# NES

Carmine Marra, Stefano Mercogliano, Daniele Ottaviano, Francesco Vitale

# 1 Descrizione del processo

# 2 Avvio

## 2.1 Vision

Si vuol sviluppare un sistema, definito NES, che sia comprensivo di *emulatore* e *assemblatore*. L'assemblatore consentirà agli utenti del sistema di poter assemblare il proprio programma, scritto ovviamente secondo il modello di programmazione del processore di riferimento, e, consecutivamente, caricare questo programma nell'emulatore consentendo l'esecuzione di quest'ultimo.
L'utente deve poter essere in grado di poter scrivere il proprio programma, interagire con l'assemblatore e assemblare il proprio programma secondo le opzioni messe a disposizione dal sistema stesso; l'utente potrà specificare qualsiasi elemento sintattico appartenente all'ISA del NES: l'assemblatore sarà in grado di interpretare quanto viene specificato dal programma e tradurre il codice e predisporlo per l'esecuzione sull'emulatore.
D'altro canto l'utente può decidere di caricare qualsiasi programma assembler sull'emulatore, che sia il programma assemblato con l'assemblatore del sistema, oppure no. L'emulatore non è finalizzato solo all'esecuzione delle linee di codice, ma è destinato anche alla visualizzazione grafica di alcune proprietà interne della macchina emulata durante l'esecuzione del programma, come:

- Il comportamento del processore e dei suoi registri interni;
- La memoria RAM in qualsiasi range di indirizzi;
- Informazioni al contorno;
- Il comportamento della PPU e il rendering grafico;
- Il comportamento interno della APU;

## 2.2 Specifiche supplementari

L'architettura deve essere modulare, soprattutto in vista di un completo disaccoppiamento dell'emulatore dall'assemblatore: l'emulatore dev'essere in grado di poter caricare qualsiasi sorgente assemblato da qualsiasi altro assemblatore, a patto che l'estensione sia la stessa.
Essendo che assemblatore ed emulatore saranno completamente disaccoppiati, il sistema risulterà essere *robusto*; l'assemblatore, quindi, sarà un componente *riusabile*, perchè produrrà codice compliant con altri emulatori.
In vista di un approccio agile, e quindi incrementale ed evolutivo, *manutenibilità* e *evolvibilità* sono importanti: entrambi i sottosistemi, essendo complessi, si prestano a essere decomposti in moduli idealmente lascamente accoppiati: bisogna assecondare questa proprietà al fine di ottenere uno sviluppo incentrato sulle parti critiche del sistema, per poter poi *estendere* quanto già sviluppato nelle prime iterazioni.
Si vuole anche rendere il sistema quanto più *interattivo* possibile: data la natura dell'emulatore, il sistema stesso potrebbe risultare poco *usabile*. Le scelte di design dovranno in qualche modo rendere il sistema più user-friendly.

## 2.3 Glossario

# 3 Specifica dei requisiti

In questa sezione ci riconduciamo ai suggerimenti tratti dal Larman per la stesura dei requisiti funzionali. In particolare, ci concentreremo sulla stesura di artefatti che catturino degli use-case che risultino essere *goal-driven*, ossia incentrandoci sugli obbiettivi degli utenti che sfruttano il sistema.

## 3.1 Attori e Obiettivi

Attori:

- Utente

Obiettivi utente:

1. Esegui programma utente (Scegliendolo dalla lista o dal file system)
2. Gestisci Lista programmi (Carica/ Modifica/ Elimina)
3. Configura Emulazione (Velocità/Periferica)
4. Gestione Codice Sorgente (Scrivi/Salva/Carica)
5. Compila il Codice
6. Visualizza Stato Architettura

Descrizione in breve dei casi d'uso:

- EseguiProgramma: Il programmatore potrà eseguire il programma ed eventualmente visualizzare a schermo l'esecuzione. L'esecuzione potrà essere fatta sia in modalità standard che istruzione per istruzione.
- CaricaInLista: L'utente carica un programma dal suo computer all'applicazione. Il programma verrà aggiunto alla lista dei programmi eseguibili.
- EliminaDallaLista: L'utente seleziona dalla lista un programma e lo rimuove dalla lista.
- ConfiguraEmulazione: L'utente prima dell'esecuzione di un programma potrà cliccare su di un pulsante per visualizzare e modificare i parametri di emulazione come la Velocità dell'esecuzione, le periferiche di ingresso e di uscita, la grandezza dello schermo etc.

- CompilaCodice: Permette di compilare il codice scritto producendo un file macchina comprensibile per l'emulatore.
- VisualizzaStatoArchitettura: Il programmatore potrà cliccare sull'immagine di una specifica componente hardware emulata per verificare il comportamento della stessa durante l'esecuzione di un programma

# 4 Analisi dei requisiti

# 5 Design funzionale/non-funzionale

# 6 Prospettiva implementativa