Pandora Quest

PROJECT PLAN

Version 2.3
January 31, 2022

INTRODUCTION

Il progetto è basato sulla creazione di un gioco in formato web app. Il gruppo di progetto è formato da Daidone, Drissi, Ianitchii, Marchesi.
L'obiettivo è quello di mettere in pratica le conoscenze acquisite durante il corso di Ingegneria del software, e anche di altre già in nostro possesso, per creare un prodotto apprezzabile da altre persone.
La scelta di creare un gioco è stata condivisa da tutti ed è nata principalmente dalla voglia di creare qualcosa di diverso dalla solita applicazione che svolge un compito o che soddisfa un bisogno.

PROCESS MODEL

Per lo sviluppo di questo progetto abbiamo deciso di utilizzare l'approccio agile poiché riflette al meglio la modalità di lavoro che vorremmo adottare.
L'idea è quella di partire da un prototipo base, creato in parallelo alla fase di progettazione, che rappresenta il prodotto con l'implementazione delle funzionalità base. Successivamente, tramite miglioramenti incrementali, vogliamo permettere l'evoluzione del prototipo in un prodotto finito. A fine di ogni intervento vogliamo ottenere un prodotto sempre funzionante.

ORGANIZATION OF THE PROJECT

Per la divisione del lavoro abbiamo deciso di sfruttare le capacità di ciascun membro del team assegnando del lavoro in base alla specializzazione del singolo componente. Questo ha permesso di utilizzare meglio il tempo ed ottenere una qualità finale migliore. Ogni membro del team possiede comunque una visione generale di tutto quello che sta succedendo.
I punti di forza dei componenti del gruppo sono:

- Daidone: documentation/backend/content creator
- Drissi: documentation/content creator
- Ianitchii: backend/documentation/product owner
- Marchesi: frontend/backend/teasting/documentation/chief programmer

STANDARDS, GUIDELINES, PROCEDURES

Come programmi software/tools abbiamo utilizzato:

- Github
- Vscode
- Vim
- Google docs
- IEE…
- Camel case per nomi classi
- Underscore per nomi file
- Flutter

- Dart
- Staruml
- ....

MANAGEMENT ACTIVITIES

Data la scarsità di tempo, abbiamo deciso di eseguire meno incontri ma più sostanziosi, soprattutto una volta finita la fase di progettazione iniziale. Ciascuna riunione ha come obbiettivo quello di aggiornare i membri del gruppo sul lavoro svolto e, tramite un prototipo sempre funzionante, avere un'idea chiara del progresso eseguito e di eventuali cambiamenti necessari. Per la lista degli incontri più importanti eseguiti vedi file: IS PW - meetings

RISKS

Tra i possibili rischi di questo progetto quello più imponente è quello di non riuscire a rispettare la deadline fissata per la consegna del prodotto. Tra gli altri rischi possono essere individuati:

- Non riuscire a completare i compiti assegnati a ciascun individuo
- Aggiungere troppi funzionalità (rischio del prototyping)
- Non riuscire a rispettare tutti i requisiti funzionali definiti
- Prodotto non funzionante a fine giornata

STAFFING

Data la quantità limitata di personale abbiamo cercato di sfruttare al meglio le nostre abilità e abbiamo cercato di modellare le nostre conoscenze in base alla situazione ed alla richiesta.

METHODS AND TECHNIQUES

Durante il progetto abbiamo deciso di:

- Commit su github per avere la versione più aggiornata sempre disponibile
- Code reviews per sezioni critiche tramite github
- Creazione di pull request prima di fare modifiche importanti sul main
- Creazione di branch per sezioni critiche. Si esegue il merge solamente quando la sezione è completa e funzionante.
- Utilizzo di tag identificativi per i componenti del software per maggiore chiarezza e leggibilità.
- Creazione di issue su github per eventuali bug/cambiamenti

Durante il progetto sono state eseguite diverse sessioni di pair programming, utili per avere una visione più ampia su metodi di implementazione delle funzionalità
Per quanto riguarda il design l'idea è di pensare a cosa contiene ciascuna delle pagine, sia a livello di UI che a livello di funzionamento e successivamente implementarlo nel prototipo. Il tutto è stato reso possibile da Flutter e dalla sua velocità di implementazione delle funzionalità. Questo aiuta nella definizione di eventuali requisiti aggiuntivi, dato che non

avendo un vero e proprio cliente ed essendo il prodotto qualcosa di diverso dal solito, non è semplice definire fin da subito tutto quello che si vuole.

Per l'implementazione, poiché l'obbiettivo è di avere un prodotto sempre funzionante, durante la scrittura iniziale del codice il focus è sul far funzionare la nuova funzionalità e successivamente eseguire refactoring per aumentare efficienza, leggibilità e qualità generale del codice.

QUALITY ASSURANCE

Per la qualità del codice ed il rispetto delle best practices sono stati utilizzati strumenti di linting.
Durante lo sviluppo abbiamo eseguito tests utilizzando persone esterne per assicurarci che il prodotto fosse comprensibile anche a elementi esterni al progetto. Da queste persone abbiamo poi raccolto dei feedback

WORK PACKAGES

Le attività principali del progetto sono:

- identificazione della tipologia di prodotto da sviluppare
  schematizzazione tramite diagrammi UML
- documentazione e sviluppo software in contemporanea
- ricerca contenuti (immagini/gif)

I compiti, eccetto la creazione dei grafici UML, sono stati portati avanti individualmente, utilizzando poi le riunioni di gruppo per mostrare i risultati ottenuti o per la risoluzione di eventuali problemi.

RESOURCES

Per lo sviluppo dell'applicazione si prevede di utilizzare:

- Librerie esterne (flutter packages)
- Chrome browser per l'esecuzione dell'applicazione
- PC propri
- Conoscenze acquisite dai materiali del corso e da internet

BUDGET AND SCHEDULE

In questo progetto non è stato presente un budget a livello monetario ma bensì un budget temporale. La risorsa limitata è stata il tempo e la conoscenza. Il tempo previsto per il completamento del progetto è entro il centinaio di ore per persona.

CHANGES

Per gestire i cambiamenti abbiamo deciso di tenere una traccia precisa di tutte le versioni dei file soggetti a modifiche nel tempo, grazie al supporto diretto di github.

Dato il metodo di lavoro adottato i cambiamenti sono all’ordine del giorno. L'obiettivo è quello di avere sempre un prodotto funzionante a fine giornata.

DELIVERY

Il prodotto finale verrà hostato è sarà disponibile agli utenti in formato web app.

Pandora Quest

Specifica dei requisiti del gioco

Version 1.3
January 31, 2022

1. Introduzione

1. Scopo

Lo scopo del progetto è quello di acquisire conoscenze pratiche per quanto riguarda la realizzazione di un prodotto software. Partendo dall'idea, adottando un approccio metodologico, utilizzando diversi strumenti e linee guida definiti dall'ingegneria del software, arrivare ad ottenere un prodotto funzionante. Il focus è imparare ad utilizzare le metodologie insegnate a lezione.

2. Scopo del prodotto

Il prodotto è un gioco 2D eseguibile su web. Il gioco vuole trascinare l'utente in un ambiente mistico in cui dovrà esplorare e affrontare diverse sfide. La dinamica del gioco è definita dal dialogo che guida il giocatore nel compiere certe azioni.
Esistono stanze esplorazione dove si possono trovare e raccogliere oggetti che aiuteranno l'utente ad affrontare la seconda tipologia di stanza. Che sono stanze da combattimento. Un combattimento contiene un nemico che sfida il giocatore mettendo alla prova le sue conoscenze su diversi argomenti attraverso delle domande, per le quali vengono visualizzate le risposte e l'utente deve scegliere quella giusta.

3. Glossario

Stanza Esplorazione - una stanza con un dialogo che fa compiere delle azioni al giocatore;
Stanza Combattimento - stanza in cui è presente un dialogo guidato da parte del nemico. Il dialogo è composto da una conversazione, la quale comprende delle domande su un certo argomento.
Pagina Statistiche - pagina finale sulla quale vengono presentati alcuni parametri della partita, domande fatte, risposte corrette, vita rimanente, tempo impiegato, etc.
Giocatore - il protagonista del gioco in base alle azioni del quale evolve lo scenario;
Nemico - lo sfidante, che può essere trovato solo in stanze da combattimento;
Azione - una mossa che può svolgere il giocatore cliccando sui tasti delle azioni, esempio: raccogli oggetto;
Oggetto - un artefatto che può essere trovato nelle stanze di esplorazione e/o di combattimento. Un oggetto può essere un'arma da combattimento, una protezione oppure una sostanza o amuleto che può favorire o danneggiare il personaggio;

4. Riferimenti

- IEEE 830-1998
- Slide lezioni
- Software Engineering: Principles and Practise by Hans van Vliet

5. Panoramica

Questo documento è basato sulla struttura dello standard IEEE 830-1998.
L'informazione che verrà a seguire in questo documento è:

- Sezione 2: Descrizione generale del prodotto, con degli screenshots dell'interfaccia utente;

- Sezione 3: definizione di requisiti funzionali, non funzionali e i requisiti non implementati.

2. Descrizione generale

Questo prodotto deve essere un’applicazione web fruibile tramite qualsiasi browser. Una volta lanciato il gioco, l’utente deve essere coinvolto dalla trama del gioco che lo guiderà per l’intero percorso fornendogli un passatempo interessante attraverso il quale esso possa divertirsi e mettere alla prova le proprie conoscenze e possibilmente scoprire qualcosa di nuovo.

1. Prospettiva del prodotto

Nel caso migliore l’applicazione suscita un interesse nell’utente per certi argomenti. Che poi possa andare a curiosare e scoprire qualcosa di nuovo. Inoltre ponendo domande su vari argomenti, bisogna dare una soddisfazione dell’esperienza, sperando che l’utente dia il maggior numero di risposte corrette per trovarsi soddisfatto a fine gioco quando vedrà la pagina con le statistiche. Per suscitare più interesse sul percorso ci saranno delle stanze di esplorazione in cui l’utente può trovare e raccogliere oggetti che faciliteranno il suo percorso oppure lo metteranno in maggior difficoltà.

1.1 Interfaccia utente

L’utente per usare il gioco ha bisogno di alcune periferiche:

- Input: mouse
- Output: monitor

1.2 Interfaccia hardware

Non necessità di alcuna interfaccia hardware.

1.3 Interfaccia software

Non ci sono interfacce software, tutti i componenti necessari vengono presi in locale.

2. Funzionalità del prodotto

Le funzionalità del prodotto sono le seguenti:

- Inserimento del nome da parte dell'utente per avviare il gioco;
- Svolgere delle azioni attraverso i tasti con un numero limitato di azioni che riguardano il contesto in cui si trova l’utente;
- Combattere con un Nemico rispondendo a delle domande a scelta multipla azionando i tasti con le risposte;
- Raccogliere oggetti;
- Equipaggiare un'arma (arco, spada, scudo) prima o durante il combattimento;
- Utilizzare amuleti;
- Visualizzare le statistiche della partita.

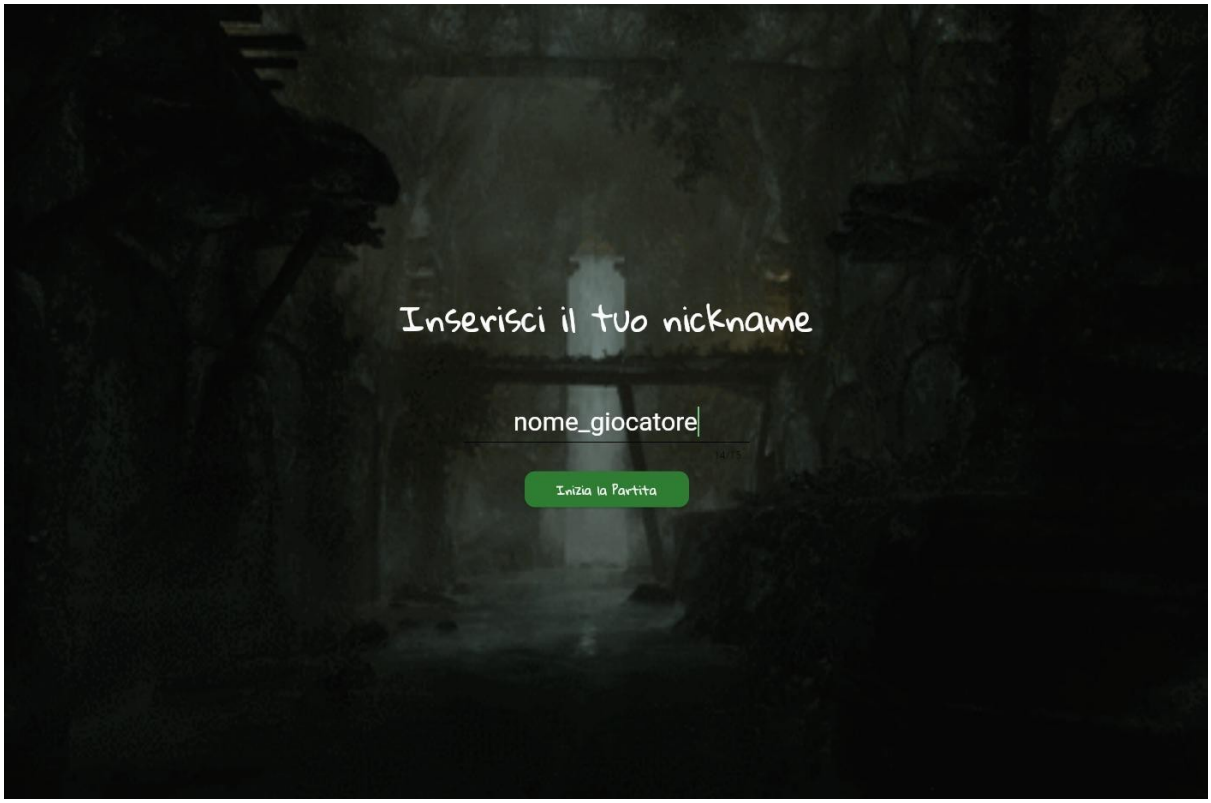
3. Assunzioni e dipendenze

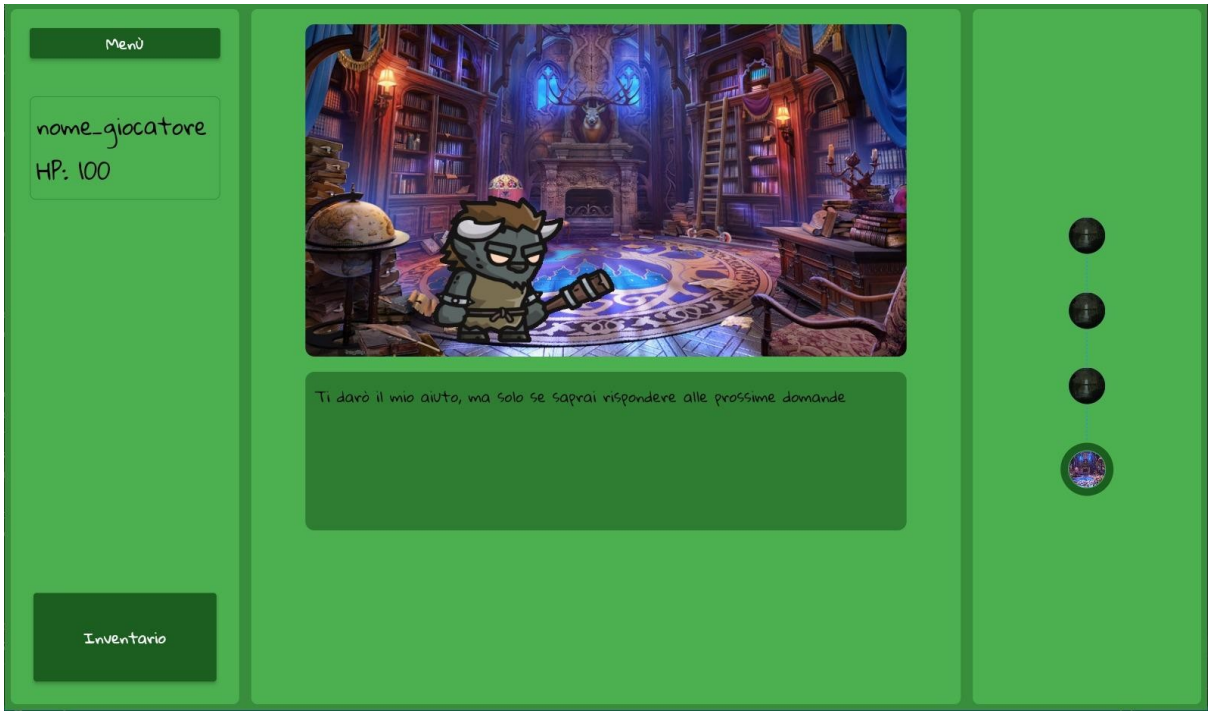
Nella versione finale il gioco non sarà disponibile online. Ma verrà ancora presentato e dimostrato il funzionamento in locale. Quindi la macchina sulla quale verrà presentato il progetto dovrà aver installato gli ambienti corretti per riuscire a costruire il progetto e farlo eseguire in locale. Fare riferimento al Project Plan, paragrafo: STANDARDS, PROCEDURES, GUIDELINES per gli strumenti software necessari per il funzionamento.
Per lo stack di tecnologie usate, non si hanno dipendenze esterne.

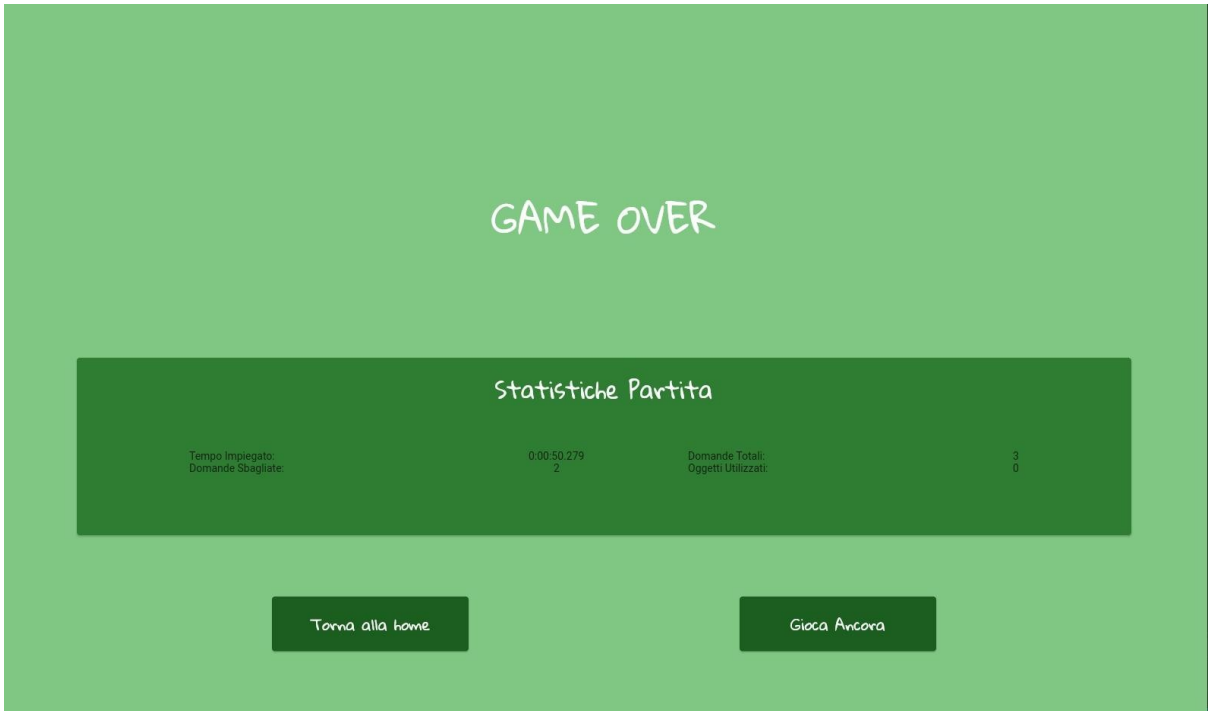
3. Specifica dei requisiti

1. Interfaccia utente

Tutte le interazioni dell'utente con il prodotto avvengono attraverso una web UI.


Inserisci il tuo nickname
nome_giocatore
Inizia la Partita


Menù
nome_giocatore
HP: 100
Ti darò il mio aiuto, ma solo se saprai rispondere alle prossime domande
Inventario



1. Requisiti funzionali del sistema

- Il sistema deve mettere alla prova le conoscenze dell'utente e possibilmente motivare per approfondire argomenti relativi a domande a cui non è riuscito a rispondere;
- Deve essere un passatempo piacevole e utile per l'utente;
- Bisogna tener traccia dell'evoluzione del gioco, ovvero delle statistiche;
- Il sistema deve cambiare stanza una volta il dialogo di una stanza si è concluso;
- Il gioco deve essere dinamico, ad ogni partita lo scenario deve essere simile ma diverso da quello precedente.

2. Requisiti non funzionali

- L'interfaccia deve essere intuitiva e facilitare l'utente a focalizzarsi sulle domande dei combattimenti;
- Il sistema deve essere veloce, il caricamento di qualsiasi componente della pagina deve essere istantaneo senza attese per l'utente;

3. Requisiti non implementati

- Accessibilità del gioco da qualsiasi dispositivo attraverso un browser;
- Animazione dei personaggi che possono fare una conversazione anche attraverso l'audio.

Pandora Quest

Project Documentation

Version 2.0

January 31, 2022

SOFTWARE LIFE CYCLE

Il nostro gruppo ha deciso di adottare un metodo di sviluppo agile per lo sviluppo di questo progetto. Essendo un tipo di prodotto non convenzionale e di dimensioni relativamente ridotte, eseguire un approccio model driven sarebbe stato controintuitivo e poco funzionale. Il nostro metodo di lavoro è stato simile all'extreme programming, ma senza delle finestre di tempo definite, ogni giorno bisognava fare il possibile. Definire i requisiti fin da subito era complesso, quindi abbiamo deciso di eseguire del prototyping (in particolare Evolutionary Prototyping). Poiché eravamo noi i clienti, i prototipi ci hanno aiutato a visualizzare il prodotto e ad applicare modifiche di volta in volta. Il codice doveva sempre essere funzionante ad ogni commit quindi abbiamo usato branch per evitare di implementare funzionalità che rompessero quello che era già presente, ed eseguire un merge una volta assicurato che tutto funzionasse. L'evoluzione è stata incrementale, ogni commit ha portato a delle nuove funzionalità, senza però stravolgere l'intero funzionamento. Gli incontri sono stati poco frequenti ed è stato eseguito molto refactoring.

CONFIGURATION MANAGEMENT

Github è stato il punto centrale per la sincronizzazione del lavoro tra i membri del gruppo. Esso ci ha aiutato a definire le task da eseguire e a visualizzare il progresso del lavoro.
Ogni commit rappresenta i cambiamenti eseguiti nel codice con la corrispettiva descrizione di cosa è stato modificato (Change Oriented).

PEOPLE MANAGEMENT AND TEAM ORGANIZATION

Per la gestione del team abbiamo deciso di adottare della burocrazia professionale, dove ognuno aveva autonomia nello svolgere la propria task.
Per quanto riguarda il focus, esso era rivolto verso il risultato finale (Task Directedness) dato che l'obiettivo era quello di ottenere un prodotto di qualità, gradevole da usare e di cui essere fieri.
In generale la team organization è stata del tipo: Chief Programmer Team. Dove Marchesi è il chief grazie alla sua discreta conoscenza di Flutter e visione del progetto, mentre il resto del gruppo ha agito come lavoratori specializzati che hanno contribuito allo sviluppo portando avanti task specifiche e producendo risultati di qualità.

SOFTWARE QUALITY

Ecco la lista di qualità/qualità d'uso che abbiamo deciso di implementare nel nostro prodotto:

- Effectiveness & Satisfaction: Il prodotto ci ha sempre soddisfatto durante lo sviluppo per i risultati ottenuti sul prototipo. Il prodotto ha centrato l'obbiettivo di essere un gioco divertente, rigiocabile e informativo.
- Usability: pensiamo che i comandi siano intuitivi e facili da utilizzare. Qualsiasi giocatore dovrebbe essere in grado di capire in poco tempo come il gioco funziona e quali meccaniche esso comprende.

- Maintainability: Durante la scrittura del codice è stata eseguita una massiccia documentazione delle classi e dei metodi utilizzati. Questo in aggiunta alla documentazione esterna e ad una efficiente suddivisione delle cartelle, rende la manutenzione veloce e semplice, nonostante la notevole quantità di codice e documenti.
- Portability: Il codice è stato scritto con Flutter, il che rende il prodotto multiplatform, inoltre durante lo sviluppo del codice sono state tenute conto le best practices che permettono di far adattare il software a diversi dispositivi con diverse dimensioni di schermo.

Cosa si potrebbe migliorare:

- Reliability: aumentare il numero di test ed espanderli anche all'interfaccia grafica.
- Efficiency: Distribuire in modo migliore la creazione della partita
- Maintainability: Aumentare la suddivisione tra Model View e Controller.

SOFTWARE ARCHITECTURE

Il nostro prodotto può essere suddiviso in componenti e connettori. Per quanto riguarda i componenti, essi sono rappresentati dalle classi. Ognuna di esse ha dei ruoli specifici, c'è che funge da memoria (es. Stanza, Personaggio), altri che fungono da elementi computazionali (es. CreazionePartita) ed infine il componente manager (Partita). I due tipi principali di connettori sono: Procedure Call e Implicit Invocation. Il primo viene utilizzato molto spesso da classi e widget che chiamano delle RPC per far eseguire delle operazioni, mentre il secondo tipo di connettore è utilizzato grazie alla libreria Provider e ChangeNotifier. Queste hanno permesso di costruire un sistema basato su eventi e su ascoltatori pronti a modificare il loro stato in base all'arrivo di eventi. Gli stili architetturali utilizzati sono quindi stati:

- Programma principale con subroutine: Programma principale definito dall'albero di widget, da cui ogni widget aveva a disposizione dei metodi da chiamare per poter far eseguire operazioni in remoto.
- Model View Controller: abbiamo cercato di dividere il più possibile la UI, dai dati e dalla logica del programma. Dentro i widget infatti, non viene eseguita quasi nessuna computazione e/o gestione dei dati. Tutto viene eseguito da funzioni in classi apposite.
- Implicit Invocation: Provider e ChangeNotifier ci hanno permesso di condividere le istanze di Partita e Personaggio lungo tutto l'albero di widget. Questo rappresenta lo stato della partita e tutte le informazioni necessarie per il funzionamento. Ogni volta che avvengono dei cambiamenti in queste due classi vengono generati degli eventi che vengono ascoltati da dei listeners che, in base al tipo di evento generato, reagiscono di conseguenza.

In generale per avere differenti punti di vista sull'applicazione a livello di granularità diversi, si possono utilizzare i diagrammi UML creati, che offrono sia una visione statica (Use case e Class) e sia una visione dinamica (Activity, Sequence, StateChart) della struttura.

SOFTWARE DESIGN

Per il design oltre ai grafici UML, abbiamo utilizzato dei tool per il calcolo della complessità e per il rispetto delle best practices. In particolare, abbiamo usato questa libreria di dart: https://pub.dev/packages/dart_code_metrics che tramite dei comandi specifici ci permette di visualizzare il numero di classi, metodi, file non utilizzati, presenza di antipattern ecc.
Ci permette inoltre di impostare un valore massimo soglia per diverse proprietà, esempio: numero max di metodi in una classe, numero max di parametri, difficoltà massima calcolata con metodo Halstead
Per quanto riguarda i pattern utilizzati abbiamo:

- Observer: Tramite il Provider abbiamo reso Observable l'istanza di partita e di personaggio. Le altre classi posso creare dei listeners rivolti verso questi oggetti. Ogni volta che una delle due istanza subisce dei cambiamenti il metodo notifyListeners() viene chiamato che procede ad avvisare del cambiamento avvenuto tutti gli ascoltatori
- Player Role: In questo caso abbiamo un'associazione tra Nemico e Stanza, dove però entrambi sono super classi; infatti, nemico può poi essere una sottoclasse Boss o Scagnozzo, mentre la stanza può essere una sottoclasse StanzaCombattimento o StanzaEsplorazione.
- Model View Controller: come spiegato in precedenza è stato implementato il MVC pattern per rendere più funzionale e migliorare la qualità del codice.

SOFTWARE TESTING

Abbiamo eseguito solamente Unit Test tramite gli strumenti forniti da dart. In un'applicazione pronta per essere mandata in produzione sarebbe bene anche scrivere test sul funzionamento della UI.
L'approccio di testing è stato quello di testare i metodi più critici e che in caso di fallimento potrebbero portare alla rottura dell'applicazione. Ovviamente l'ideale sarebbe stato scrivere test per qualsiasi metodo di qualsiasi classe utilizzata, ma data la quantità di tempo a disposizione e la dimensione del gruppo di sviluppo abbiamo puntato verso un approccio più mirato.
In particolare sono stati testati i metodi della classe Partita, della classe Stanza e di quella Personaggio utilizzando un approccio Error Based Testing.

SOFTWARE MAINTENANCE

In generale la manutenzione del nostro prodotto è stata principalmente di due tipi: Correttiva ed Perfettiva. Anche se il prodotto non è stato effettivamente deployato, da quanto è stato possibile giocare una partita dall'inizio alla fine, abbiamo considerato il nostro prodotto completo e quindi da quel punto in poi le modifiche eseguite sono state di manutenzione.
Durante lo sviluppo sono state eseguite molte attività di refactoring, quelle più frequenti sono state:

- Spostamento di codice da una classe all'altra (per separare model, view e controller)
- Rinominazione di variabili

- Riscrittura di metodi per eseguire la stessa operazione in modo più efficiente
- Scrittura di commenti per descrivere il funzionamento di metodi e il ruolo delle variabili

Solitamente il refactoring è stato eseguito dopo l'implementazione di ciascuna funzionalità. Questo perché una volta implementato un nuovo pezzo di codice funzionante, l'obbiettivo era di adattarlo alla qualità del codice già presente e renderlo più leggibile e mantenibile.