SOFTWARE LIFE CYCLE

Il nostro team 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

SOFTWARE DESIGN

SOFTWARE TESTING

SOFTWARE MAINTENANCE

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.