**Primo Sprint**

In base alla definizione del primo Sprint Backlog, al membro del team in questione sono stati assegnati i seguenti task:

* Realizzazione Sprite: Personaggio, alcuni Nemici (Mob) d’esempio e la Prima Mappa;
* View: World;
* View: Combattimento;
* Controller: avvio di un combattimento.
* In base alle conoscenze personali del membro, si è deciso, che per la realizzazione degli Sprite, di dividere il lavoro su due supporti: uno web “piskelapp.com”, per la realizzazione dei personaggi, e un’applicazione “Paint.net” per la realizzazione della mappa.

Per Sprite s’intende quell’elemento grafico di gioco, di una determinata dimensione, che descrive visivamente: un personaggio, una zona della mappa….

Nel nostro si è deciso di creare Sprite di dimensione pixel, per avere una visibilità minima dei dettagli, dopodiché questi verranno ridimensionati in modo consono agli schermi attuali.

|  |  |  |
| --- | --- | --- |
| **pixel** | **Ridimensionamento** | **pixel** |
|  |  |  |

Mediante il supporto web si è innanzitutto il personaggio giocabile in varie posizioni, in modo tale da dare una bozza di movimento grafico nel gioco

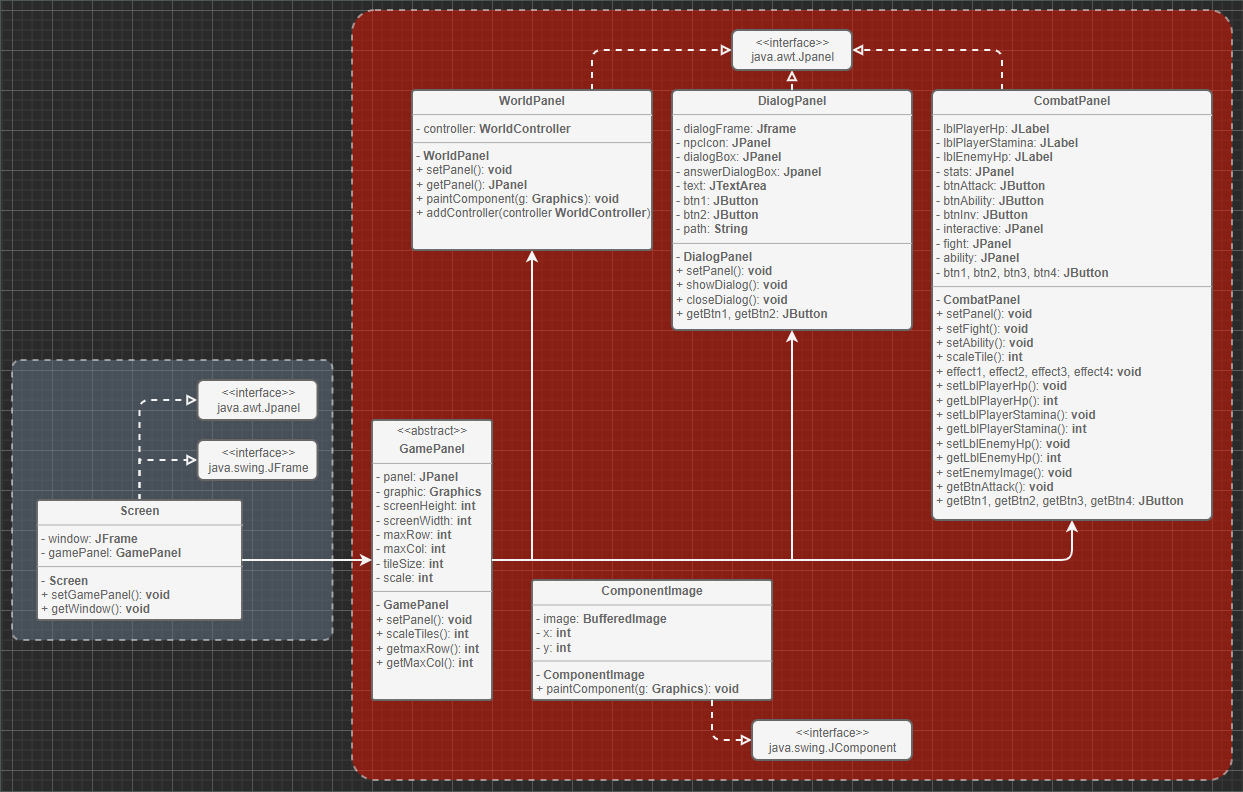
|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

Successivamente si è passato alla realizzazione di alcuni Mob applicando la stessa metodologia.

In seguito, mediante l’applicazione, grazie all’uso dei livelli grafici, si sono potuti realizzare gli Sprite della mappa, avendo sempre il riferimento dello Sprite precedente; si è fatto uso, inoltre, di una griglia che ha permesso di comporre visivamente la prima mappa:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

* Innanzitutto, si è deciso che per la realizzazione delle view si è deciso di far uso delle interfacce AWT e Swing.  
  La finestra del gioco deve essere una sola, pertanto, in vista delle duplici view, World e Combattimento, è stata stabilita un’astrazione in modo tale che le possa racchiudere tutte:



In primi termini generali si ha il seguente funzionamento:

* + La classe Screen include entrambe le librerie AWT e Swing, così facendo, fa da classe principe per la finestra di gioco. La classe introduce due metodi:   
    setGamePanel(), in modo tale da impostare il pannello corretto in base agli eventi del gioco,   
    getWindow(), che restituisce l’intera finestra che verrà fatta visualizzare all’avvia del gioco.
  + La classe GamePanel funziona come astrazione per i vari pannelli delle situazioni di gioco:  
    setPanel(), imposta i componenti grafici di gioco in base al tipo di pannello,  
    scaleTale(), imposta le dimensioni grafiche, in pixel, degli Sprite,   
    getMaxRow(), imposta il numero di righe nella mappa di gioco,   
    getMaxCol(), imposta il numero di colonne nella mappa di gioco.

Per la View: World lo schema si basa semplicemente su un semplice pannello, il quale richiama i metodi getMaxRow()e getMaxCol() impostando una matrice di gioco dove verranno caricati gli Sprite.

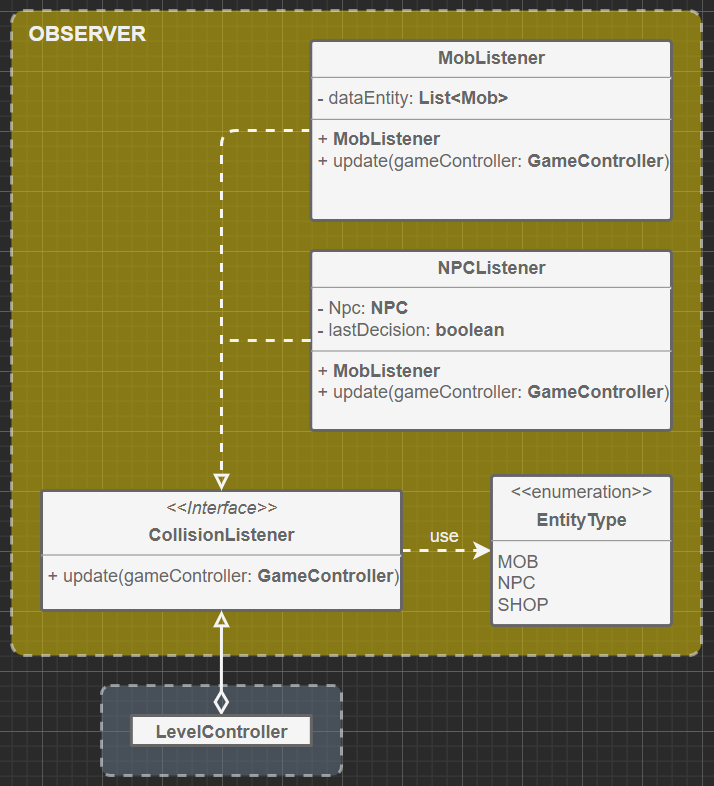
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

* Per la View: Combattimento si è voluto creare una struttura semplice ed intuibile, per questo si è pensato di seguire il seguente schema:

Immagine che contiene testo

Descrizione generata automaticamente

L’implementazione di tale schema, nella classe CombatPanel, è stata facilitata grazie ai metodi di Layout per i pannelli che la libreria AWT fornisce; per una maggiore visione della classe si descrivono in particolare i seguenti metodi:

* setFight(), imposta/scambia in “Pulsanti di combattimento” un pannello con i pulsanti sul come affrontare il nemico, i quali: *Attack*, *Ability*, *Inventory*;
* setAbility(), imposta/scambia in “Pulsanti di combattimento” un pannello specifico per le abilità con i relativi pulsanti, i quali: *Healing*, *Study Power*, *Cursed Attack*, *Sinner Attack*;
* scaleTile(), ridimensiona il tile del nemico che va impostato al centro;
* effect1, effect2, effect3, effect4(), rispettivamente imposta gli effetti delle relative abilità.
* Riguardo l’avvio del combattimento si è osservata la necessità di introdurre un Sistema di Collisione, la quale è stato analizzato e identificato nel pattern Observer (accanto si ha lo schema).

La classe CollisionListener si mette in ascolto della classe LevelController, il listener dopodiché, non appena avviene una collisione con lo Sprite di un Mob, avvia il metodo update(), la quale (effettuando prima di tutto un controllo con l’enumeration EntityType per capire il tipo di Sprite cui ci si scontra) avvia la (relativa) schermata di combattimento, con i dati del Mob a cui ci si è andato a collidere.

MobListener presenta:

* Una lista, dataEntity, di Mob, in quanto è possibile che in un combattimento si affrontino più nemici;
* update(), che trasferisce i dati del Mob al GameController e avvia il pannello di combattimento.

**Secondo Sprint**

A seguire, alla conclusione del primo Sprint, al membro, sono stati assegnati i seguenti task in base al secondo Sprint Backlog:

* Realizzazione Sprite: NPC, Icona dialogo NPC e la Seconda Mappa;
* View: Abilità;
* Business logic: NPC e dialoghi;
* Controller: Dialoghi;
* View: Dialoghi;
* Bug: più finestre si aprono durante lo switch tra combattimento ed esplorazione.
* Per la realizzazione degli Sprite: NPC, Icona dialogo NPC e della seconda mappa, si è sfruttata la stessa metodologia descritta nel primo task dello Sprint precedente; si specifica in particolare che l’Icona dialogo NPC è un’immagine più dettagliata dell’NPC con cui ci si va a parlare.

Immagine che contiene testo

Descrizione generata automaticamente

* Per la View: Abilità si è preso la struttura precedente del combattimento, impostato già in modo tale da anticipare l’onero dell’implementazione.

La classe CombatPanel non fa altro che richiamare il metodo setAbility() che, come descritto in precedenza, imposta/scambia in “Pulsanti di combattimento” un pannello specifico per le abilità con i relativi pulsanti, i quali: *Healing*, *Study Power*, *Cursed Attack*, *Sinner Attack*.

Immagine che contiene tavolo

Descrizione generata automaticamente

* Si evidenzi innanzitutto la seguente gerarchia   
  qui accanto.

Si è notato come la classe NPC possa detenere i dati dei Dialoghi; pertanto, quest’ultimi sono stati inseriti al suo interno, si può osservare che il funzionamento è molto simile ai dati contenuti nei Mob per il combattimento.

NPC inoltre implementa la classe NPCSprite, la quale questa detiene i dati riguardo gli Sprite e i relativi metodi get per la loro restituzione.

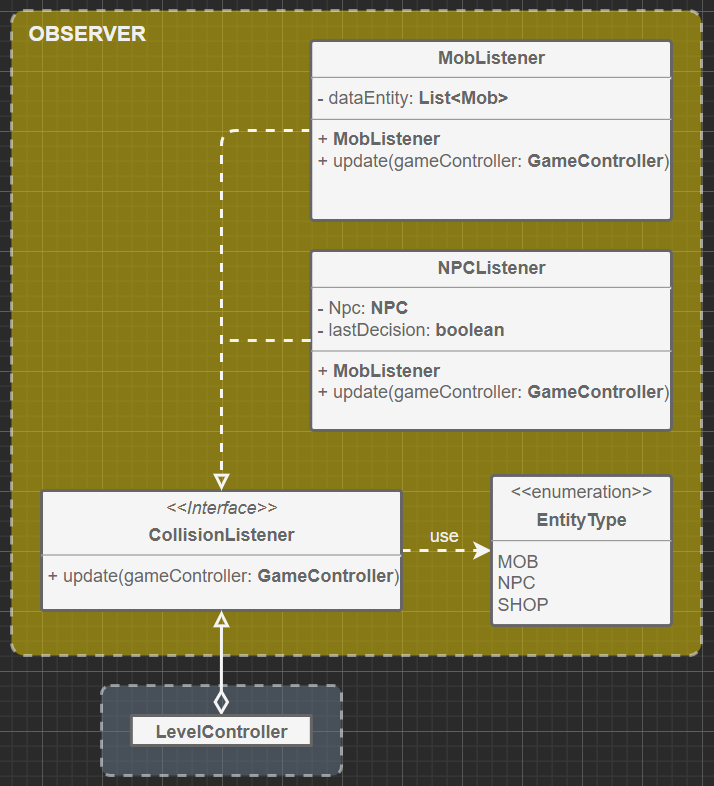
A seguire vengono mostrati i codice della Business Logic appena descritta.

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

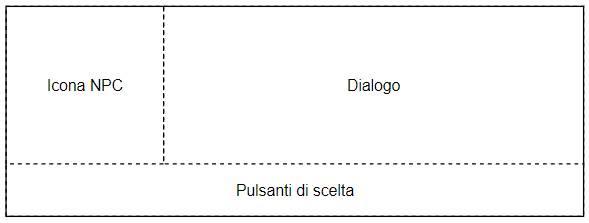
* Il Controller dei Dialoghi segue la stessa logica del combattimento, questa volta però l’enumeration identifica che il tipo di collisione e con un NPC, quindi notificherà la classe NPCListener e chiamerà il metodo update().

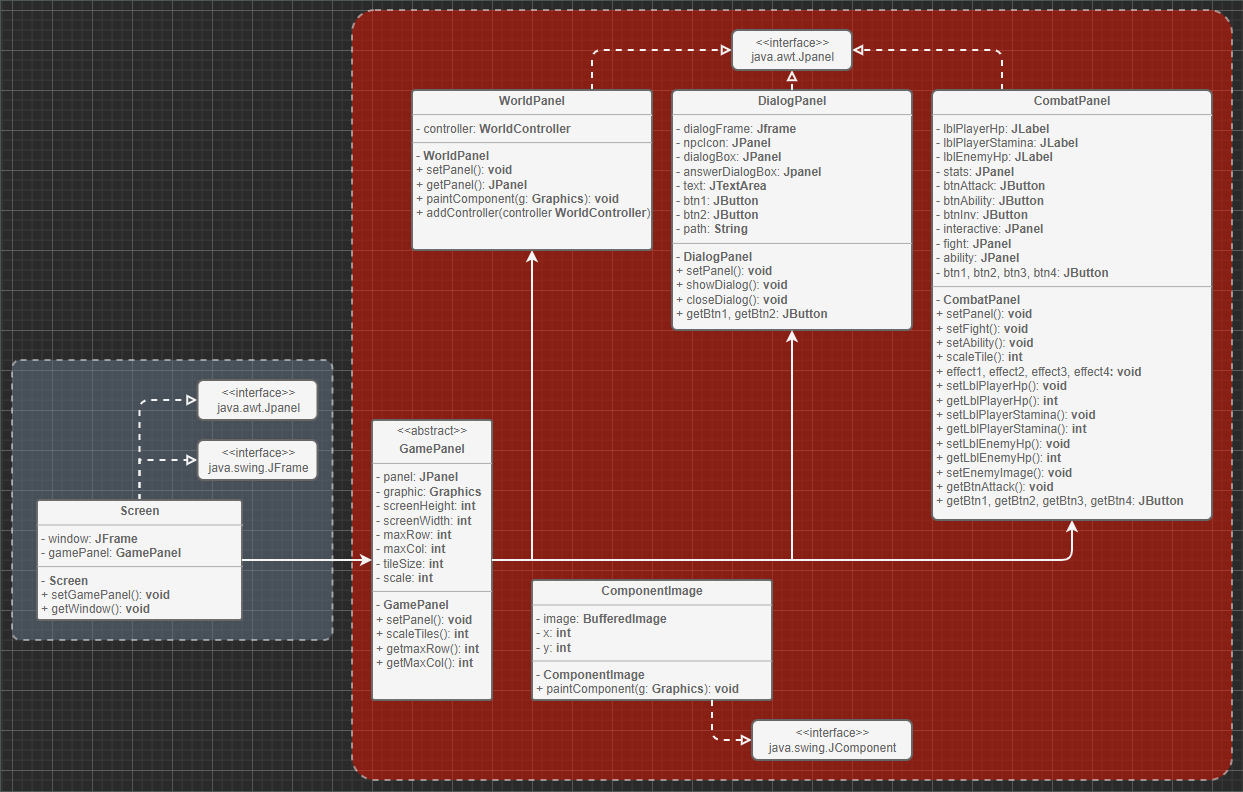
NPCListener presenta:

* Una classe, NPC, che detiene tutte le informazioni relative al dialogo;
* update(), che trasferisce i dati dell’NPC al GameController che avvierà una nuova finestra, mantenendo la finestra del mondo intatto.

Immagine che contiene testo

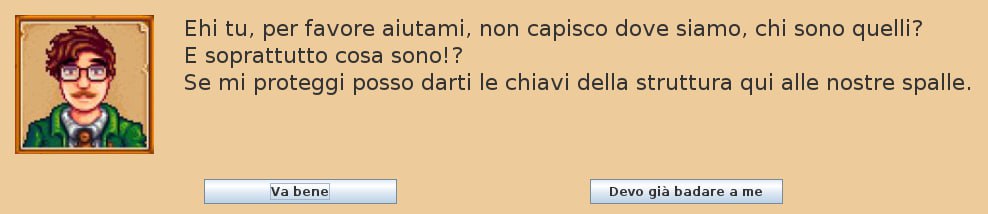
Descrizione generata automaticamente

* Per la View: Dialoghi si è tenuto è osservato che non è necessario l’intero cambiamento del pannello della finestra di gioco, si è pensato quindi l’apertura di un’ulteriore finestra temporanea dove avviene il dialogo stesso e poi chiusa (vedi figura per lo schema).

L’implementazione di tale schema avviene nella classe DialogPanel, si nota che in questa, per l’appunto, è stata introdotta una variabile JFrame dove verrà costruito tutto il dialogo al suo interno e mostrata quando richiamata; per una maggiore visione della classe si descrivono in particolare i seguenti metodi:

* showDialog(), mostra il dialogo costruito non appena avviene una collisione con l’NPC;
* closeDialog(), chiude il dialogo dopo aver dato una risposta;
* getBtn1, getBtn2(), rispettivamente i metodi per restituire il due pulsanti di scelta del dialogo.

Bisogna precisare che gli ultimi due metodi get servono principalmente per sapere il tipo di scelta che viene effettuata, in modo tale da proseguire nel sistema decisionale, il quale porterà vantaggi o svantaggi a secondo della scelta fatta.



* Si è osservato che dopo il completamento del primo Sprint che ogni volta che si passava dal pannello del mondo a quello di combattimento, e viceversa, si apriva una nuova finestra tenendo il stallo la finestra con la “situazione di gioco” precedente; risultando così quindi un Bug.  
  Analizzando il codice ci si è accorto che nel loop della classe Game, ad ogni switch di pannello, creava una nuova finestra; identificato il problema si è corretto riottenendo un uso corretto dei pannelli e dello stato di gioco.