Alma Mater Studiorum · Università di Bologna

Corso di
Paradigmi di Programmazione e Sviluppo
Docenti: Mirko Viroli, Roberto Casadei

Lost 'n Souls
Roguelike game

Autori:
Matteo Brocca · 1005681
Alan Mancini · 1005481
Federico Mazzini · 980559

Settembre 2021

Indice

Introduzione

Il progetto scelto mira a sviluppare un single player game di genere Roguelike, ispirato al videogioco The Binding of Isaac.

Scopo del gioco, per un giocatore, è controllare un personaggio all'interno di un dungeon composto in modo procedurale da un numero finito di stanze definite in maniera procedurale. All'interno delle stanze saranno presenti in maniera casuale elementi bloccanti, nemici e oggetti. Il giocatore dovrà cercare di sconfiggere i nemici per passare alla stanza successiva e raccogliere oggetti al fine di potenziarsi o cambiare le sue caratteristiche. Una partita termina con la sconfitta del boss, un nemico unico presente all'interno del dungeon in una specifica stanza, o con la morte perenne del personaggio, dove i progressi fatti durante il gioco vengono persi.

1 Processo di sviluppo adottato

Il processo di sviluppo adottato dal team è incrementale e iterativo. Si è cercato il più possibile di attenersi al framework **Scrum**, adattato alle esigenze lavorative e scolastiche dei membri del team. Il team ha effettuato **sprint** settimanali, in modo tale da massimizzare il numero di cicli iterativi di sviluppo. Inizialmente, gli sprint hanno avuto una parte di **planning**, mentre al loro termine, una parte di **review**. Di seguito si analizza nel dettaglio il metodo utilizzato.

1.1 Meeting

Ai meeting ha sempre partecipato il team al completo. Al bisogno, Matteo ha svolto il ruolo di esperto di dominio/committente. Ogni scelta all'interno del progetto è comunque sempre stata condivisa da tutto il team.

1.1.1 Sprint planning

Lo sprint planning è svolto all'inizio di ogni Sprint ed è di fondamentale importanta in quanto permette di definire nel dettaglio i task da eseguire all'interno dello sprint e i goal per esso. Ogni sprint planning si compone di:

- Raffinamento del product backlog e identificazione dei goal per lo sprint.
- Definizione dei **Task** come unità di lavoro pratica per soddisfare i **requisiti**;
- Assegnazione dei Task ai membri del team.

Lo Sprint Planning ha una durata massima di 2 ore.

1.1.2 Daily Scrum

Durante il Daily Scrum ogni sviluppatore espone al team i seguenti punti:

- Quale lavoro ha svolto la giornata precedente;
- Quale lavoro intende svolgere nella giornata corrente;
- Eventuali possibili impedimenti per il lavoro da svolgere, e come gli altri membri del team potrebbero aiutare ad affrontare il problema.

La durata di questo incontro è al massimo di 15 minuti.

1.1.3 Sprint review

La Sprint review analizza l'iterazione appena avvenuta e si concentra sul prodotto software in sè, in particolare si discute di:

- Ispezione dell'incremento ottenuto in termini di funzionalità e risultati tangibili per il cliente
- Adattamento del Product Backlog;
- Discussione su ciò che potrebbe essere fatto nel prossimo Sprint, utile come preparazione al prossimo Sprint Planning.

Durata massima: 1 ora.

1.1.4 Sprint retrospective

La Sprint retrospective analizza l'iterazione appena avvenuta concentrandosi sul processo di sviluppo e il team, in particolare si discute di:

- Come sono stati utilizzati i tool per il team e si analizza l'andamento dei meeting
- Idee per migliorare il processo di sviluppo, in particolare i punti critici individuati al punto sopra

Durata massima: 45 minuti.

1.2 Divisione dei Task

La suddivisione dei task è su base volontaria. Questo significa che tutti i membri del team si offrono volontariamente per svolgere un determinato task, nei limiti ovviamente della totalità dei task e dei goal per lo specifico sprint. Durante il daily scrum, può essere rivista qualche decisione, se non troppo radicale.

1.3 Definition of done

Il team ha definito, come e in che modo, un task può essere definito come done e di conseguenza concluso

1. Superamento di tutti gli Scalatest
2. Codice documentato con opportuna Scaladoc
3. Code review

1.4 Tool

Il team ha individuato i seguenti strumenti per favorire un processo agile, migliorare l'efficienza e favorire l'automazione durante il processo di sviluppo

- **SBT** come strumento di build automation
- **Scalatest** per la scrittura ed esecuzione dei test automatizzati
- **GitHub** come strumento per la *continuous integration*
- **Jira** come strumento a supporto di scrum, gestione del product backlog e delle varie board di sviluppo

2 Requisiti

2.1 Requisiti di business

- Creazione di un gioco di genere Roguelike
- Possibilità di gioco su browser

2.2 Requisiti utente

Matteo Brocca in questo progetto ricopre il ruolo di esperto di dominio e committente. E' un appassionato di giochi Roguelike ma essendo troppo bravo li ha finiti tutti. Da qui l'idea di crearne uno nuovo per lui.

1. L'utente potrà
   - 1.1 avviare una nuova partita da un menu contestuale
   - 1.2 controllare il proprio personaggio all'interno della stanza per
     - 1.2.1 spostarsi e cambiare direzione
     - 1.2.2 sparare ai nemici
     - 1.2.3 raccogliere elementi utili all'aumento delle sue caratteristiche
     - 1.2.4 spostarsi da una stanza all'altra attraverso delle porte
     - 1.2.5 visualizzare in modo continuo le sue statistiche e i suoi punti vita
2. L'utente dovrà riuscire a
   - 2.1 distinguere nemici, oggetti e elementi di disturbo
   - 2.2 capire in che direzione si sta muovendo
   - 2.3 capire dove sta sparando
   - 2.4 capire di aver colpito un nemico
   - 2.5 capire di aver raccolto un oggetto
   - 2.6 capire qual'è la stanza del boss
   - 2.7 capire che sta fronteggiando un boss
   - 2.8 capire di aver vinto o perso una partita

2.3 Requisiti funzionali

1. Menù di gioco
   - 1.1 presenza di un tasto per avviare una nuova partita
2. Generazione di una mappa di gioco 2D in maniera casuale
   - 2.1 Una mappa è formata da più stanze quadrate
   - 2.2 Ogni stanza è fisicamente adiacente ad almeno un'altra stanza
   - 2.3 Tra stanze adiacenti è presente una porta che le collega
   - 2.4 Una stanza è di diverse tipologie
     - 2.4.1 Iniziale
       - 2.4.1.1 Vuota
       - 2.4.1.2 Una sola nella mappa
     - 2.4.2 Oggetto
       - 2.4.2.1 Contiene un singolo oggetto scelto randomicamente
     - 2.4.3 Combattimento
       - 2.4.3.1 Il contenuto è generato randomicamente
       - 2.4.3.2 Contiene alcuni nemici di una singola tipologia
       - 2.4.3.3 Contiene alcuni elementi bloccanti
       - 2.4.3.4 Le porte si chiudono quando il giocatore entra nella stanza
       - 2.4.3.5 Le porte si aprono quando il giocatore ha eliminato tutti i nemici
     - 2.4.4 Boss
       - 2.4.4.1 Contiene solamente il boss
       - 2.4.4.2 Una sola nella mappa
       - 2.4.4.3 Adiacente ad una ed una sola altra stanza
       - 2.4.4.4 Riconoscibile grazie ad una porta particolare
   - 2.5 A video è visibile solo la stanza dove è presente il personaggio
3. Personaggio controllabile con caratteristiche e punti vita
   - 3.1 Il personaggio è caratterizzato da
     - 3.1.1 Punti vita
     - 3.1.2 Velocità di movimento
     - 3.1.3 Danno
     - 3.1.4 Rate di fuoco
   - 3.2 Il personaggio è controllato dall'utente con la tastiera

        3.2.1 L'utente può muovere il personaggio nelle quattro direzioni principali (sopra, sotto, destra, sinistra)
        3.2.2 L'utente può eseguire uno o più "shot" verso una delle quattro direzioni principali (sopra, sotto, destra, sinistra)
        3.2.3 Il personaggio è vincolato a stare nei limiti della stanza e può uscire solo dalle porte
    3.3 Il personaggio infligge danno ai nemici quando uno "shot" colpisce un nemico
    3.4 Il personaggio può raccogliere oggetti che modificano le sue caratteristiche
4. Nemici con diverse caratteristiche
    4.1 Un nemico può essere di diverse tipologie
        4.1.1 A (nome del nemico)
        4.1.2 B
        4.1.3 C
    4.2 Ogni tipologia è caratterizzata da
        4.2.1 Punti vita
        4.2.2 Velocità di movimento
        4.2.3 Danno
        4.2.4 Modalità di movimento
            4.2.4.1 Casuale all'interno di una stanza
            4.2.4.2 In direzione del giocatore
    4.3 Un nemico infligge danno al personaggio quando lo tocca
    4.4 Un nemico si muove solo all'interno di una stanza
5. Oggetti con diversi comportamenti
    5.1 Un oggetto può essere di diverse tipologie
        5.1.1 A (nome dell'oggetto)
        5.1.2 B
        5.1.3 C
    5.2 Un oggetto in base alla sua tipologia varia alcune caratteristiche del giocatore

2.4 Requisiti non funzionali

1. Fluidità di gioco

1.1 Il gioco non deve presentare lag o inestetismi marcati che rendano la user experience non ottimale

2. Bilanciare numero e caratteristiche dei nemici per ottenere un livello di difficoltà di gioco medio

   2.1 Il gioco non deve essere troppo facile

      2.1.1 Presenza di pochi nemici all'interno di una stanza

      2.1.2 Presenza di oggetti che potenzino le caratteristiche del giocatore rendendo il conflitto giocatore-nemici impari

   2.2 Il gioco non deve essere troppo difficile

      2.2.1 Presenza di troppi nemici all'interno di una stanza

      2.2.2 Presenza di oggetti che non potenzino o riducano le caratteristiche del giocatore rendendo il conflitto giocatore-nemici impari

2.5 Requisiti di implementazione

1. Implementazione mediante Prolog del movimento dei nemici

2. Testing mediante ScalaTest

3. Sviluppo del software in modo da poter aggiungere nuovi componenti o estensioni a quelli gia esistenti in modo semplice

   3.1 Aggiunta di diversi personaggi con cui giocatore

   3.2 Aggiunta di nuovi nemici e boss

   3.3 Aggiunta di nuovi elementi bloccanti

   3.4 Aggiunta di nuovi oggetti con diverse caratteristiche

4. Rilascio del gioco su server tramite GitHub Actions

3 Design architetturale

Di seguito vengono analizzate l'architettura complessiva dell'applicazione e le decisioni critiche riguardanti pattern architetturali e tecnologie rilevanti. Prima di pensare a qualsiasi architettura per il gioco da realizzare, ciò che si è cercato di fare è stato individuare l'engine ideale per il progetto. La scelta è ricaduta su Indigo in quanto:

- E' un framework orientato al game development, strutturato in modo da favorire l'implementazione di componenti riusabili.
- E' scritto in Scala e favorisce lo sviluppo tramite paradigma funzionale.
- Permette il gioco su browser, come da requisito di business, mediante compilazione tramite Scala.js.

I concetti chiave di Indigo che ci guidano nello sviluppo del gioco sono

- Model, View, ViewModel
- Game loop con modello ad eventi
- Scene

3.1 Model, View, ViewModel

Il pattern architetturale **MVVM** (Model, View, ViewModel) è una variante del famoso **MVC** (Model, View, Controller). Scopo di questo pattern è quello di separare il modello dei dati dalla sua rappresentazione, tenendo entrambi puri e funzionali al loro scopo, frapponendo tra loro un elemento "ibrido", il ViewModel.

Model Rappresenta il modello di gioco puro, indipendentemente dalla sua rappresentazione visuale. Esso contiene lo stato del gioco e la sua logica.

ViewModel Si interpone tra Model e View. Ha lo scopo di mantenere alcuni dati utili per la rappresentazione che tuttavia non concernono la logica di gioco, devono perciò rimanere separati e non essere inclusi all'interno del Model.

View Si occupa di offrire una rappresentazione del Model e del ViewModel a schermo. Mantiene quindi la logica di rappresentazione di questi, senza contenere però alcuna logica di gioco.

3.2 Game loop con modello ad eventi

La logica di Indigo si basa su un game loop che processa, ad ogni iterazione, una coda di eventi. Ad ogni ciclo, il framework esegue le seguenti operazioni:

- Per ogni evento, update del Model
- Per ogni evento, update del ViewModel
- Presentazione e rendering degli elementi a video
- Reset della coda di eventi

Il concetto di immutabilità, presente nel paradigma funzionale, qui si traduce non in un update di uno stato interno del modello, ma bensì nella generazione di un nuovo modello aggiornato.

3.3 Scene

Le scene sono un modo di organizzare il codice secondo una logica di gioco ben definita. Sono un meccanismo di suddivisione che permette di individuare delle "fasi" di gioco da sviluppare in modo separato le une dalle altre. In ogni istante all'interno del gioco, il modello, l'aggiornamento e la presentazione sono delegate alla scena corrente in esecuzione, la quale si occupa anche di gestire gli eventi.

Abbiamo identificato quattro scene principali su cui andare a definire Lost 'n Souls.

4 Design di dettaglio

Design di dettaglio (scelte rilevanti, pattern di progettazione, organizzazione del codice – corredato da pochi ma efficaci diagrammi)

5 Implementazione

Implementazione (per ogni studente, una sotto-sezione descrittiva di cosa fatto/co-fatto e con chi, e descrizione di aspetti implementativi importanti non già presenti nel design)

6 Retrospettiva

6.1 Sprint 1

Svolgimento e sviluppo dello sprint

Considerazioni finali

6.2 Sprint 2

Svolgimento e sviluppo dello sprint

Considerazioni finali

6.3 Sprint 3

Svolgimento e sviluppo dello sprint

Considerazioni finali

6.4 Sprint 4

Svolgimento e sviluppo dello sprint

Considerazioni finali

6.5 Sprint 5

Svolgimento e sviluppo dello sprint

Considerazioni finali

6.6 Sprint 6

Svolgimento e sviluppo dello sprint

Considerazioni finali