.1.21 metodo isAttackingOrNot

Il metodo determina se l'entità può attaccare il giocatore in base a una frequenza di attacco specifica e a una distanza specifica in orizzontale e verticale. Se le condizioni sono soddisfatte, l'entità viene impostata nello stato di attacco.

5.1.22 metodo isShootingOrNot

Il metodo determina se l'entità può sparare un proiettile basandosi su una frequenza di sparo specifica e su un intervallo tra gli spari. Se le condizioni sono soddisfatte, viene creato un nuovo proiettile e assegnato a uno slot libero nella matrice dei proiettili.

5.1.23 isChasing

Il metodo determina se l'entità deve iniziare a inseguire un bersaglio in base alla distanza massima consentita e a una frequenza specifica. Se le condizioni sono soddisfatte, viene impostata la variabile onPath a true.

5.1.24 isNotChasing

Il metodo determina se l'entità deve interrompere l'inseguimento di un bersaglio in base alla distanza massima consentita e a una frequenza specifica. Se le condizioni sono soddisfatte, viene impostata la variabile onPath a false.

5.1.25 getRandomDirection

Il metodo restituisce una direzione casuale per l'entità chiamante con una frequenza definita dall'intervallo specificato.

5.1.26 moveToPlayer

Il metodo fa muovere l'entità chiamante verso il giocatore, determinando la direzione di movimento in base alla posizione relativa del giocatore rispetto all'entità. L'intervallo specificato controlla la frequenza di aggiornamento del movimento.

5.1.27 attacking

All'interno del metodo, viene incrementato il contatore spriteCounter, che tiene traccia del progresso dell'animazione dell'attacco. Viene quindi eseguito un controllo per determinare quale sprite deve essere visualizzato in base alla durata delle diverse fasi dell'animazione. Se il contatore spriteCounter è inferiore o uguale alla durata della prima fase dell'animazione (motion1_duration), viene impostato

il primo sprite (spriteNum = 1). Se il contatore spriteCounter è compreso tra la durata della prima fase dell'animazione (motion1_duration) e la durata della seconda fase dell'animazione (motion2_duration), viene impostato il secondo sprite (spriteNum = 2). Durante questa fase, vengono eseguite diverse operazioni relative all'attacco:

- viene salvata la posizione corrente della solidArea dell'entità chiamante;
- viene modificata la solidArea per adattarla all'area di attacco in base alla direzione dell'attacco;
- viene eseguito un controllo delle collisioni per determinare gli eventuali bersagli dell'attacco e applicare loro il danno;
- nel caso in cui l'entità chiamante sia di tipo type_mon (mostro), viene danneggiato il giocatore;
- nel caso in cui l'entità chiamante sia il giocatore, vengono eseguiti i seguenti controlli:
  - collisione con mostri: il giocatore danneggia il mostro coinvolto nell'attacco;
  - collisione con piastrelle interattive: il giocatore interagisce con la piastrella coinvolta nell'attacco;
  - collisione con proiettili: il giocatore elimina i proiettili colpiti durante l'attacco;
- dopo aver eseguito i controlli delle collisioni, vengono ripristinate la posizione e le dimensioni originali della solidArea.

Infine, se il spriteCounter supera la durata della seconda fase dell'animazione (motion2_duration), viene reimpostato il primo sprite, il contatore spriteCounter viene azzerato e la variabile attacking viene impostata su false, indicando la fine dell'attacco.

5.1.28 getOppositeDirection

Il metodo restituisce la posizione opposta. Il metodo è controllato per verificare che in una situazione di blocco del colpo da parte del giocatore, le due entità si trovino faccia a faccia

5.1.29 metodo damagePlayer

Il metodo viene utilizzato per infliggere danni al giocatore durante un combattimento nel gioco. Dopo verificare l'invincibilità del giocatore, calcola il danno effettivo sottraendo la difesa del giocatore dall'attacco del nemico. Se il giocatore è in modalità difensiva e la direzione del nemico è opposta alla direzione di difesa del giocatore, gestisce i casi di blocco totale "parry" o di riduzione del colpo. Viene riprodotto un effetto sonoro appropriato in base alla situazione. Se il danno non è nullo, vengono eseguite azioni come rendere il giocatore trasparente e impostare la reazione al colpo. Infine, viene aggiornata la vita del giocatore e viene impostata l'invincibilità temporanea.

5.1.30 metodo setKnockBack

Il metodo viene utilizzato per impostare l'effetto di respinta su un'entità bersaglio in seguito a un attacco da parte di un'altra entità aggressore. Vengono impostati l'aggressore, la direzione di respinta, la velocità incrementata e viene attivato il flag di respinta sull'entità bersaglio.

5.1.31 metodo inCamera

Il metodo indica se qualcosa è nello schermo di gioco in tal caso restituisce true, false altrimenti.

5.1.32 metodo draw

Il metodo è responsabile per disegnare l'entità sulla schermata del gioco. Si verifica se l'entità è all'interno del campo visivo del giocatore, utilizzando il metod inCamera(). A seconda della direzione e del numero di sprite, viene selezionata l'immagine appropriata per l'entità. Se l'entità è invincibile, viene regolata l'opacità dell'immagine utilizzando il metodo changeAlpha e viene disegnata l'animazione di morte se l'entità sta morendo (dying è true). Infine, viene disegnata l'immagine dell'entità utilizzando le coordinate calcolate.

5.1.33 metodo dyingAnimation

Il metodo gestisce l'animazione di morte dell'entità regolando l'opacità dell'immagine in fasi di transizione e terminando l'animazione una volta completata infine imposta il flag alive su false.

5.1.34 metodo changeAlpha

Metodo utilizzato per cambiare la trasparenza (alphaValue) di un mostro per indicare che è stato sconfitto.

5.1.35 metodo setup

Il metodo riceve il percorso dell'immagine, la larghezza e l'altezza desiderate come parametri e restituisce un oggetto BufferedImage. All'interno del metodo, viene creata un'istanza di UtilityTool per utilizzare un metodo di utilità. Successivamente, l'immagine viene letta dal percorso specificato utilizzando ImageIO.read() e ridimensionata utilizzando il metodo scaleImage() dell'oggetto uTool. Infine, l'immagine ridimensionata viene restituita dal metodo.

5.1.36 metodo searchPath

Il metodo nella classe implementa l'algoritmo A* per cercare un percorso ottimale tra la posizione corrente del personaggio controllato dal giocatore e una destinazione desiderata nel mondo di gioco. Utilizzando le coordinate della destinazione, il metodo calcola le coordinate di partenza e imposta i nodi corrispondenti. Successivamente, avvia la ricerca del percorso chiamando il metodo search. Se viene trovato un percorso valido, vengono determinate la direzione corretta in cui il personaggio deve muoversi confrontando le posizioni del personaggio e del prossimo nodo nel percorso. Infine, il metodo controlla se il prossimo nodo corrisponde alla destinazione, e in tal caso, imposta il flag onPath su false.

5.1.37 metodo getDetected

Il metodo restituisce l'indice dell'entità specificata all'interno della matrice target corrispondente alla posizione successiva dell'entità chiamante in base alla sua direzione di movimento. Viene eseguito un controllo sulle entità presenti nella matrice target per trovare una corrispondenza con le coordinate calcolate e il nome specificato.

5.2 NPC_Mage.java

La classe estende la classe Entity e rappresenta un personaggio non giocante "mago" nel gioco.

5.2.1 Costruttore

Viene chiamato il costruttore della classe padre Entity passando l'istanza di GamePanel come parametro. Viene impostata la direzione del personaggio su "down" (verso il basso) e la velocità a 1. Viene inizializzata l'area solida (solidArea) del personaggio impostando le coordinate x e y, la

larghezza (width) e l'altezza (height). Vengono anche salvate le coordinate predefinite dell'area solida (solidAreaDefaultX e solidAreaDefaultY). Vengono quindi chiamati i metodi getImage() per ottenere l'immagine del personaggio e setDialogue() per impostare il dialogo associato al personaggio.

5.2.2 metodo getImage

Il metodo getImage() viene utilizzato per caricare le immagini associate alle diverse direzioni del personaggio "mago". Vengono utilizzati i metodi setup() e vengono passati i percorsi delle immagini delle diverse direzioni del personaggio, insieme alla dimensione del tile (gp.tileSize).

5.2.3 metodo setDialogue

Vengono aggiunte stringhe all'array dei dialoghi in modo che quando si interagisce con esso scorrano i dialoghi nell'array.

5.2.4 metodo setAction

Il metodo setAction() determina l'azione da eseguire per il personaggio "mago". Se onPath è true, il personaggio inizierà a cercare il percorso migliore per arrivare alla destinazione prefissata da goalCol e goalRow. Se onPath è false, significa che il personaggio non segue un percorso predefinito e viene chiamato il metodo getRandomDirection() della classe Entity. Il metodo è l'overload del metodo setAction() in Entity.java.

5.2.5 metodo speak

Il metodo inizia il dialogo con il Mago. Il metodo è l'overload del metodo speak() in Entity.java.

5.3 NPC_Merchant.java

La classe estende la classe Entity e rappresenta il mercante nel gioco.

5.3.1 Costruttore

Il costruttore della classe NPC_Merchant riceve un oggetto GamePanel come argomento e chiama il costruttore della classe padre Entity mediante super(gp). Viene impostata la direzione del personaggio come "down" e vengono specificate le dimensioni dell'area solida del personaggio mediante l'oggetto solidArea. Successivamente, vengono chiamati i metodi getImage() per ottenere le immagini del personaggio, setDialogue() per impostare i dialoghi e setItems() per impostare gli oggetti associati al mercante.

5.3.2 metodo getImage

Il metodo viene utilizzato per caricare le immagini associate al personaggio. Vengono utilizzati i metodi setup() e vengono passati i percorsi delle immagini delle diverse direzioni del personaggio, insieme alla dimensione del tile (gp.tileSize). Dato che il personaggio è fermo non avremo bisogno di caricare immagini del personaggio verso l'alto, destra e sinistra che saranno sempre uguali dato che il personaggio è immobile.

5.3.3 metodo setDialogue

Vengono aggiunte stringhe all'array dei dialoghi in modo che quando si interagisce con esso scorrano i dialoghi nell'array.

5.3.4 metodo setItems

Il metodo definisce gli oggetti che possiede il mercante, gli oggetti presenti nell'inventario del mercante potranno essere acquistati dal giocatore. Gli oggetti variano in base al livello del giocatore.

5.3.5 metodo speak

Il metodo speak() avvia una conversazione con l'NPC, imposta lo stato di gioco e l'NPC corrente nell'interfaccia utente, e aggiorna gli oggetti nell'inventario del giocatore.

5.4 Particle.java

La classe Particle estende la classe Entity e rappresenta una particella nel gioco.

5.4.1 Costruttore

Il costruttore definisce le seguenti variabili membro:

- generator: un oggetto di tipo Entity che rappresenta il generatore della particella;
- color: un oggetto di tipo Color che rappresenta il colore della particella;
- xd e yd: due variabili intere che indicano lo spostamento della particella lungo gli assi x e y;
- size: una variabile intera che rappresenta la dimensione della particella.

5.4.2 metodo update

Il metodo aggiorna lo stato della particella. Decrementa il valore di life per indicare la diminuzione della durata della particella. Se la vita della particella è inferiore a un terzo del valore massimo (maxLife), aumenta il valore di yd per far sì che la particella cada verso il basso. Aggiorna le coordinate della particella (worldX e worldY) in base ai valori di xd, yd e speed, spostando la particella lungo gli assi x e y in modo da simulare un effetto gravità. Se la vita della particella raggiunge 0, imposta il valore di alive su false per indicare che la particella non è più attiva.

5.4.3 metodo draw

Il metodo draw(Graphics2D g2) disegna la particella sullo schermo. Calcola le coordinate screenX e screenY della particella rispetto al giocatore e alla posizione della telecamera, imposta il colore della particella. Disegna un rettangolo di dimensioni (size), posizionato alle coordinate screenX e screenY, rappresentando così la particella sullo schermo.

5.5 Player.java

La classe estende la classe Entity e rappresenta il giocatore nel gioco. Si crea un oggetto KeyHandler di nome keyH utilizzato per gestire gli input da tastiera per il giocatore, vengono definite le variabili screenX e screenY, infine la variabile attackCanceled indica se l'attacco del giocatore è stato annullato o meno.

5.5.1 Costruttore

Nel costruttore della classe si passano come parametri gp e keyH in modo da poter utilizzare i metodi definiti nelle due classi GamePanel.java e KeyHandler.java, viene quindi inizializzata l'area solida del personaggio utilizzando un oggetto Rectangle. L'area solida viene posizionata in base alle coordinate specificate e viene impostata la larghezza e l'altezza. Infine, vengono richiamati i metodi setDefaultValues(), getPlayerImage(), getPlayerAttackImage() e setItems() per inizializzare ulteriori valori e ottenere le immagini del giocatore in movimento e degli attacchi del giocatore, nonché impostare gli oggetti.

5.5.2 metodo setDefaultValues

Vengono impostate tutte le variabili di gioco riguardanti il personaggio come vita, mana, livello, esperienza attuale, esperienza necessaria per aumentare di livello, ecc....

5.5.3 metodo setDefaultPosition

Il metodo imposta la posizione e la direzione iniziale del personaggio nel mondo i gioco.

5.5.4 metodo setDialogues

Inizializza i dialoghi utilizzati nella classe Player.

5.5.5 metodo restoreStatus

Il metodo ripristina salute e mana del personaggio, i suoi stati di invincibilità, trasparenza, attacco, guardia, knockback, aggiorna il filtro di luce e resetta la sua velocià il metodo è richiamato nel metodo resetGame().

5.5.6 metodo setItems

Il metodo ripulisce l'inventario e imposta come items di default un arma e uno scudo base.

5.5.7 metodo getAttack

Il metodo restituisce l'attacco del giocatore basato sulla forza e sull'arma attuale del giocatore in gioco.

5.5.8 metodo getDefense

Il metodo restituisce la difesa del giocatore basato sulla resistenza e sullo scudo attuale del giocatore in gioco.

5.5.9 metodi getCurrentWeaponSlot, getCurrentShieldSlot, getCurrentToolSlot

Il metodo getCurrentWeaponSlot() scorre l'inventario del giocatore e restituisce l'indice dell'arma corrente, che corrisponde all'oggetto currentWeapon. Il metodo getCurrentShieldSlot() scorre l'inventario del giocatore e restituisce l'indice dello scudo corrente, che corrisponde all'oggetto currentShield. Il metodo getCurrentToolSlot() scorre l'inventario del giocatore e restituisce l'indice dello strumento corrente, che corrisponde all'oggetto currentTool.

5.5.10 metodo getImage

Il metodo viene utilizzato per impostare le immagini del personaggio del giocatore (Player) nelle diverse direzioni e stati. All'interno del metodo, vengono richiamate diverse volte il metodo setup() per ottenere le immagini e assegnarle alle variabili di istanza corrispondenti.

5.5.11 metodo getSleepingImage

Il metodo sostituisce all'immagine del giocatore l'immagine "image" passata nel metodo come parametro. In questo caso verrà sostituito quando si utilizzerà l'oggetto Tenda.

5.5.12 metodo getAttackImage

Il metodo di funzionamento è identico al metodo precedente a differenza che le immagini per attaccare variano in base alla staffa equipaggiata.

5.5.13 metodo getGuardImage

Il metodo viene utilizzato per impostare le immagini del personaggio del giocatore (Player) nelle diverse direzioni mentre utilizza lo scudo.

5.5.14 metodo update

Il metodo update() è responsabile per l'aggiornamento del giocatore (Player) durante il gioco. All'interno del metodo, vengono eseguite diverse azioni in base agli input del giocatore e alle condizioni di gioco. Si controlla se il giocatore sta attaccanto, in tal caso si esegue il metodo attacking() per gestire le animazioni d'attacco e le collisioni del caso. Negli altri casi si controlla:

- la direzione del giocatore in base al tasto premuto;
- viene disattivata la variabile booleana collisionOn che sarà aggiornata in base ai metodi per il controllo delle collisioni con oggetti, NPC, mostri e caselle interattive della classe CollisionChecker.java;

- si verificano se avvengono degli eventi con il metodo checkEvent() della classe EventHandler.java;
- se la collisione non è attiva e il tasto di attacco (enterPressed) non è premuto, il giocatore viene spostato nella direzione corrispondente;
- se viene premuto il tasto per l'attacco, si richiama il metodo per attaccare attacking(), dopodiché si reimpostano a false attackCanceled e enterPressed;
- si aggiorna il contatore per gestire l'animazione della camminata del personaggio formata da 2 sprite separati che si alternano;
- si gestiscono le palle di fuoco sparate dal giocatore, se si verificano le condizioni necessarie quindi se il tasto "SPACE" è stato premuto se non ci sono proiettili in vita se il contatore dei colpi è uguale a 30 e se il giocatore ha abbastanza risorse:
  - viene creato un nuovo proiettile (projectile) e viene decrementata la risorsa necessaria per utilizzarlo (subtractResource);
  - il proiettile viene aggiunto alla lista dei proiettili nel mondo di gioco (projectile[currentMap][i]);
  - si riproduce un effetto sonoro;
  - fin quando il contatore non è minore di 30 viene incrementato;
- si imposta una frazione di tempo in cui il giocatore non può essere colpito utilizzando un contatore, se il contatore supera un certo valore l'invincibilità viene disattivata;
- si controlla se vita e mana del giocatore superano i valori massimi in tal caso vengono impostati al massimo disponibile;
- si imposta un contatore che gestisce la rinascita dei mostri nella mappa dopo un certo lasso di tempo;
- viene eseguito un controllo sulla vita del giocatore. Se la vita è inferiore o uguale a 0, viene impostato lo stato di gioco come "game over" (gameState) e vengono eseguite altre azioni correlate.

5.5.15 metodo pickUpObject

Il metodo 'pickUpObject' viene utilizzato per raccogliere un oggetto specifico nell'indice 'i'. Se l'indice non è uguale a 999, viene eseguita una serie di operazioni. Se l'oggetto è di tipo "pickUp" (come le monete), viene chiamato il metodo 'use(this)' sull'oggetto per eseguire l'effetto corrispondente e successivamente l'oggetto viene impostato su 'null'. Se l'oggetto è di tipo "obstacle" (ostacolo), viene verificato se il tasto "enter" è premuto e, in caso affermativo, viene chiamato il metodo 'interact()' sull'oggetto e la variabile 'attackCanceled' viene impostata su 'true'. Altrimenti, se l'oggetto non rientra nelle categorie precedenti, viene verificato se l'inventario non è pieno. In tal caso, l'oggetto viene aggiunto all'inventario ('inventory') e viene riprodotto un effetto sonoro. Viene anche visualizzato un messaggio appropriato tramite 'gp.ui.addMessage()'. Se l'inventario è già pieno, viene visualizzato un messaggio di inventario pieno. Infine, l'oggetto nella mappa corrente ('gp.obj[gp.currentMap][i]') viene impostato su 'null'.

5.5.16 metodo interactNPC

All'interno del metodo, viene eseguito un controllo per verificare se il tasto "Enter" è stato premuto (gp.keyH.enterPressed == true). Viene impostata la variabile attackCanceled su true per annullare eventuali azioni di attacco in corso. Successivamente, viene impostato lo stato di gioco (gameState) su dialogueState per avviare lo stato di dialogo del gioco. Infine, viene chiamato il metodo speak() del personaggio non giocante per avviare il dialogo con il personaggio.

5.5.17 metodo contactMon

All'interno del metodo, viene controllato se il giocatore è invincibile (invincible == false) e se il mostro non è in fase di morte (gp.mon[gp.currentMap][i].dying == false). Se queste condizioni sono soddisfatte, viene riprodotto un effetto sonoro per indicare l'attacco del mostro al giocatore. Viene calcolato il danno inflitto dal mostro sottraendo la difesa del giocatore dall'attacco del mostro. Se il valore del danno è inferiore a 0, viene impostato a 0 per evitare danni negativi. Successivamente, il danno viene sottratto dalla vita del giocatore (life -= damage) e la variabile invincible viene impostata su true per renderlo temporaneamente invincibile.

5.5.18 metodo damageMon

All'interno del metodo, si controlla se il mostro che si sta attaccando è temporaneamente invincibile, se si soddisfa la condizione viene riprodotto un effetto sonoro per indicare il danno al mostro e verrà applicato al mostro colpito l'effetto "knock back" il mostro viene quindi spostato indietro in base alla forza dell'arma. Viene calcolato il danno inflitto dal giocatore al mostro sottraendo la difesa del mostro dall'attacco del giocatore. Se il valore del danno è inferiore a 0, viene impostato a 0 per evitare danni negativi. Viene sottratto il danno calcolato dalla vita del mostro viene attribuita l'invincibilità al mostro per un determinato periodo di tempo e viene chiamato il metodo damageReaction per far reagire il mostro all'attacco del giocatore, se la vita di quest'ultimo diventa 0 la variabile dying viene impostata su true per indicare che il mostro sta morendo, viene aggiunto un messaggio nell'interfaccia utente per informare il giocatore che ha ucciso il mostro e riceve una certa quantità di punti esperienza (exp). Successivamente, viene chiamato il metodo checkLevel per controllare se il giocatore ha raggiunto un nuovo livello in base all'esperienza accumulata.

5.5.19 metodo damageProjectile

Il metodo controlla se mentre si sta attaccando si colpisce un proiettile lanciato da un mostro se la condizione si avvera il proiettile viene distrutto e si generano delle particelle.

5.5.20 metodo interactTile

All'interno del metodo viene controllato se la casella interattiva è distruttibile (destructible) e se il giocatore ha l'oggetto corretto per interagire con la casella interattiva (isCorrectItem). Se queste condizioni sono soddisfatte, vengono eseguite le seguenti operazioni:

- viene chiamato il metodo generateParticle per generare particelle o effetti speciali in seguito all'interazione con la casella interattiva. I parametri passati al metodo sono entrambi l'oggetto gp.iTile[gp.currentMap][i], che rappresenta la casella interattiva stessa;
- viene riprodotto un effetto sonoro associato alla casella interattiva chiamando il metodo playSE();
- viene sostituita la casella interattiva con la sua forma distrutta (getDestroyedForm()).

5.5.21 metodo checkLevel

Questo metodo controlla se il giocatore ha raggiunto l'esperienza necessaria per salire di livello e, se sì, esegue l'aumento di livello con le relative modifiche ai valori di caratteristiche e statistiche del giocatore infine resetta l'esperienza attuale a 0.

5.5.22 metodo selectItem

All'interno del metodo, viene prima ottenuto l'indice dell'oggetto selezionato tramite il metodo getItemIndex() dell'interfaccia utente (ui). L'indice viene confrontato con la dimensione dell'inventario per verificare se l'oggetto selezionato esiste effettivamente nell'inventario. Se l'indice è valido, viene recuperato l'oggetto corrispondente dall'inventario utilizzando il metodo get della lista inventory. Successivamente, viene effettuato un controllo sul tipo di oggetto selezionato utilizzando la proprietà type dell'oggetto:

- se l'oggetto è di tipo "Wrod", "Irod" o "Grod" viene assegnato come arma corrente (currentWeapon) e viene ricalcolato il valore dell'attacco (attack) richiamando il metodo getAttack(). Viene inoltre chiamato il metodo getPlayerAttackImage() per ottenere l'immagine dell'attacco del giocatore in base all'arma posseduta e selezionata;
- se l'oggetto è di tipo "Wshield", "Ishield" o "Gshield", viene assegnato come scudo corrente (currentShield) e viene ricalcolato il valore della difesa (defense) richiamando il metodo getDefense().
- se l'oggetto è di tipo "tool", viene assegnato come attrezzo corrente (currentTool);
- se l'oggetto è di tipo "consumable" (presumibilmente oggetti consumabili), viene utilizzato chiamando il metodo use(this) sull'oggetto selezionato, che può avere effetti sul personaggio del giocatore. Successivamente, l'oggetto viene rimosso dall'inventario utilizzando il metodo remove della lista inventory;

5.5.23 metodo searchItemInInventory

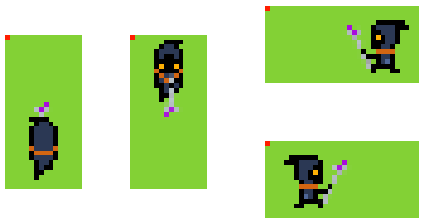
Il metodo scansiona l'inventario del giocatore e controlla al momento che si raccoglie un oggetto se questo è gia contenuto all'interno dell'inventario.

5.5.24 metodo canObtainItem

Il metodo verifica se è possibile ottenere un oggetto specifico (item) e restituisce un valore booleano che indica se è possibile o meno. All'inizio del metodo, viene inizializzata una variabile booleana canObtain con il valore false. Questa variabile sarà utilizzata per indicare se è possibile ottenere l'oggetto o meno. Successivamente, viene controllato se l'oggetto è impilabile (item.stackable == true). Se lo è, viene cercato l'oggetto nell'inventario utilizzando il metodo searchItemInInventory() che restituisce l'indice dell'oggetto nell'inventario. Se l'indice non è 999 (ovvero l'oggetto è presente nell'inventario), viene incrementato il valore amount dell'oggetto corrispondente e la variabile canObtain viene impostata su true. Se l'oggetto non è presente nell'inventario, viene controllato se l'inventario ha ancora spazio disponibile (inventory.size() != maxInventorySize). Se c'è spazio disponibile, l'oggetto viene aggiunto all'inventario e la variabile canObtain viene impostata su true. Se l'oggetto non è impilabile, viene controllato se c'è spazio disponibile nell'inventario e, se sì, l'oggetto viene aggiunto e la variabile canObtain viene impostata su true. Infine, viene restituito il valore di canObtain, che indica se è stato possibile ottenere l'oggetto o meno.

5.5.25 metodo draw

All'interno del metodo, viene inizializzata una variabile image di tipo BufferedImage a null. Successivamente, viene salvata la posizione corrente del personaggio sullo schermo nelle variabili tempScreenX e tempScreenY. Viene quindi eseguito uno switch sulla direzione del personaggio:



in tutte e quattro le direzioni si controlla se il personaggio sta attaccando in caso positivo si sottrae a screenY la dimensione di una cella nei casi "up" e "left" dato che l'angolo con coordinate x = 0, y = 0 dell'immagine caricata è quello in alto a sinistra quindi sarà necessario rimuovere il valore di una cella (gp.tileSize) alle variabili tempScreenX nel caso si stia attaccando verso l'altro e tempScreenY nel caso si stia attaccando verso sinistra. Successivamente, viene applicato un effetto grafico di invincibilità, se la variabile invincible è impostata su true. Viene utilizzato il metodo setComposite di g2 per impostare l'opacità dell'immagine a 0.4f.

5.6 Projectile.java

La classe Projectile estende ancora la classe Entity e rappresenta un proiettile nel gioco. Ha un attributo aggiuntivo user di tipo Entity che rappresenta l'entità che ha lanciato il proiettile.

5.6.1 Costruttore

Il costruttore della classe Projectile accetta un parametro gp di tipo GamePanel e chiama il costruttore della classe padre Entity passando il parametro gp.

5.6.2 metodo set

Il metodo set viene utilizzato per impostare le informazioni iniziali del proiettile, come la posizione, la direzione, lo stato di vita e l'entità che l'ha lanciato.

5.6.3 metodo update

Il metodo viene chiamato per aggiornare il proiettile ad ogni ciclo di gioco. Controlla se il proiettile è stato lanciato dal giocatore o da un'altra entità. Se è stato lanciato dal giocatore, controlla se colpisce un mostro e in tal caso danneggia il mostro, genera particelle e imposta lo stato del proiettile su "non vivo". Se è stato lanciato da un'altra entità, controlla se entra in contatto con il giocatore e in tal caso danneggia il giocatore, genera particelle e imposta lo stato del proiettile su "non vivo". Aggiorna anche la posizione del proiettile in base alla sua direzione e decrementa la vita del proiettile. Se la vita del proiettile raggiunge o va al di sotto di zero, imposta lo stato del proiettile su "non vivo".

5.6.4 metodo haveResource, subtractResource

I metodi sono sovraccaricati nelle classi specifiche come Rock.java e Fireball.java

6 Pacchetto environment

6.1 EnvironmentManager.java

La classe EnvironmentManager, è responsabile della gestione dell'ambiente di gioco, come alternanza giorno-notte e effetti luce.

6.1.1 Costruttore

Il costruttore EnvironmentManager(GamePanel gp) prende un oggetto GamePanel come parametro e lo assegna alla variabile gp.

6.1.2 metodo setup

Il metodo setup() inizializza il componente lighting creando una nuova istanza della classe Lighting e passando l'oggetto gp come parametro.

6.1.3 metodo update

Il metodo update() richiama il metodo update() del componente lighting per aggiornare lo stato dell'illuminazione nell'ambiente di gioco.

6.1.4 metodo draw

Il metodo draw(Graphics2D g2) richiama il metodo draw(Graphics2D g2) del componente lighting per disegnare l'illuminazione nell'ambiente di gioco.

6.2 Lighting.java

La classe fornisce le basi per gestire il ciclo giorno/notte nell'ambiente di gioco, con la possibilità di applicare un filtro di oscurità per creare l'effetto di luce ambientale.

6.2.1 Costruttore

Il costruttore della classe Lighting accetta un oggetto GamePanel come argomento e lo assegna alla variabile gp per tenerne traccia. Inoltre, chiama il metodo setLightSource() per impostare la sorgente di luce dell'illuminazione.

6.2.2 metodo setLightSource

Il metodo setLightSource() crea un'immagine bufferizzata di dimensioni gp.screenWidth per la larghezza e gp.screenHeight per l'altezza, utilizzando il tipo di immagine BufferedImage.TYPE_INT_ARGB. Quindi ottiene un oggetto Graphics2D dalla buffered image. Successivamente, calcola il centro (coordinate x e y) del cerchio di luce basandosi sulla posizione del giocatore (gp.player.screenX e gp.player.screenY) e sulle dimensioni di una cella (gp.tileSize). Se gp.player.currentLight è null, crea un array di colori (color) e un array di frazioni (fraction) per definire la transizione del colore nel cerchio di luce. Vengono specificati valori predefiniti per le frazioni (fractionValues) e l'opacità (opacityValues). Successivamente, viene creato un oggetto RadialGradientPaint[19] con il centro e le dimensioni del cerchio di luce, le frazioni e i colori specificati, e viene impostato come paint per il Graphics2D. Se gp.player.currentLight non è null, viene seguito lo stesso procedimento, ma la dimensione del cerchio di luce viene presa dalla variabile gp.player.currentLight.lightRadius. Infine, viene disegnato un rettangolo pieno con il paint selezionato nel Graphics2D, che copre l'intera immagine bufferizzata. Infine, viene rilasciata la risorsa del Graphics2D chiamando il metodo dispose().

6.2.3 metodo resetDay

Semplicemente reimposta il giorno a "day" e resetta il filtro luce.

[19] RadialGradientPaint è una classe in Java che rappresenta una pittura a gradiente radiale. Essa crea un gradiente che si espande da un punto centrale verso l'esterno, formando una forma circolare o ellittica di transizione dei colori.

6.2.4 metodo update

Il metodo update() della classe Lighting gestisce l'aggiornamento delle impostazioni di illuminazione in base allo stato del giorno. Se la variabile gp.player.lightUpdated è impostata su true, viene chiamato il metodo setLightSource() per aggiornare la sorgente di luce in base alla posizione del giocatore. Successivamente, gp.player.lightUpdated viene impostata su false per indicare che l'aggiornamento è stato eseguito. Se lo stato del giorno è "day", il contatore del giorno viene incrementato. Se il contatore supera il valore di 3600 (che rappresenta un periodo di tempo), lo stato del giorno viene impostato su "dusk" e il contatore viene reimpostato a 0. Se lo stato del giorno è "dusk", viene aumentato gradualmente l'alpha del filtro di illuminazione (filterAlpha) di 0.001f. Quando filterAlpha raggiunge il valore massimo di 1, lo stato del giorno viene impostato su "night". Se lo stato del giorno è "night", il contatore del giorno viene incrementato. Se il contatore supera il valore di 3600, lo stato del giorno viene impostato su "dawn" e il contatore viene reimpostato a 0. Se lo stato del giorno è "dawn", viene diminuito gradualmente l'alpha del filtro di illuminazione (filterAlpha) di 0.001f. Quando filterAlpha raggiunge il valore minimo di 0, lo stato del giorno viene reimpostato su "day".

6.2.5 metodo draw

Il metodo draw() si occupa di disegnare l'effetto di illuminazione sullo schermo, insieme al testo che indica lo stato del giorno.

7 Pacchetto monster

7.1 MON_Bat

La classe MON_Bat rappresenta un mostro di tipo "Pipistrello" nel gioco. La classe estende la classe Entity e contiene le caratteristiche specifiche del mostro.

7.1.1 Costruttore

7.1.2 metodo getImage

7.1.3 metodo setAction

Semplicemente viene richiamato il metodo per dare al pipistrello un movimento casuale con un intervallo di 10 quindi cambierà direzione casualmente in maniera molto frenetica.

7.1.4 metodo damageReaction

Resetta la variabile che determina il cambiamento di direzione.

7.1.5 metodo checkDrop

Il metodo gestisce il drop rate del mostro nel momento in cui la sua vita è inferiore o uguale a 0, utilizzando un generatore casuale di numeri "i" da 1 a 100, in termini di percentuali viene impostato che :

- si ha il 30% di possibilità di ottenere una moneta;

- si ha il 20% di possibilità di ottenere una pozione di mana;
- si ha il 20% di possibilità di ottenere una pozione di salute;
- si ha il 10% di possibilità di ottenere un minerale in ferro;
- si ha il 5% di possibilità di ottenere un minerale in oro;
- si ha il 3% di possibilità di ottenere un minerale di diamante;
- si ha il 11% di possibilità di ottenere una borsa di monete;

7.2 MON_Boss

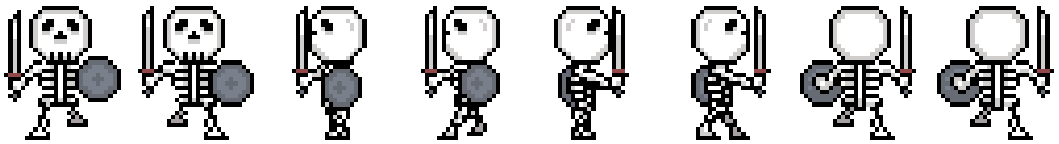
La classe MON_Boss rappresenta il boss finale del gioco. La classe estende la classe Entity e contiene le caratteristiche specifiche del mostro.

7.2.1 Costruttore

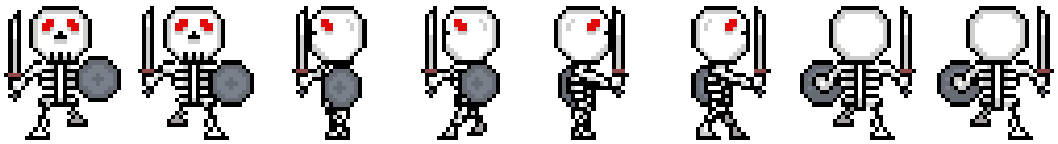
Il costruttore inizializza le proprietà del Boss come il nome, la velocità, la vita, l'attacco, la difesa e l'esperienza. Imposta anche l'area solida del pipistrello e ottiene l'immagine corrispondente per il movimento e per l'attacco.

7.2.2 metodo getImage

Il metodo carica le immagini, ingrandite di 5 volte rispetto alla norma, del mostro per tutte e 4 le direzioni.

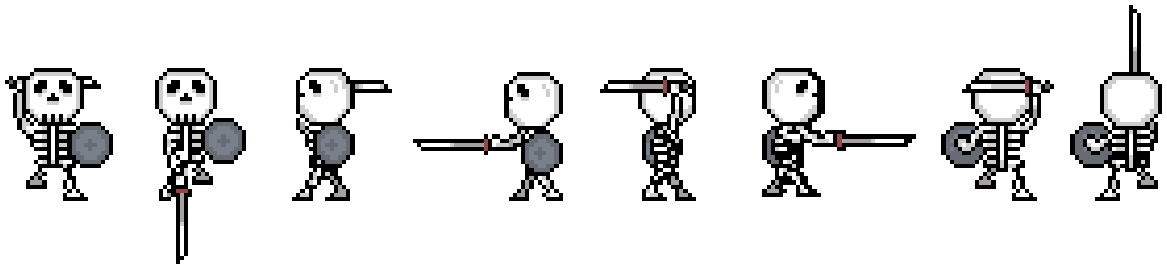


Il mostro, dopo aver perso metà della sua vita entra in fase 2:

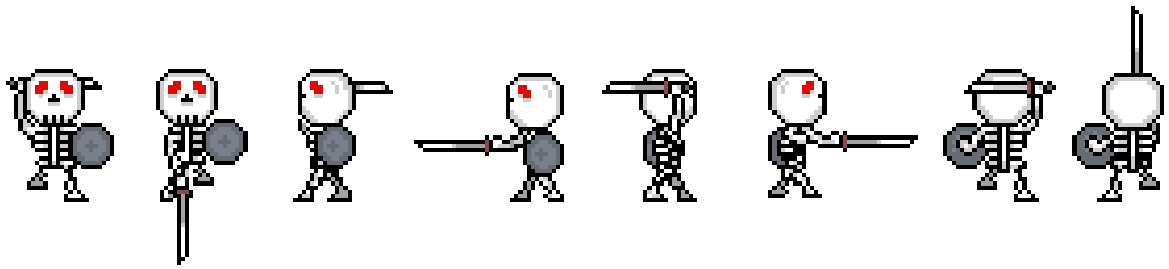


7.2.3 getAttackImage

Il metodo carica le immagini, ingrandite di 5 volte rispetto alla norma, del mostro per tutte e 4 le direzioni nel caso in cui stia attaccando.



Di seguito sono riportate le immagini di attacco della fase 2 del mostro.



7.2.4 metodo setAction

Controlla se il boss non è in uno stato di "rage" (furia) e la sua vita è inferiore alla metà della vita massima. In tal caso, imposta il flag inRage su true, ottiene l'immagine corrispondente alla furia (getImage()) e l'immagine dell'attacco (getAttackImage()). Aumenta anche la velocità di base (defaultSpeed) di uno, reimposta la velocità corrente (speed) sulla velocità di base e aumenta l'attacco (attack) di metà del suo valore attuale. Verifica se la distanza del boss dal giocatore (gp.player) è inferiore a 10 tile. In tal caso, chiama il metodo moveToPlayer(60) per muoversi verso il giocatore con una durata di movimento di 60 millisecondi. Se la distanza non è inferiore a 10 tile, chiama il metodo getRandomDirection(120) per ottenere una direzione casuale con una durata di 120 millisecondi. Verifica se il boss non sta attaccando (attacking == false). In tal caso, chiama il metodo isAttackingOrNot(60, gp.tileSize * 7, gp.tileSize * 5) per determinare se attaccare o meno. L'attacco avviene se il giocatore si trova all'interno di un'area specifica del boss definita dalle dimensioni specificate.

7.2.5 metodo damageReaction

Resetta la variabile che determina il cambiamento di direzione.

7.3 MON_Orc

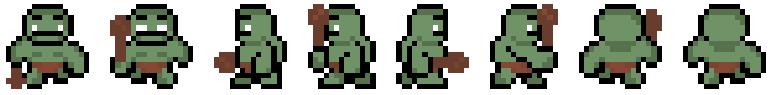
La classe MON_Orc rappresenta un mostro di tipo "Orco Verde" nel gioco. La classe estende la classe Entity e contiene le caratteristiche specifiche del mostro.

7.3.1 Costruttore

Inizializza le variabili specifiche dell'orco verde, come il tipo (type_mon), il nome ("Orco Verde"), la velocità predefinita (defaultSpeed), la velocità corrente (speed), la vita massima (maxLife), la vita corrente (life), l'attacco (attack), la difesa (defense), l'esperienza assegnata quando viene sconfitto (exp) e la potenza di respinta (knockBackPower). Definisce l'area solida dell'orco verde, che determina la sua collisione con gli oggetti nel gioco. L'area solida è un rettangolo con coordinate e dimensioni specifiche. Imposta la durata dei movimenti specifici dell'orco verde (motion1_duration e motion2_duration). Chiama i metodi getImage() e getAttackImage() per ottenere le immagini dell'orco verde e dell'attacco dell'orco verde, rispettivamente.

7.3.2 metodo getImage

Il metodo carica le immagini scalate del mostro per tutte e 4 le direzioni.



7.3.3 getAttackImage

Il metodo carica le immagini per l'animazione di attacco del mostro per tutte e 4 le direzioni.

7.3.4 metodo setAction

Controlla se l'orco verde sta inseguendo il giocatore (variabile onPath impostata su true). Se si controlla se il giocatore è lontano 10 case dal mostro in tal caso smette di seguirlo altrimenti avrà come obbiettivo quello di raggiungere il giocatore per colpirlo. Nel caso in cui il giocatore si riavvicinerà al mostro di almeno 5 case allora la variabile onPath ridiventerà true data dal metodo isChasing(), e ricomincerà a seguire il giocatore se sarà abbastanza vicino a lui, altrimeti il mostro riceverà una direzione casuale dettata dal metodo getRandomDirection().

7.3.5 metodo damageReaction

Imposta la variabile onPath su true, il mostro quindi cercherà di attaccare il giocatore e a inseguirlo.

7.3.6 metodo checkDrop

Il metodo gestisce il drop rate del mostro nel momento in cui la sua vita è inferiore o uguale a 0, utilizzando un generatore casuale di numeri "i" da 1 a 100, in termini di percentuali viene impostato che :

- si ha il 25% di possibilità di ottenere una moneta;
- si ha il 13% di possibilità di ottenere una pozione di salute;
- si ha il 13% di possibilità di ottenere una pozione di mana;
- si ha il 15% di possibilità di ottenere una borsa di monete;
- si ha il 3% di possibilità di ottenere una Staffa d'Oro;
- si ha il 3% di possibilità di ottenere uno Scudo in Diamante;
- si ha il 10% di possibilità di ottenere un minerale in ferro;
- si ha il 14% di possibilità di ottenere un minerale in oro;
- si ha il 3% di possibilità di ottenere un minerale in diamante;

7.4 MON_RedSlime.java

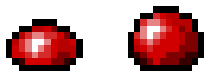
La classe MON_RedSlime rappresenta un mostro di tipo "Slime Rosso" nel gioco. La classe estende la classe Entity e contiene le caratteristiche specifiche del mostro.

7.4.1 Costruttore

Il costruttore della classe MON_RedSlime inizializza le varie proprietà del mostro "Slime Rosso" con valori predefiniti come tipo, nome, velocità, vita massima, attacco e difesa ecc....

7.4.2 metodo getImage

Il metodo carica le immagini scalate del mostro per tutte e 4 le direzioni.



7.4.3 metodo setAction

Il metodo gestisce il comportamento del mostro, cotrolla se onPath è true, quindi se è stato colpito, in tal caso il mostro riceverà come coordinate da raggiungere la posizione del giocatore nella mappa e iniziera a lanciare delle rocce come proiettili. Per evitare che colpisca a raffica vengono impostati dei limiti infatti il mostro potrà sparare solo se il numero generato casualmente da un generatore casuale da 1 a 200 sarà maggiore di 198, se c'è un proiettile alive non ne potrà sparare un altro, e se il contatore, che funge da "cooldown" non sarà uguale a 30. Se onPath è falso quindi il mostro non è stato triggerato dal giocatore compierà dei movimenti casuali verso le 4 direzioni.

7.4.4 metodo damageReaction

Imposta onPath su true in modo che possa iniziare a seguire il giocatore e colpirlo.

7.4.5 metodo checkDrop

Il metodo gestisce il drop rate del mostro nel momento in cui la sua vita è inferiore o uguale a 0, utilizzando un generatore casuale di numeri "i" da 1 a 100, in termini di percentuali viene impostato che :

- si ha il 25% di possibilità di ottenere una moneta;
- si ha il 10% di possibilità di ottenere una pozione di mana;
- si ha il 10% di possibilità di ottenere una pozione di salute;
- si ha il 15% di possibilità di ottenere una borsa di monete;
- si ha il 5% di possibilità di ottenere una Staffa in Ferro;
- si ha il 5% di possibilità di ottenere uno Scudo in Ferro;
- si ha il 15% di possibilità di ottenere dei frammenti di ferro;
- si ha il 10% di possibilità di ottenere dei frammenti di oro;
- si ha il 5% di possibilità di ottenere dei frammenti di diamante;

8 Pacchetto object

8.1 OBJ_Blue_Potion.java

La classe rappresenta un oggetto di gioco specifico, ovvero una "Pozione di Mana" di colore blu. Estende la classe Entity, che fornisce funzionalità di base per gli oggetti nel gioco.

8.1.1 Costruttore

Nel costruttore della classe, viene chiamato il costruttore della classe padre (super(gp)) per inizializzare gli attributi ereditati dalla classe Entity. Viene impostato il tipo dell'oggetto (type_consumable), il nome ("Pozione di Mana"), il valore (value) a 2 e viene caricata l'immagine dell'oggetto utilizzando il metodo setup() che carica l'immagine da un file. Viene inoltre impostata una descrizione (description) dell'oggetto e un prezzo di 10.

8.1.2 metodo use

Il metodo viene richiamato quando l'oggetto in questione viene utilizzato in questo caso viene stampato a schermo un dialogo che indica che la pozione è stata bevuta e che viene ripristinato una certa quantità di mana.

8.1.3 metodo setDialogue

Il metodo inizializza i dialoghi per l'oggetto corrente.

8.2 OBJ_Bronze_Key.java

La classe rappresenta una "Chiave di Bronzo" nel gioco.

8.2.1 Costruttore

Nel costruttore si definiscono nome dell'oggetto("Chiave di Bronzo"), viene caricata l'immagine utilizzando il metodo setup() e viene aggiunta una descrizione dell'oggetto.

8.2.2 metodo use

Il metodo gestisce l'evento di utilizzo della chiave, se utilizzata davanti una porta quest'ultima viene aperta e la chiave scompare dall'inventario altrimenti mostra un dialogo di errore.

8.3 OBJ_Chest.java

La classe rappresenta gli oggetti "Cassa" nel gioco. Viene definito un oggetto Entity che sarà il loot ricevuto dopo aver aperto la cassa e una variabile booleana che indica se la cassa è aperta o meno.

8.3.1 Costuttore

Il costruttore della classe crea un oggetto di tipo cassa nel gioco, impostando le sue caratteristiche iniziali come il tipo, il nome, l'immagine e l'area solida.

8.3.2 metodo setLoot

Override del metodo setLoot() nella classe Entity.

8.3.3 metodo interact

Questo metodo gestisce l'interazione con la cassa nel gioco. Quando il metodo viene chiamato, viene impostato lo stato di gioco a dialogueState. Se la cassa non è stata ancora aperta (opened == false), viene riprodotto un effetto sonoro ad indicare l'apertura della cassa. Successivamente, viene verificato se il giocatore può ottenere l'oggetto loot utilizzando il metodo canObtainItem() della classe gp.player. Se il giocatore non può ottenere l'oggetto (ovvero l'inventario è pieno), viene aggiunto al messaggio di dialogo l'indicazione che non puoi trasportare altro. Altrimenti, se il giocatore può ottenere l'oggetto, viene aggiunto al messaggio di dialogo l'indicazione che hai ottenuto l'oggetto loot.name e viene impostata l'immagine della cassa aperta (down1 = image2) e il flag opened viene impostato su true. Se la cassa è già stata aperta (opened == true), il messaggio di dialogo sarà semplicemente "E' vuota...".

8.3.4 metodo setDialogue

Il metodo inizializza i dialoghi per l'oggetto corrente.

8.4 OBJ_Coin_Bag.java

La classe rappresenta una "Borsa di Monete" nel gioco.

8.4.1 Costuttore

Nel costruttore della classe vengono definiti tipo dell'oggetto, nome, un valore che oscilla da 1 a 5 e viene caricata l'immagine dell'oggetto.

8.4.2 metodo use

Se usato l'oggetto aggiunge al giocatore delle monete che variano in base al valore di value definito nel costruttore e viene mostrato a schermo un dialogo di gioco che indica quante monete ha ricevuto dall'apertura della sacca.

8.4.3 metodo setDialogue

Il metodo inizializza i dialoghi per l'oggetto corrente.

8.5 OBJ_Coin.java

La classe rappresenta l'oggetto "Moneta" nel gioco.

8.5.1 Costuttore

Nel costruttore della classe vengono definiti tipo dell'oggetto, nome, un valore pari a 1 e viene caricata l'immagine dell'oggetto.

8.5.2 metodo use

Il metodo viene chiamato nel momento in cui si raccoglie la moneta da terra, emette un effetto sonoro e mostra a schermo un messaggio di gioco che indica che la moneta è stata raccolta.

8.6 OBJ_Diamond_Ore.java

8.6.1 Costruttore

8.7 OBJ_Diamond_Shield.java

La classe definisce uno degli scudi presenti nel gioco.

8.7.1 Costuttore

l costruttore della classe crea un'istanza del dello scudo e configura le sue proprietà: il tipo, il nome, l'immagine, il valore di difesa, la descrizione e il prezzo.

8.8 OBJ_Door.java

La classe rappresenta le gli oggetti "Porta" nel gioco.

8.8.1 Costuttore

Nel costruttore della classe vengono definiti il tipo di oggetto, il nome, l'immagine dell'oggetto, e la collisione, inoltre viene definita la parte solida.

8.8.2 metodo interact

Il metodo definisce cosa deve accadere quando si preme il tasto "ENTER" davanti la porta, semplicemente mostra un messaggio che indica la necessità di una chiave per aprire la porta.

8.8.3 metodo setDialogue

Il metodo inizializza i dialoghi per l'oggetto corrente.

8.9 OBJ_Fireball.java

La classe rappresenta il colpo "Palla di Fuoco" che il giocatore può sparare alla pressione del tasto "SPACE" in gioco.

8.9.1 Costuttore

All'interno del costruttore, vengono impostati i dettagli dell'oggetto, come il nome "Palla di Fuoco", la velocità 5, la vita massima 80, l'attacco 2 e il costo d'uso 1. Inizialmente, lo stato di vita dell'oggetto viene impostato su false e viene chiamato il metodo getImage() per ottenere l'immagine associata alla palla di fuoco.

8.9.2 metodo getImage

Ottiene le immagini nelle 4 direzioni per la palla di fuoco mediante il metodo setup().

8.9.3 metodo haveResource

Il metodo controlla se l'utilizzatore del colpo ha abbastanza risorse (mana) per lanciare la palla di fuoco.

8.9.4 metodo subtractResoure

Il metodo sottrae le risorse (mana) dall'utilizzatore.

8.9.5 metodi getParticleColor, getParticleSize, getParticleSpeed, getParticleMaxLife

I metodi impostano il colore, la grandezza, la velocità e la vita massima delle particelle generate allo schiantarsi di una palla di fuoco.

8.10 OBJ_Gate.java

La classe rappresenta un oggetto "Cancello" che blocca l'accesso al giocatore ad una delle zone nella mappa.

8.10.1 Costruttore

Il costruttore della classe OBJ_Gate inizializza l'oggetto "Cancello" nel gioco. Esso imposta il tipo dell'oggetto come "obstacle" e il nome utilizzando un valore specifico. Carica le immagini associate all'oggetto gate e abilita la collisione. Definisce l'area solida dell'oggetto e salva le coordinate di default. Infine, imposta il dialogo associato all'oggetto.

8.10.2 metodo setDialogue

Il metodo inizializza i dialoghi per l'oggetto corrente.

8.10.3 metodo interact

Il metodo avvia i dialoghi dopo aver interagito con il cancello.

8.11 OBJ_Gold_Ore.java

8.11.1 Costruttore

8.12 OBJ_Golden_Rod.java

La classe definisce una delle armi presenti nel gioco.

8.12.1 Costuttore

Il costruttore della classe crea un'istanza del bastone d'oro e configura le sue proprietà, comprese le dimensioni dell'area di attacco, il tipo, il nome, l'immagine, il valore di attacco, la descrizione, il prezzo e la potenza di knockback.

8.13 OBJ_Heart.java

La classe definisce gli oggetti di nome "Cuore" nel gioco.

8.13.1 Costuttore

Il costruttore carica in memoria le immagini che rappresentano i cuori nel gioco.

8.14 OBJ_Iron_Key.java

La classe rappresenta una "Chiave in Ferro" nel gioco.

8.14.1 Costruttore

Nel costruttore si definiscono nome dell'oggetto("Chiave di Ferro"), viene caricata l'immagine utilizzando il metodo setup() e viene aggiunta una descrizione dell'oggetto. La chiave può essere utilizzata per aprire un cancello.

8.14.2 metodo setDialogue

Imposta i dialoghi da utilizzare nei vari casi di utilizzo della chiave.

8.14.3 metodo use

Il metodo use controlla se la chiave può essere utilizzata per aprire un cancello nelle vicinanze e gestisce le azioni appropriate di dialogo e rimozione del cancello.

8.15 OBJ_Iron_Ore.java

8.15.1 Costruttore

8.16 OBJ_Iron_Rod.java

La classe definisce una delle armi presenti nel gioco.

8.16.1 Costuttore

Il costruttore della classe crea un'istanza del bastone in ferro e configura le sue proprietà, comprese le dimensioni dell'area di attacco, il tipo, il nome, l'immagine, il valore di attacco, la descrizione, il prezzo e la potenza di knockback.

8.17 OBJ_Iron_Shield.java

La classe rappresenta uno scudo presente nel gioco.

8.17.1 Costuttore

Il costruttore della classe crea un'istanza del dello scudo e configura le sue proprietà: il tipo, il nome, l'immagine, il valore di difesa, la descrizione e il prezzo.

8.18 OBJ_Lantern.java

La lanterna è un oggetto che emette luce nell'ambiente di gioco.

8.18.1 Costruttore

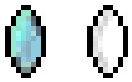
Nel costruttore, viene chiamato il costruttore della classe padre (Entity) passando il riferimento al pannello di gioco. Vengono impostati anche il tipo dell'oggetto (type_light), il nome, l'immagine dell'oggetto nella direzione "down1" e una descrizione. Viene inoltre specificato un prezzo e il raggio di luce della lanterna.

8.19 OBJ_Mana_Crystal.java

La classe definisce gli oggetti di nome "Mana" nel gioco.

8.19.1 Costuttore

Il costruttore carica in memoria le immagini che rappresentano i cristalli di mana nel gioco.



8.20 OBJ_Null.java

La classe è utilizzata nel gioco per rappresentare uno slot non utilizzato nell'inventario del giocatore.

8.20.1 Costuttore

Nel costruttore vengono inizializzate le variabili tipo, nome, viene caricata l'immagine dell'oggetto e una descrizione. L'oggetto è utilizzato come placeholder per occupare momentaneamente lo slot dedicato ai tool.

8.21 OBJ_Red_Potion.java

La classe rappresenta un oggetto di gioco specifico, ovvero una "Pozione di Salute" di colore rosso. Estende la classe Entity, che fornisce funzionalità di base per gli oggetti nel gioco.

8.21.1 Costuttore

Nel costruttore della classe, viene chiamato il costruttore della classe padre (super(gp)) per inizializzare gli attributi ereditati dalla classe Entity. Viene impostato il tipo dell'oggetto (type_consumable), il nome ("Pozione di Salute"), il valore (value) a 2 e viene caricata l'immagine dell'oggetto utilizzando il metodo setup() che carica l'immagine da un file. Viene inoltre impostata una descrizione (description) dell'oggetto e un prezzo di 10.

8.21.2 metodo setDialogue

Imposta i dialoghi ad utilizzare quando la pozione è utilizzata.

8.21.3 metodo use

Il metodo viene richiamato quando l'oggetto in questione viene utilizzato in questo caso viene stampato a schermo un dialogo che indica che la pozione è stata bevuta e che viene ripristinato una certa quantità di salute.

8.22 OBJ_Rock.java

La classe rappresenta l'oggetto proiettile lanciato dal mostro "Red Slime" della classe Entity.

8.22.1 Costuttore

Nel costruttore vengono definite variabili come nome dell'oggetto, velocità del proiettile, vita del proiettile, il danno che infligge e la variabile alive viene impostata su false. Inoltre si caricano le immagini ottenute dal metodo getImage().

8.22.2 metodo getImage

Il metodo carica le immagini della Roccia per tutte e 4 le direzioni.

8.22.3 metodo getParticleColor, getParticleSize, getParticleSpeed, getParticleMaxLife

I metodi ricevono le informazioni necessarie riguardanti colore, grandezza, velocità e vita delle particelle generate dallo schianto della roccia contro il giocatore.

8.23 OBJ_Tent.java

La classe rappresenta un oggetto di tipo "Tenda" nel gioco. La tenda è un oggetto che consente al giocatore di dormire durante la notte.

8.23.1 Costruttore

Vengono inizializzate le variabili che indicano tipo dell'oggetto, nome, immagine, descrizione, prezzo e viene chiamato il metodo per impostare i dialoghi.

8.23.2 metodo setDialogue

Inizializza le stringhe di dialogo per la classe.

8.23.3 metodo use

Specifica cosa accade se la tenda viene utilizzata, può essere usata solo di notte, imposta lo stato di gioco su "sleepState", viene riprodotto un suono, ripristina salute e mana del giocatore e sostituisce l'attuale immagine del giocatore con l'immagine della tenda altrimenti se è giorno viene avviato il dialogo impostato in precedenza.

8.24 OBJ_Tree_Cutter.java

La classe indica il tool (attrezzo) specifico per abbattere le tile interattive IT_Dead_Tree.

8.24.1 Costruttore

Nel costruttore si definisce il tipo, il nome dell'oggetto, l'immagine, una piccola descrizione e il suo costo.

8.25 OBJ_Wooden_Rod.java

La classe rappresenta una delle armi disponibili in gioco.

8.25.1 Costruttore

Il costruttore della classe crea un'istanza del bastone di legno e configura le sue proprietà, comprese le dimensioni dell'area di attacco, il tipo, il nome, l'immagine, il valore di attacco, la descrizione, il prezzo e la potenza di knockback.

8.26 OBJ_Wooden_Shield.java

La classe rappresenta uno scudo presente nel gioco.

8.26.1 Costruttore

Il costruttore della classe crea un'istanza del dello scudo e configura le sue proprietà: il tipo, il nome, l'immagine, il valore di difesa, la descrizione e il prezzo.

9 Pacchetto tile

9.1 Map.java

La classe Map estende TileManager e rappresenta la mappa di gioco. Ha un riferimento al pannello di gioco (GamePanel) e un array di immagini (worldMap) per la mappa del mondo. La variabile booleana miniMapOn indica se la minimappa è attivata o meno.

9.1.1 Costruttore

Il costruttore richiama il metodo per generare la mini mappa.

9.1.2 metodo createWorldMap

Il metodo crea la mappa del mondo in base alla posizione del giocatore. Inizializza l'array worldMap come un array di immagini, con una dimensione corrispondente alla larghezza e altezza della mappa del mondo. Successivamente, itera attraverso le colonne e le righe della mappa e disegna ogni tile sulla rispettiva immagine della mappa del mondo utilizzando il numero di tile corrispondente. Una volta completata la creazione della mappa del mondo, le risorse grafiche vengono rilasciate.

9.1.3 drawFullMapScreen

Il metodo disegna la schermata della mappa completa. Prima di tutto, viene riempito lo sfondo con il colore nero. Successivamente, viene disegnata l'immagine della mappa corrente utilizzando l'array worldMap nelle coordinate (x, y) con una larghezza e altezza predefinite. In seguito, viene calcolata la scala per adattare il giocatore alla dimensione della mappa e viene disegnata l'immagine del giocatore nelle coordinate corrispondenti nella mappa. Infine, viene visualizzato un suggerimento per chiudere la schermata della mappa.

9.2 Tile.java

La classe fornisce una struttura di base per la gestione delle tessere nel gioco, consentendo di specificare l'immagine e le proprietà di collisione per ciascuna tessera.

9.3 TileManager.java

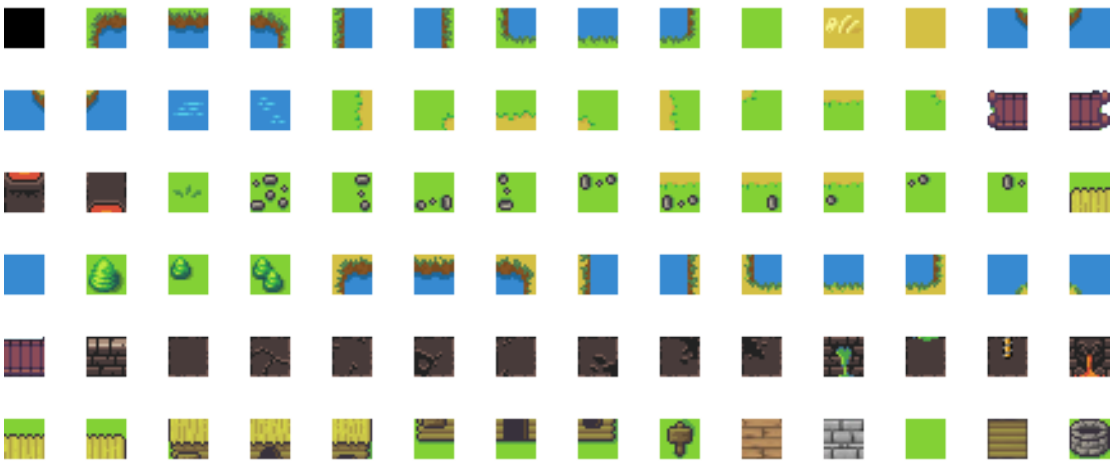
La classe TileManager gestisce le tessere del gioco. Contiene i seguenti attributi: gp di tipo GamePanel che rappresenta il pannello di gioco, tile che è un array di oggetti Tile per memorizzare tutte le tessere del gioco e mapTileNum che è una matrice tridimensionale di interi per memorizzare i numeri delle tessere per ciascuna mappa.

9.3.1 Costruttore

Il costruttore TileManager inizializza l'attributo gp con il pannello di gioco passato come parametro. Crea un array di oggetti Tile con una dimensione massima di 150 e una matrice mapTileNum con dimensioni basate sui valori di gp.maxMap, gp.maxWorldCol e gp.maxWorldRow. Successivamente, chiama il metodo getTileImage() per ottenere le immagini associate alle tessere e carica le mappe di gioco utilizzando i file di testo specificati.

9.3.2 metodo getTileImage

Il metodo richiama il metodo setup() per caricare le immagini delle celle che riceve come parametri un index che rappresenterà l'indice della cella nel file di testo della mappa, il nome dell'immagine e il parametro collisione che sarà vero se è una cella solida come muri o alberi o falso altrimenti. In basso tutte le tile utilizzate nel gioco:



9.3.3 metodo setup

Il metodo setup della classe TileManager viene utilizzato per configurare una tessera specifica. Prende come parametri un indice per identificare la posizione della tessera nell'array tile, il nome dell'immagine associata alla tessera e un valore booleano che indica se la tessera ha una collisione. All'interno del metodo, viene creato un oggetto UtilityTool per utilizzare alcune utility di supporto. Viene quindi eseguito un blocco try-catch per gestire eventuali eccezioni di I/O. Nel blocco try, viene istanziato un nuovo oggetto Tile all'indice specificato nell'array tile. Viene letto l'immagine corrispondente al percorso specificato nella cartella delle immagini delle tessere. L'immagine viene ridimensionata utilizzando il metodo scaleImage dell'oggetto uTool per adattarla alle dimensioni

desiderate (gp.tileSize x gp.tileSize). Infine, viene impostato il valore della proprietà collision della tessera in base al parametro fornito.

9.3.4 metodo loadMap

Il metodo è responsabile del caricamento di una mappa da un file di testo specificato tramite il percorso del file. Inizialmente, il metodo apre uno stream di input per leggere il file utilizzando getClass().getResourceAsStream(filePath), che restituisce un oggetto InputStream a partire dal percorso del file, successivamente, viene creato un oggetto BufferedReader per leggere il contenuto del file riga per riga utilizzando l'oggetto InputStreamReader per leggere i byte dall'input stream. All'interno di un ciclo while, viene letta una riga alla volta tramite il metodo readLine() del BufferedReader. All'interno di questo ciclo, la riga letta viene suddivisa in numeri utilizzando il metodo split(" "), che suddivide la stringa in base agli spazi e restituisce un array di stringhe contenente i singoli numeri come elementi. Successivamente, viene eseguito un ciclo for per iterare sui numeri estratti dalla riga. Ogni numero viene convertito in intero utilizzando Integer.parseInt() e assegnato all'elemento corrispondente dell'array mapTileNum per la mappa, la colonna e la riga correnti, il metodo continua a leggere le righe successive finché non vengono raggiunti i limiti massimi delle colonne gp.maxWorldCol o delle righe gp.maxWorldRow. Quando si raggiunge la fine di una riga, l'indice della colonna viene riportato a 0 e l'indice della riga viene incrementato.

9.3.5 metodo draw

Il metodo draw della classe TileManager si occupa di disegnare le tiles sulla schermata di gioco. All'interno di un ciclo while, il metodo itera su tutte le colonne e le righe della mappa. Per ogni posizione di piastrella nella mappa, vengono calcolate le coordinate nel mondo (worldX e worldY) e le coordinate sullo schermo (screenX e screenY) tenendo conto della posizione corrente del giocatore. Successivamente, per evitare di stampare a schermo caselle inutili che quindi utilizzerebbero solo risorse inutilmente, viene verificato se la piastrella si trova all'interno del rettangolo visibile dello schermo del giocatore. Se la piastrella è visibile, l'immagine della piastrella corrispondente al numero di piastrella tileNum viene disegnata alle coordinate screenX e screenY utilizzando il metodo drawImage di Graphics2D. Dopo il disegno della piastrella corrente, l'indice della colonna worldCol viene incrementato. Se l'indice della colonna raggiunge il limite massimo gp.maxWorldCol, l'indice della colonna viene riportato a 0 e l'indice della riga worldRow viene incrementato.

10 Pacchetto tile_interactive

10.1 InteractiveTile.java

La classe InteractiveTile ha un attributo gp di tipo GamePanel, che rappresenta il pannello di gioco corrente, e un attributo booleano destructible, che indica se la piastrella è distruttibile o meno.

10.1.1 Costruttore

Il costruttore InteractiveTile prende come argomenti il riferimento al pannello di gioco gp, oltre alla colonna col e alla riga row in cui si trova la piastrella. Inizializza l'attributo gp con il valore fornito.

10.1.2 metodo isCorrectItem

Il metodo isCorrectItem prende un'entità come argomento e verifica se è l'oggetto corretto per interagire con la piastrella. Restituisce un valore booleano che indica se l'oggetto è corretto o meno.

10.1.3 metodo playSE

Il metodo playSE è dichiarato ma non ha un'implementazione specifica nel codice fornito. Può essere sovrascritto nelle classi derivate per riprodurre un effetto sonoro specifico associato all'interazione con la piastrella.

10.1.4 metodo getDestroyedForm

Il metodo getDestroyedForm restituisce una nuova istanza di InteractiveTile che rappresenta la forma distrutta della piastrella. Nel codice fornito, il metodo restituisce semplicemente null, indicando che non è definita una forma distrutta specifica per questa piastrella. Questo metodo può essere sovrascritto nelle classi derivate per fornire una logica personalizzata per la forma distrutta della piastrella.

10.1.5 metodo update

Il metodo update è dichiarato ma non ha un'implementazione specifica nel codice fornito. Può essere sovrascritto nelle classi derivate per aggiornare lo stato della piastrella interattiva in base alle esigenze del gioco.

10.2 IT_Dead_Tree.java

La classe IT_Dead_Tree estende la classe InteractiveTile e rappresenta un tipo specifico di piastrella interattiva, in particolare un "albero secco" nel gioco che potrà essere rotto.

10.2.1 Costruttore

La classe IT_Dead_Tree estende la classe InteractiveTile e rappresenta un tipo specifico di piastrella interattiva, in particolare un albero morto nel gioco. Il costruttore IT_Dead_Tree prende come argomenti il riferimento al pannello di gioco gp, oltre alla colonna col e alla riga row in cui si trova la piastrella. Chiama il costruttore della classe genitore InteractiveTile passando i parametri corrispondenti. Inizializza l'attributo gp con il valore fornito e imposta le coordinate worldX e worldY in base alla posizione della piastrella nel mondo di gioco. Viene utilizzato il metodo setup per impostare l'immagine down1 dell'albero morto, fornendo il percorso dell'immagine e le dimensioni desiderate. Viene anche impostato l'attributo destructible su true, indicando che l'albero può essere distrutto.

10.2.2 metodo isCorrectItem

Il metodo isCorrectItem controlla se l'oggetto entity passato come parametro è l'oggetto corretto per interagire con l'albero morto. Restituisce true se l'oggetto corrente (currentTool) dell'entità è di tipo type_tool, altrimenti restituisce false. Questo implica che solo gli oggetti di tipo type_tool possono essere considerati corretti per interagire con l'albero morto. Altri tipi di oggetti non soddisfano la condizione e restituiranno false.

10.2.3 metodo playSE

Riproduce il suono nel momento in cui l'albero viene rotto.

10.2.4 metodo getDestroyedForm

Il metodo getDestroyedForm restituisce un nuovo oggetto InteractiveTile che rappresenta la forma distrutta dell'albero morto. Nella specifica implementazione, viene creato un oggetto IT_Trunk_Tree passando il GamePanel (gp) e le coordinate del mondo (worldX e worldY) divise per la dimensione del tile (gp.tileSize).

10.2.5 metodo getParticleColor, getParticleSize, getParticleSpeed, getParticleMaxLife

Metodi getter per ottenere le informazioni riguardanti le particelle da generare dopo la distruzione dell'albero.

10.3 IT_Trunk_Tree.java

La classe IT_Trunk_Tree è una sottoclasse di InteractiveTile che rappresenta la forma dell'albero distrutto.

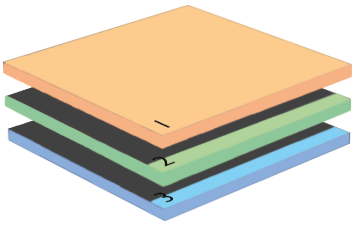
10.3.1 Costruttore

Nel suo costruttore, viene passato il GamePanel (gp) insieme alle coordinate del mondo (col e row). Le coordinate del mondo vengono utilizzate per impostare la posizione dell'albero distrutto nel mondo di gioco. Viene inoltre inizializzata l'area solida dell'albero distrutto con valori impostati in modo che non generi collisioni con il giocatore. Successivamente, l'immagine del tronco dell'albero distrutto viene caricata utilizzando il metodo setup() con il percorso dell'immagine e le dimensioni del tile (gp.tileSize).

11 Testing e Debug

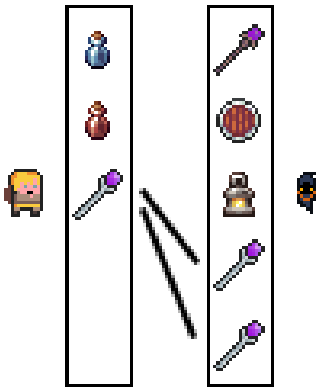
11.1 Ampliamento della mappa di gioco

Durante lo sviluppo del mio gioco in 2D basato sulla libreria grafica Graphics2D di Java, ho affrontato diverse sfide riguardanti la gestione delle mappe di gioco dopo aver introdotto mappe tridimensionali. Inizialmente, la mappa di gioco era rappresentata come una matrice, ovvero un array bidimensionale di numeri salvati su un file di testo. Il programma leggeva il file di testo e associava a ogni numero una specifica casella, rappresentata da un'immagine importata nel programma, che veniva quindi stampata sullo schermo. Tuttavia, con l'introduzione delle mappe tridimensionali, ho dovuto riadattare la struttura dell'array. Ho trasformato l'array bidimensionale in un array tridimensionale per gestire le diverse mappe presenti nel gioco. Ogni mappa è stata rappresentata come uno strato separato nella terza dimensione dell'array. Questo cambiamento ha richiesto una revisione del codice che leggeva e associava le immagini alle caselle sulla mappa. Ho dovuto adattare le iterazioni sugli array per lavorare con le nuove dimensioni, garantendo un accesso corretto ai dati delle diverse mappe. Inoltre, il riadattamento degli array bidimensionali e tridimensionali ha richiesto una modifica nella gestione delle collisioni. Ho dovuto rivedere l'algoritmo di rilevamento delle collisioni per considerare le nuove dimensioni dell'array e assicurarmi che il personaggio controllato dal giocatore interagisse correttamente con gli elementi sulla mappa, evitando collisioni indesiderate.



11.2 Problemi relativi agli oggetti acquistati

Durante lo sviluppo del mio gioco, ho riscontrato un problema relativo agli oggetti acquistati dal mercante. Nel caso in cui un giocatore selezionasse uno o più oggetti equipaggiabili, come spade o scudi, il cursore segnalava erroneamente che tutti gli oggetti dello stesso tipo acquistati dal mercante fossero equipaggiati contemporaneamente. Questo problema si verificava perché il gioco generava un oggetto condiviso invece di oggetti distinti per ciascun acquisto. Per investigare ulteriormente, ho deciso di testare il sistema introducendo temporaneamente la durabilità delle armi. Quando ho utilizzato le armi, ho notato che la durabilità di tutte le armi acquistate dal mercante aveva lo stesso valore, indicando che tutte condividevano le stesse proprietà di gioco. Questo non era intenzionale e andava contro il concetto di oggetti unici nel gioco. Per risolvere il problema, ho creato una nuova classe chiamata "Entity Generator" (Generatore di Entità). Questa classe aveva il compito di creare un nuovo oggetto con un nome specifico, garantendo che ogni oggetto avesse le proprie proprietà e non condividesse più i "parametri di gioco". Ogni volta che il giocatore acquistava un oggetto equipaggiabile dal mercante, il generatore di entità creava un nuovo oggetto unico con il nome specifico desiderato.



Dopo aver implementato questa soluzione, ho eseguito ulteriori test e ho riscontrato che il problema era stato risolto con successo. Ora, quando il giocatore acquistava oggetti equipaggiabili dal mercante, ciascun oggetto era unico e aveva le proprie proprietà individuali, inclusa la durabilità delle armi.

11.3 Problemi audio

Durante la fase di testing, ho riscontrato alcuni problemi riguardanti gli audio di gioco. Era fondamentale posizionare correttamente gli effetti sonori e la colonna sonora per garantire un'esperienza coinvolgente. Inizialmente, gli audio non erano sincronizzati correttamente con gli eventi del gioco o venivano riprodotti nel momento sbagliato. Per risolvere questi problemi, ho riveduto il sistema audio e ho apportato le necessarie modifiche. Ho assicurato che gli effetti sonori fossero posizionati correttamente nelle diverse mappe e che si attivassero nei momenti opportuni, come ad esempio quando il personaggio interagisce con oggetti o quando si verificano eventi specifici nel gioco. Inoltre, ho ampliato la libreria di audio, aggiungendo nuovi suoni per arricchire l'esperienza di gioco e creare un'atmosfera più coinvolgente.
Successivamente, ho effettuato test approfonditi per verificare che gli audio fossero riprodotti correttamente e che si integrassero perfettamente con gli altri aspetti del gioco.

11.4 Problemi minori

11.4.1 Errore nella vendita di oggetti

Durante lo sviluppo del mio gioco, ho riscontrato un errore minore riguardante la vendita degli oggetti. L'errore si verificava a causa di un problema nel codice che gestiva il processo di vendita, il quale impediva al giocatore di vendere gli oggetti correttamente. Nel dettaglio, il problema risiedeva nel ciclo che ottenne gli indici degli oggetti nell'inventario. Invece di iterare correttamente l'inventario del giocatore, il ciclo scorreva erroneamente l'inventario del mercante. Ciò significava che se il mercante aveva, ad esempio, tre oggetti nel suo inventario, il giocatore non era in grado di vendere gli oggetti a partire dal quarto in poi, poiché il codice non riconosceva l'esistenza di quegli oggetti. Per risolvere questo errore, ho apportato una correzione al codice che gestiva la vendita degli oggetti. Ho modificato il ciclo in modo che iterasse correttamente l'inventario del giocatore, consentendo al giocatore di vendere gli oggetti dal suo inventario personale senza alcuna limitazione.

Dopo aver apportato questa correzione, ho eseguito una serie di test per verificare che la funzionalità di vendita funzionasse correttamente.
Ho verificato che il giocatore potesse selezionare e vendere gli oggetti desiderati dall'inventario senza incontrare problemi di accesso o di riconoscimento degli oggetti. In aggiunta alla correzione menzionata in precedenza, ho introdotto un ulteriore miglioramento nel sistema di vendita degli oggetti. Ho aggiunto un parametro aggiuntivo che consente di distinguere gli oggetti che possono essere venduti da quelli che non possono essere venduti. Questo parametro è rappresentato da una variabile booleana che viene impostata su "true" nel caso in cui la vendita sia possibile e su "false" altrimenti. Quando il giocatore tenta di vendere un oggetto che non può essere venduto, il sistema rileva la condizione impostata sulla variabile booleana e visualizza un messaggio di errore appropriato. Questo messaggio di errore avverte il giocatore che l'oggetto selezionato non può essere venduto e suggerisce di provare con un altro oggetto nell'inventario.

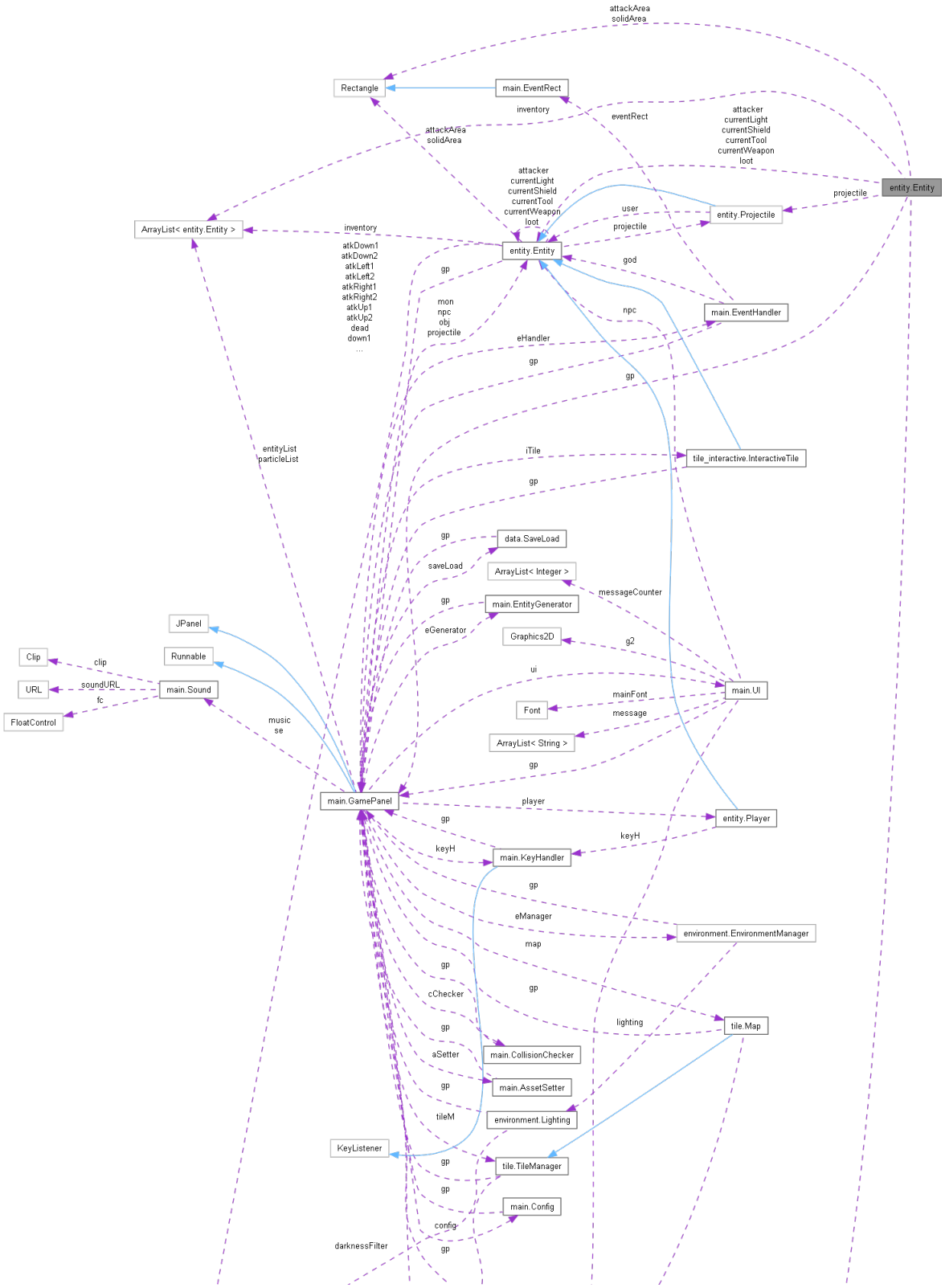
11.4.2 Aree solide troppo grandi

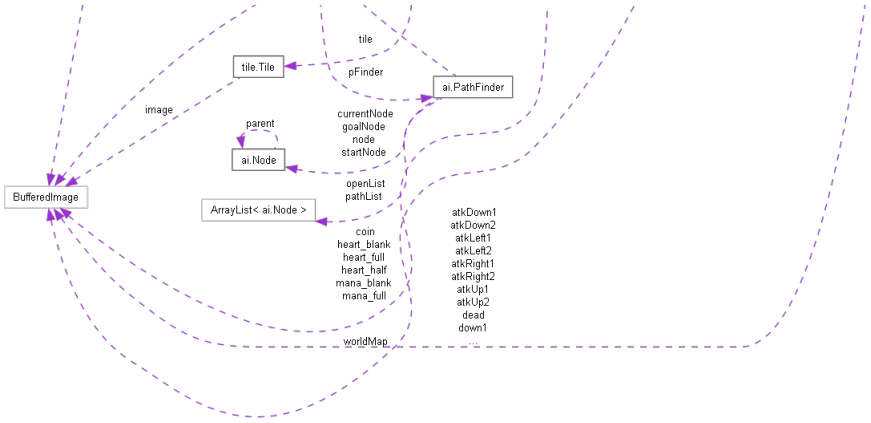
Durante l'implementazione del percorso dei personaggi basato sull'algoritmo A*, ho incontrato un problema relativo alle aree solide. In particolare, il personaggio doveva seguire un percorso prestabilito, ma si trovava nell'impossibilità di attraversare delle case 1x1 a causa dell'intersezione delle forme delle aree solide. Questo comportava un ostacolo insormontabile per il personaggio e limitava la sua capacità di navigare correttamente nell'ambiente di gioco. Per risolvere questa problematica, ho apportato una modifica alle aree solide delle entità. Ho ridotto l'area solida delle entità, consentendo al personaggio di attraversare gli spazi ristretti delle case 1x1. Questa modifica ha permesso al percorso calcolato dall'algoritmo A* di considerare tali spazi come attraversabili, garantendo che il personaggio potesse muoversi senza problemi anche in ambienti confinati. Dopo aver apportato questa modifica, ho eseguito una serie di test per verificare l'efficacia della soluzione. Ho constatato che il personaggio era in grado di superare gli ostacoli rappresentati senza intoppi, seguendo il percorso calcolato dall'algoritmo come previsto. Questa correzione ha quindi migliorato la navigazione dei personaggi nel gioco, rendendo l'esperienza di gioco più fluida e realistica.

11.4.3 Respawn delle entità di gioco

Durante lo sviluppo del mio gioco, ho affrontato un problema relativo al posizionamento del Boss sulla mappa di gioco. Inizialmente, ho cercato di spostare la gestione del posizionamento del Boss al di fuori dei metodi responsabili della generazione dei mostri normali, al fine di evitare il reset ciclico del Boss stesso. Tuttavia, ciò ha portato a un inconveniente inaspettato. Il problema era che alcuni mostri condividevano lo stesso indice dell'array del Boss, che rappresentava il numero di mappa. Quando i mostri venivano ripristinati regolarmente e gli array venivano resettati, l'indice che puntava alla posizione X e Y del Boss non veniva aggiornato. Di conseguenza, al primo respawn dei mostri, il Boss scompariva erroneamente dalla mappa di gioco. Per risolvere questa problematica, ho preso la decisione di spostare tutti i mostri presenti nella mappa di gioco del Boss al di fuori del metodo responsabile del respawn dei mostri. In questo modo, il posizionamento del Boss non era influenzato dalla logica di generazione dei mostri. Dopo aver apportato questa modifica, ho effettuato diversi test per assicurarmi che il Boss venisse correttamente posizionato sulla mappa e che non scomparisse durante i respawn dei mostri. Ho verificato che il Boss mantenesse la sua posizione stabilita e che potesse essere affrontato dal giocatore senza problemi.

12 Gerarchia delle classi




tile
tile.Tile
pFinder
ai.PathFinder
image
parent
currentNode
goalNode
node
startNode
ai.Node
openList
pathList
BufferedImage
ArrayList< ai.Node >
coin
heart_blank
heart_full
heart_half
mana_blank
mana_full
atkDown1
atkDown2
atkLeft1
atkLeft2
atkRight1
atkRight2
atkUp1
atkUp2
dead
down1
worldMap