Immagine che contiene testo

Descrizione generata automaticamente

|  |  |
| --- | --- |
| **Capitolo 1: Analisi dei requisiti**  - Specifiche dei requisiti  - Requisiti non funzionali  - Requisiti funzionali | pag.  pag.  pag. |
| **Capitolo 2: Progettazione architetturale**  -  -  - | pag.  pag.  pag. |
| **Capitolo 3: Progettazione del software**  -  -  - | pag.  pag.  pag. |

// Nel Cap. 1 si parla dei requisiti  
// Nel Cap. 2 si parla della progettazione MVC e Scrum Method  
// Nel Cap. 3 si parla dei nostri task

Partendo dalle specifiche preliminari, il primo step dell’analisi dei requisiti è stato concepire la storia di base del gioco.

CAPITOLO 1:  
ANALISI DEI REQUISITI

**Storia del gioco:**

L’idea del gioco è che il protagonista affronti un certo numero di livelli e sfide, a ognuno legato un particolare tema, inoltre al relativo tema sarà legato anche un nuovo personaggio che si aggiungerà al gruppo ad ogni livello.

Durante il viaggio verranno svelati i motivi dietro al male che sta corrompendo il mondo (7 peccati capitali) e si scoprirà la mente dietro a tutto questo.

In base alle scelte compiute durante il gioco il protagonista può scegliere se arrivare a un finale buono o a un finale cattivo. Nel finale buono, il protagonista, insieme agli alleati che raggrupperà, salverà il mondo dalla corruzione, Nel finale cattivo, invece, il protagonista, insieme ai suoi compagni, costituiranno il nuovo male del mondo.

**Specifiche dei requisiti**

Le specifiche dei requisiti sono suddivise in due categorie: meccaniche del gioco e sistema di livelli.

**Meccaniche del gioco:**

1. Il giocatore deve poter personalizzare il proprio personaggio;
2. Il giocatore deve poter scegliere almeno tra tre tipologie di classi, che determineranno:
3. L’equipaggiamento;
4. Sistema di statistiche, che influiscono sul combattimento;
5. Un albero delle abilità, che permette al giocatore di sbloccare abilità uniche utilizzabili in combattimento;
6. Il gioco deve presentare un tutorial che introduce alle meccaniche del gioco;
7. Il giocatore deve potere mettere in pausa il gioco;
8. Il gioco deve poter permettere di salvare la partita e riprenderla in un momento successivo;
9. Il gioco deve presentare un sistema di combattimento a turni;
10. Il giocatore alla fine di ogni combattimento deve ricevere:
11. Un certo numero di punti esperienza, che permetteranno di espandere l’albero delle abilità;
12. Un certo numero di valuta di gioco e materiali, che gli permettono di aumentare/potenziare le statistiche ed equipaggiamento.
13. Il gioco presenta diverse situazioni di stallo (almeno una per livello), in cui il giocatore è tenuto a scegliere tra diverse opzioni che influiscono sul proseguimento del gioco;
14. Il gioco deve presentare almeno due finali.
15. Il gioco deve tenere conto delle scelte fatte durante il gioco che stabiliscono il finale del gioco;

**Sistema di livelli:**

1. Il gioco deve essere suddiviso in livelli, che a loro volta devono essere suddivisi in aree;
2. Il superamento dei livelli si basa sulla sconfitta di un boss di fine livello;
3. Ciascuna area deve presentare un certo numero di nemici che devono essere sconfitti per raggiungere il boss del livello;
4. Il giocatore può trovare degli oggetti durante l’esplorazione dei livelli, che possono facilitare il proseguimento del gioco;
5. Il gioco propone delle sfide al giocatore che offrono **bonus** in caso di vittoria o **malus** in caso di sconfitta.

**Requisiti non funzionali**

Dalle specifiche dei requisiti, sono stati dedotti i requisiti non funzionali, elencati di seguito.

1. L’introduzione alle meccaniche del gioco deve essere graduale:
2. Come criterio di verificabilità, il giocatore non deve essere introdotto a tutte le meccaniche del gioco nella prima fase, ma deve risultare suddiviso in più parti, sparse nelle fasi iniziali del gioco.
3. Il salvataggio deve:
4. Essere rapido, in particolare non deve essere superare (indicativamente) i 30 secondi;
5. Poter essere effettuato in qualunque momento al di fuori dei combattimenti;
6. Essere notificato al giocatore una volta correttamente effettuato;
7. L’interfaccia utente deve essere semplice e intuitiva, in particolare devono essere presenti due menù:
8. Menù di pausa per permettere:

* Di riprendere la partita;
* Di uscire dal gioco;
* Di salvare la partita;

1. Menù di gioco per permettere:

* Di visualizzare le proprie statistiche;
* Di visualizzare e usare gli oggetti raccolti durante i livelli;
* Di visualizzare e modificare il proprio albero delle abilità;

1. La schermata di combattimento non deve presentare troppi pulsanti;
2. Requisiti organizzativi:
3. Il processo di sviluppo dovrà seguire la Metodologia Agile, con particolare riferimento a Scrum;
4. Il linguaggio di programmazione del software sarà Java;
5. I software che usati per lo sviluppo includono:

* L’IDE IntelliJ IDEA;
* Il CVS Git e la piattaforma Github;
* Il DBMS MySQL;
* Strumenti per il disegno del sistema software in UML;

**Requisiti di fidatezza**

**salvataggio**

**Requisiti funzionali**

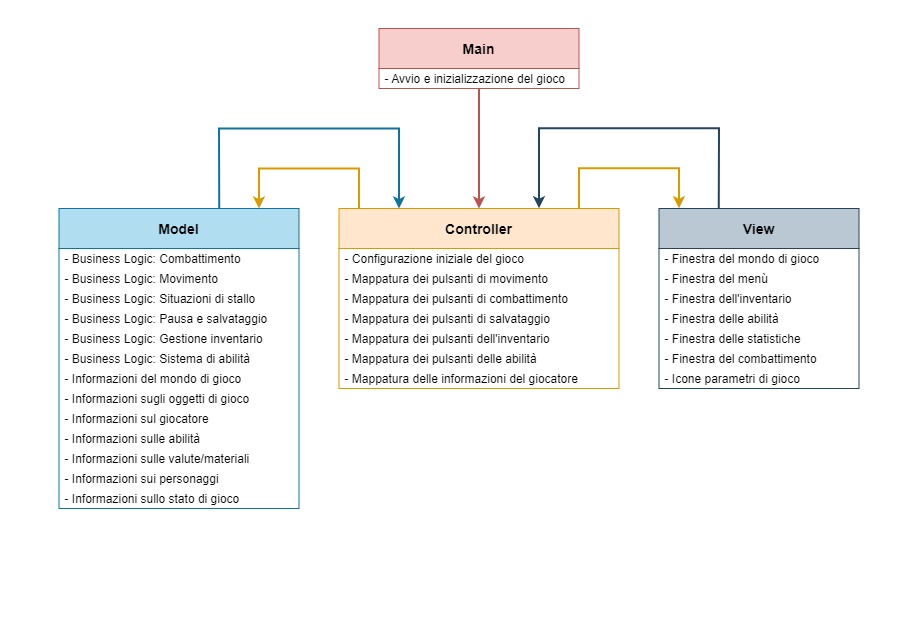
I requisiti funzionali individuati sono stati definiti nella forma di user story, usando il modello “Who, what, why” e la specifica sintassi “As a <type of user> i want <some goal> so that <some reason>, includendo per ognuna di esse uno o più criteri di accettazione.

Le user story prodotte hanno contribuito alla formazione del product backlog. A ogni user story è stata assegnata una priorità, un tipo, un criterio di accettazione, un tema, story points e uno status. Per gli story points è stato deciso di utilizzare come unità di misura le taglie delle t-shirt, scelta che si è rivelata vantaggiosa al momento del poker planning. Il product backlog viene elencato di seguito.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Priority** | **Type** | **Item** | **Criteria** | **Theme** | **SP** | **Status** |
| 1 | **1** | Story | In qualità di giocatore,  voglio potermi muovere nei livelli in modo tale da poterli esplorare | Il giocatore deve poter: - Premere uno tra i pulsanti   W,A,S,D - Visualizzare il personaggio   muoversi verso le rispettive   direzioni SU,GIU', SX e DX | Meccaniche | **L** | Completed |
| 2 | **1** | Story | In qualità di giocatore, voglio poter avviare un combattimento con un nemico in modo tale da avanzare nella storia | Il giocatore deve poter: - Visualizzare i nemici nella mappa   di gioco - Decidere se affrontarli - Avviare un combattimento | Meccaniche | **M** | Completed |
| 3 | **1** | Story | In qualità di giocatore, voglio poter attaccare con l'arma in modo tale da infliggere al nemico danni in combattimento | Il giocatore deve poter: - Premere un tasto durante il   combattimento - Infliggere un certo danno in base   alle proprie statistiche | Meccaniche | **M** | Completed |
| 4 | **2** | Story | In qualità di giocatore, voglio poter utilizzare le abilità  in modo tale da infliggere danni al nemico in combattimento | Il giocatore deve poter: - Premere un bottone - Scegliere l'abilità - Attaccare il nemico | Meccaniche | **M** | Completed |
| 5 | **2** | Story | In qualità di giocatore, voglio poter prendere decisioni in modo tale poter influenzarne il proseguimento | Il giocatore deve poter: - Visualizzare almeno due pulsanti - Sceglierne solo uno  - Influenzare   (aumentare/diminuire) le  caratteristiche del gioco | Meccaniche | **L** | Completed |
| 6 | 2 | Epic | In qualità di sistema, voglio poter suddividere il gioco in livelli in modo tale da strutturare la storia | Ogni livello deve corrispondere a una specifica parte della storia,  contenendo oggetti, personaggi, nemici e sfide uniche del livello. Il completamento di un livello fornisce l'accesso al successivo. | Sistema | XL | In progress |
| 7 | **2** | Story | In qualità di sistema, voglio poter fornire un primo livello in modo tale da introdurre la storia | Il primo livello deve corrispondere alla parte iniziale della storia di gioco, contenere due mappe di gioco ognuna con almeno una decisione da prendere e vari nemici da affrontare. | Sistema | **L** | Completed |
| 8 | **2** | Story | In qualità di sistema, voglio poter fornire un secondo livello in modo tale da sviluppare la storia | Il secondo livello deve corrispondere alla parte centrale della storia di gioco, contenere tre mappe di gioco ognuna con almeno una decisione da prendere, nemici da affrontare e sfide da superare | Sistema | **L** |  |
| 9 | **2** | Story | In qualità di sistema, voglio poter fornire un terzo livello in modo tale da concludere la storia | Il terzo livello deve corrispondere alla parte finale della storia di gioco, contenere tre mappe di gioco ognuna con almeno una decisione da prendere, nemici da affrontare, sfide da superare boss finale da sconfiggere | Sistema | **L** |  |
| 10 | **2** | Story | In qualità di giocatore,  voglio poter visualizzare la mia salute e stamina, durante l'esplorazione,  in modo tale da decidere come proseguire il combattimento | Il giocatore deve poter: - Visualizzare nella schermata delle   barre che indicano la salute e la   stamina del personaggio | Meccaniche | **S** |  |
| 11 | **2** | Story | In qualità di giocatore,  voglio poter visualizzare il mio attuale obiettivo  in modo tale da sapere cosa devo fare | Il giocatore deve poter: - Visualizzare nella schermata una   finestra con scritto l’obiettivo   attuale | Meccaniche | **M** |  |
| 12 | **2** | Bug | Più finestre si aprono durante lo switch tra combattimento e esplorazione | Deve essere presente un solo JFrame comune per esplorazione e combattimento, per tutto il gioco | Meccaniche | **M** | Completed |
| 13 | **3** | Story | In qualità di giocatore, voglio poter visualizzare il mio inventario  in modo tale da utilizzare i miei oggetti, anche in combattimento | Il giocatore deve poter: - Premere un tasto (I) - Scorrere la schermata degli   oggetti - Selezionare l’oggetto da scegliere   se utilizzare |  |  |  |
| 14 | **3** | Story | In qualità di giocatore, voglio poter visualizzare una descrizione degli oggetti  in modo tale da sapere il loro effetto, anche in combattimento | Il giocatore deve poter: - Premere un tasto (I) - Scorrere la schermata degli   oggetti - Scegliere un oggetto da vederne   l’effetto |  |  |  |
| 15 | **3** | Story | In qualità di giocatore,  voglio poter mettere in pausa il gioco  in modo tale da riprenderlo più avanti | Il giocatore deve poter: - Premere un tasto (Esc) - Visualizzare un menu - Aspettare il tempo necessario - Premere lo stesso tasto per   tornare in partita |  |  |  |
| 16 | **3** | Story | In qualità di giocatore,  voglio poter mettere in pausa il gioco  In modo tale da salvare i miei progressi | Il giocatore deve poter: - Premere un tasto (Esc) - Visualizzare un menu - Premere un bottone per salvare - Visualizzare se il salvataggio ha   successo |  |  |  |
| 17 | **3** | Story | In qualità di giocatore,  voglio poter mettere in pausa il gioco  in modo tale da uscire da gioco | Il giocatore deve poter: - Premere un tasto (Esc) - Visualizzare un menu - Premere un bottone per uscire |  |  |  |
| 18 | **3** | Story | In qualità di giocatore, voglio poter avviare un combattimento con un nemico in modo tale da guadagnare dei materiali | Il giocatore deve poter: - Visualizzare i nemici - Decidere se affrontarli - Combattere i nemici - Vinto il combattimento ricevere   dei materiali |  |  |  |
| 19 | **3** | Epic | In qualità di giocatore, voglio poter affrontare delle sfide durante i livelli in modo tale da ottenere dei vantaggi | Durante i livelli, il giocatore deve poter interagire con un personaggio che gli propone la sfida e decidere se iniziarla o meno |  |  |  |
| 20 | **3** | Story | In qualità di giocatore,  voglio poter visualizzare i miei punti esperienza  in modo tale da sapere se posso potenziare l'albero delle abilità | Il giocatore deve poter: - Premere un tasto (I) - Cambiare finestra - Scorrere la schermata delle abilità |  |  |  |
| 21 | **3** | Story | In qualità di giocatore,  voglio poter visualizzare l'albero delle abilità  in modo tale da avere un resoconto delle mie abilità,  anche in combattimento | Il giocatore deve poter: - Premere un tasto (I) - Cambiare finestra - Scorrere la schermata delle abilità |  |  |  |
| 22 | **3** | Story | In qualità di giocatore,  voglio poter visualizzare l'albero delle abilità  in modo tale da poter espandere le mie abilità | Il giocatore deve poter: - Premere un tasto (I) - Cambiare finestra - Scorrere la schermata delle abilità   scegliere  - Selezionare l’abilità da scegliere se   espandere |  |  |  |
| 23 | **4** | Story | In qualità di giocatore,  voglio poter visualizzare i miei materiali e valuta  in modo tale da sapere se posso potenziare le mie statistiche | Il giocatore deve poter: - Premere un tasto (I) - Cambiare finestra - Scorrere la schermata delle   statistiche |  |  |  |
| 24 | **4** | Story | In qualità di giocatore,  voglio poter visualizzare le mie statistiche in modo tale da avere un resoconto del mio personaggio,  anche in combattimento | Il giocatore deve poter: - Premere un tasto (I) - Cambiare finestra - Scorrere la schermata delle   statistiche |  |  |  |
| 25 | **4** | Story | In qualità di giocatore,  voglio poter visualizzare le mie statistiche in modo tale da potenziarle | Il giocatore deve poter: - Premere un tasto (I) - Cambiare finestra - Scorrere la schermata delle   statistiche  - Scegliere l’equipaggiamento da   potenziare |  |  |  |
| 26 | **5** | Story | In qualità di giocatore,  voglio poter visualizzare una guida di gioco in modo tale da sapere come si gioca | Il giocatore deve poter: - Premere un tasto (Esc) - Visualizzare un menu - Premere un bottone per cambiare   finestra - Visualizzare la guida del gioco |  |  |  |
| 27 | **5** | Story | In qualità di giocatore,  voglio poter controllare una mappa  in modo tale da sapere in quale punto mi trovo e decidere dove andare | Il giocatore deve poter: - Premere un tasto (M) - Visualizzare una mappa |  |  |  |
| 28 | **6** | Epic | In qualità di giocatore,  voglio poter personalizzare il mio personaggio  in modo tale da provare diverse esperienze di gioco | All'inizio del gioco, il giocatore deve poter: - Scegliere la classe del personaggio - Scegliere nome e sesso |  |  |  |

Dopo la stesura del product backlog, il team di sviluppo si è concentrato sulla definizione dell’architettura da utilizzare per il progetto e l’implementazione del gioco. L’architettura che è stata ritenuta più consona per lo sviluppo è stata l’architettura MVC (Model View Controller).

CAPITOLO 2:  
PROGETTAZIONE ARCHITETTURALE



Per il disegno dell'architettura si è scelta una raffigurazione più testuale che pittorica.

L'architettura scelta si compone di 4 strati: il model, il controller, e il main.

La parte di “Model” fa riferimento a tutta la business logic necessaria per il funzionamento del gioco, oltre che a una parte di database per conservare tutte le informazioni necessarie per la gestione dei livelli e del giocatore. Si è deciso di utilizzare un database relazionale e il DBMS MySQL.   
La parte di "view" si riferisce a tutte le finestre che compongono ciò con cui l'utente interagisce: tra queste figurano, principalmente, la schermata di esplorazione e la schermata di combattimento. La parte di view è stata interamente realizzata in Java Swing.   
La parte denominata come "Controller" si riferisce a tutti i controller di gioco che permettono lo scambio di informazioni tra View e Controller. Tra questi, due controller costituiscono le fondamenta delle meccaniche del gioco: il controller di movimento del personaggio e il controller di mappatura dei pulsanti di movimento.   
La parte di “Main” rappresenta il punto di ingresso del software. Questa parte si occupa dell’inizializzazione del gioco, intesa come costruzione degli oggetti controller, model e view, oltre che all’avvio dello stesso. Il criterio di funzionamento cardine dell’intero gioco è basato sull’idea che vi sono due possibili macro stati: lo stato di “esplorazione” e lo stato di “combattimento”. Lo stato di esplorazione include il movimento del personaggio per i vari livelli, l’esplorazione degli stessi e l’interazione con i personaggi e oggetti di gioco. Lo stato di “combattimento” include invece il sistema di combattimento a turni specificato nei requisiti funzionali. Quando un giocatore interagisce con un nemico, passa allo stato di “combattimento”. Una volta finito il combattimento si torna allo stato di “esplorazione”. Il processo si ripete per tutto il gioco.

**Definizione degli sprint**

Seguendo passo per passo la metodologia agile, con particolare riferimento al framework Scrum, dopo la stesura del product backlog e la definizione dell’architettura il team si è riunito per definire la durata degli sprint e redigere il primo sprint backlog. Per la durata degli sprint si è scelto di usare come unità di misura le ore e si è stabilito che ogni sprint abbia una durata di 20 ore, da suddividere in 10 giorni lavorativi.

Ogni user story è stata suddivisa in task per facilitare il processo di sviluppo e ognuno di essi è stato assegnato a un particolare membro del team, con riferimento alle competenze tecniche di ognuno. Le user story si definivano completate (done) quando i criteri di accettazione risultavano verificati.  
Approssimativamente, le ore di sviluppo calcolate risultano pari a 100. Alla fine del progetto, risultano effettuati due sprint, i cui sprint goal sono stati entrambi pienamente raggiunti. Di seguito, si elencano i due sprint backlog di riferimento.

**Primo sprint backlog**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Goal:** | ***Completare le meccaniche di movimento e combattimento basilari*** |  |  | **IEH:** | ***100*** |  |  |  |  |  |
| **ID** | **Item/Acceptance Critera** | **Tasks** | **SP** | **Owner** | **Planned hours** | **Giov** | **Ven** | **Sab** | **Mer** | **Giov** |
| **1** | In qualità di giocatore,  voglio potermi muovere neI livelli in modo tale da poterli esplorare |  | **L** |  |  | Remaining hours | | | | |
|  | - Premere uno tra i pulsanti   W,A,S,D - Visualizzare il personaggio   muoversi verso le rispettive   direzioni SU,GIU', SX e DX | Creare view mappa di gioco |  | Andrea |  |  |  |  |  |  |
| Controller movimento personaggio |  | Raffaele |  |  |  |  |  |  |
| Business logic: movimento personaggio |  | Raffaele |  |  |  |  |  |  |
| Disegno delle entità di gioco su schermo |  | Raffaele |  |  |  |  |  |  |
| **2** | In qualità di giocatore,  voglio poter avviare un combattimento con un nemico in modo tale da avanzare nella storia |  | **M** |  |  | Remaining hours | | | | |
|  | - Visualizzare i nemici nella   mappa di gioco - Decidere se affrontarli - Avviare un combattimento | Creare view combattimento |  | Andrea |  |  |  |  |  |  |
| Business logic: avvio di un combattimento |  | Andrea |  |  |  |  |  |  |
| Creare controller combattimento |  | Alessia |  |  |  |  |  |  |
| **3** | In qualità di giocatore,  voglio poter attaccare con l'arma in modo tale da infliggere al nemico danni in combattimento |  | **M** |  |  | Remaining hours | | | | |
|  | - Premere un tasto durante   il combattimento - Infliggere un certo danno   in base alle proprie   statistiche | Business logic: sistema di statistiche |  | Alessia |  |  |  |  |  |  |
| Business logic: combattimento a turni |  | Giovanni |  |  |  |  |  |  |
| Progetto database entità di gioco |  | Giovanni |  |  |  |  |  |  |

**Secondo sprint backlog**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Goal:** | ***Completare il primo livello*** |  |  | **IEH:** | ***100*** |  |  |  |  |  |
| **ID** | **Item/Acceptance Critera** | **Tasks** | **SP** | **Owner** | **Planned hours** | **Giov** | **Ven** | **Sab** | **Mer** | **Giov** |
| **6** | "In qualità di giocatore, voglio poter utilizzare le abilità in modo tale da avere vantaggi in combattimento" |  | **M** |  |  | Remaining hours | | | | |
|  | Il giocatore deve poter:  - Premere un bottone  - Scegliere l'abilità  - Attaccare il nemico | Business logic: skill tree e combattimento |  | Giovanni |  |  |  |  |  |  |
| Controller combattimento: abilità |  | Alessia |  |  |  |  |  |  |
| View abilitá |  | Andrea |  |  |  |  |  |  |
| **7** | "In qualità di giocatore, voglio poter prendere decisioni in modo tale poter influenzare il proseguimento del gioco" |  | **L** |  |  | Remaining hours | | | | |
|  | Il giocatore deve poter: - Visualizzare almeno due pulsanti - Sceglierne solo uno  - Influenzare   (aumentare/diminuire)le  caratteristiche del gioco | Business logic: NPC e dialoghi |  | Andrea |  |  |  |  |  |  |
| Disegno NPC su schermo |  | Raffaele |  |  |  |  |  |  |
| Controller: dialoghi |  | Andrea |  |  |  |  |  |  |
| Business logic: sistema decisionale |  | Raffaele |  |  |  |  |  |  |
| Controller: sistema decisionale |  | Raffaele |  |  |  |  |  |  |
| Realizzazione sprite NPC |  | Andrea |  |  |  |  |  |  |
| View: dialoghi |  | Andrea |  |  |  |  |  |  |
| **8** | In qualità di sistema, voglio poter fornire un primo livello di gioco in modo tale da introdurre la storia |  | **L** |  |  | Remaining hours | | | | |
|  | Il primo livello deve corrispondere alla parte iniziale della storia di gioco, contenere due mappe di gioco con varie situazioni di stallo | Connessione al DB |  | Giovanni |  |  |  |  |  |  |
| Business logic: livelli e mappe |  | Alessia |  |  |  |  |  |  |
| Realizzazione sprite seconda mappa |  | Andrea |  |  |  |  |  |  |
| Controller livelli e mappe |  | Raffaele |  |  |  |  |  |  |
| **25** | Bug: più finestre si aprono durante lo switch tra combattimento ed esplorazione |  | **M** |  |  |  | | | | |
|  | Deve essere presente un solo JFrame comune per esplorazione  e combattimento, per tutto il gioco |  |  | Andrea |  |  |  |  |  |  |

Per una maggiore chiarezza, la parte di progettazione è suddivisa in tre macroaree, corrispondenti alle meccaniche principali del gioco: esplorazione, combattimento e inizializzazione.

CAPITOLO 3:  
PROGETTAZIONE DEL SOFTWARE

**Esplorazione**

**Entità di gioco: Player e Mob**

Sviluppatore: Raffaele Terracino

Le entità di gioco individuate nel primo sprint sono state il Player, rappresentante del personaggio controllato dall’utente, e Mob, rappresentante dei nemici con cui il Player può combattere. Player e Mob incapsulano un oggetto di tipo Statistics, che incapsula le statistiche del combattimento, e rispettivamente un oggetto di tipo PlayerSprite e MobSprite, che incapsulano le informazioni sulle immagini da stampare a schermo. La classe Player contiene anche il metodo move(), che si occupa, in base all’EventType passato, di aggiornare i due interi worldX e worldY dell’oggetto playerSprite, sommando o sottraendo ad essi l’offset movementSpeed, pari a 10. Il metodo move è stato sviluppato seguendo il TDD. Seguono i test progettati per tale metodo.

class PlayerTest {  
 private Player p;  
  
 @BeforeEach  
 void setUp() {  
 p = new Player();  
 }  
  
 @AfterEach  
 void tearDown() {  
  
 }  
  
 @Test  
 void testMove() {  
 p.move(EventType.*MOVED\_DOWN*);  
 *assertEquals*(130, p.getPlayerSprite().getWorldY());  
 p.move(EventType.*MOVED\_UP*);  
 *assertEquals*(120, p.getPlayerSprite().getWorldY());  
 p.move(EventType.*MOVED\_RIGHT*);  
 *assertEquals*(110, p.getPlayerSprite().getWorldX());  
 p.move(EventType.*MOVED\_LEFT*);  
 *assertEquals*(100, p.getPlayerSprite().getWorldX());  
 }  
}

Il costruttore di default imposta il Player alla posizione di default, utilizzata nei test.

Le classi MobSprite e Mob non contengono altri metodi oltre i getter e setter, per cui non è stato necessario effettuare i test di unità.

**Entità di gioco: NPC**

Immagine che contiene tavolo

Descrizione generata automaticamenteSviluppatore: Andrea Spinelli

* Introduzione a cos’è un npc --

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.

**Sistema di mappe e livelli**

Sviluppatore: Alessia Bonì

**Immagine che contiene testo

Descrizione generata automaticamente**

**Classe Map**

La classe Map contiene quattro matrici:

* mapEnemies: contiene le posizioni e i riferimenti dei nemici
* mapNpc : contiene le posizioni e i riferimenti agli npc
* worldTiles: contiene tutti i tile del mondo relative all’esplorazione
* endMap: matrice booleana che contiene la fine della mappa: ovvero i tile che indicano al controller di cambiare mappa

Metodi get restituiscono sia le singole matrici ma, permettono anche di ottenere il singolo nemico ,tile o Npc.

**Classe MapBuilder**

Classe Map, si avvantaggia di un builder per la costruzione dei suoi oggetti, in quanto essendo quattro matrici, il costruttore prenderebbe a parametro 8 coordinate, causando il problema dei costruttori telescopici, inoltre il pattern builder apporta un vantaggio fondamentale: costruzione Step-by-Step degli oggetti, che permette un codice più leggibile e una costruzione semplificata e organica dei singoli oggetti

Il sistema è così composto:

* Un’interfaccia Builder che propone i metodi principali per costruire i vari oggetti
* Un classe MapBuilder che implementa builder, che supporta Map
* La classe Map che contiene la logica fondamentale delle mappe

In *MapBuilder* sono presenti anche dei metodi set per facilitare il riuso di tutte le matrici , e i metodi build… relativo alle matrici da costruire che prende a parametro le coordinate della grandezza della specifica mappa, con infine il metodo build() che restituisce un oggetto completo di tipo Map

Oltre ai vari metodi build…() si sono forniti anche dei metodi specifici per costruire i singoli Nemici, Npc , worldTile e endTile per semplificare la costruzione della mappa, rispettivamente addMob,addNpc,addWorldTile,addEndTile che prendono a parametro le cordinate per aggiungere l’elemento.

Il metodo reset( ) setta tutti i parametri a null, in caso d’errore di inserimento

Immagine che contiene testo, tavolo

Descrizione generata automaticamente***Classe Level***

La classe Level si occupa dell’organizzazione di un livello, contiene un ArrayList di mappe: maps che contiene due mappe per livello, e un counter che permette di scegliere le mappe

Il metodo ***switchMap***: si occupa di cambiare la mappa, ovvero passare da Mappa 1 a Mappa 2 rispettivamente maps(0) e maps(1) incrementando o decrementando il counter.

Il costruttore si occupa di aggiungere le mappe passate a parametro , i metodi get restituiscono il counter e la mappa corrente.

**View dell’esplorazione**

Sviluppatore: Andrea Spinelli

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

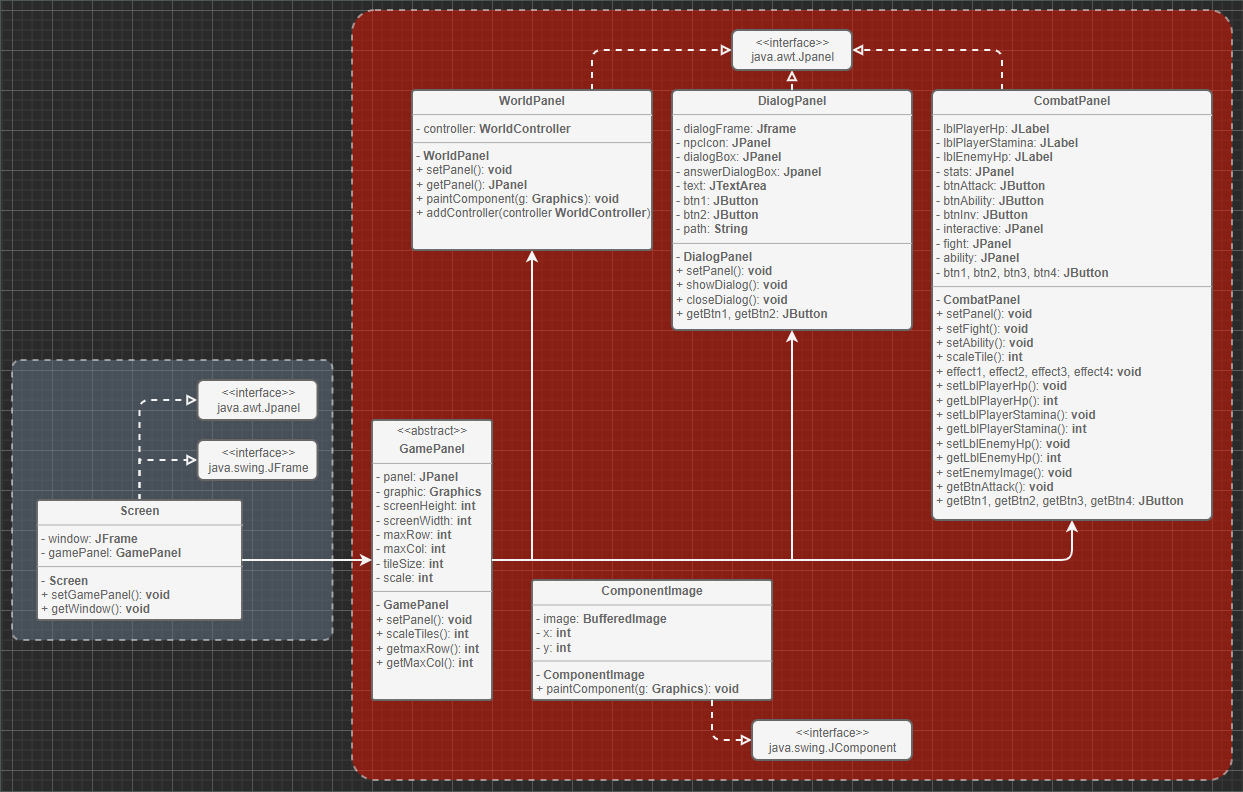
|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

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:

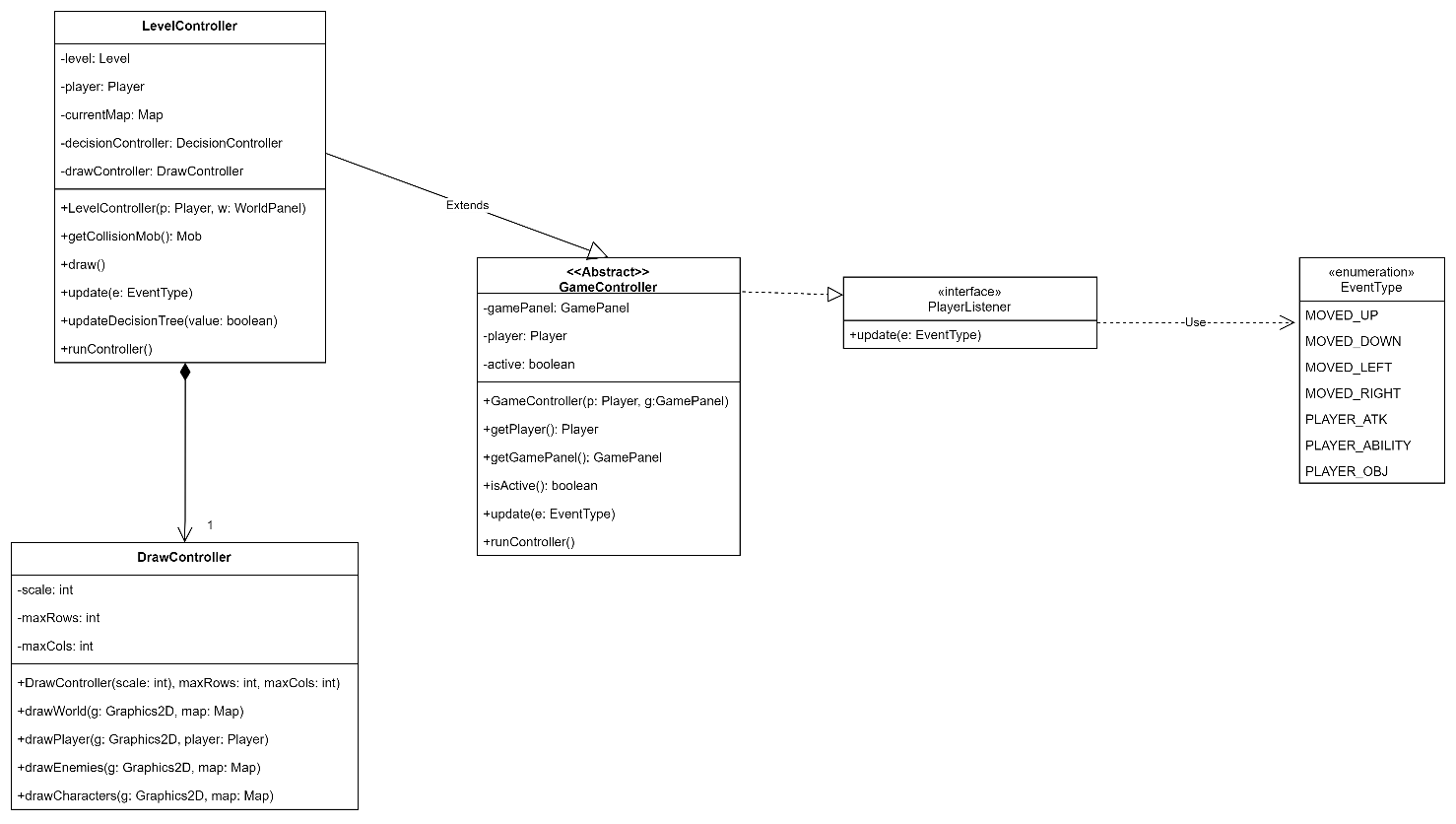
La classe Screen include entrambe le librerie AWT e Swing, così facendo, fa da classe principe per la finestra di gioco. La classe introduce due metodi:   
setGamePanel(), in modo tale da impostare il pannello corretto in base agli eventi del gioco,   
getWindow(), che restituisce l’intera finestra che 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.

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

**Disegno delle entità di gioco su schermo**

****

La classe LevelController è una sottoclasse di GameController, che rappresenta un generico controller di meccaniche di gioco. La classe GameController incapsula un riferimento a un GamePanel, classe astratta rappresentante di una generica interfaccia di gioco, un riferimento a un Player e un booleano active, che stabilisce lo stato del controller. Oltre il costruttore e il metodo runController, che si occupa di svolgere le funzioni cardine del controller, la classe implementa il metodo update() dell’interfaccia PlayerListener.

**Connessione con level e map**

Il disegno delle entità di gioco è gestito dalle classi dalle classi LevelController e DrawController. Il LevelController, all’interno del metodo runController, 60 volte per secondo, chiama il metodo repaint della classe WorldPanel. Il metodo repaint(), proprio della classe JPanel di cui WorldPanel è una sottoclasse, a sua volta chiama il metodo paintComponent di worldPanel, che non fa altro che chiamare il metodo draw del LevelController passandogli un oggetto di classe Graphics2D, classe di JavaSwing per il disegno di immagini su schermo. Tale metodo chiama, nel seguente ordine, i metodi drawWorld(), drawEnemies(), drawCharacter e drawPlayer(), che si occupano di richiedere a Swing il disegno su schermo del livello e dei personaggi. La logica di disegno si basa sul seguente criterio: poiché la schermata di esplorazione è suddivisa in righe e colonne, all’interno di un doppio ciclo for si esplorano le possibili combinazioni di indici (i, j), prendendo dall’oggetto map immagini di gioco, nemici e personaggi, invocando per ognuno di essi il metodo drawImage() di Graphics2D. Per il giocatore, invece, è sufficiente chiamare una sola volta tale metodo. Questo processo viene ripetuto 60 volte per secondo finché, a causa di un evento, la variabile booleana active viene settata a false, facendo terminare l’esecuzione del metodo runController().

Poiché il disegno delle entità di gioco si basa su meccanismi interamente gestiti da Java Swing e AWT, non è stato possibile sviluppare test di unità per questa parte, in quanto non vi è modo di verificare, se non visivamente, che ogni entità di gioco sia stata correttamente disegnata.