Simulazione Evangelizzazione

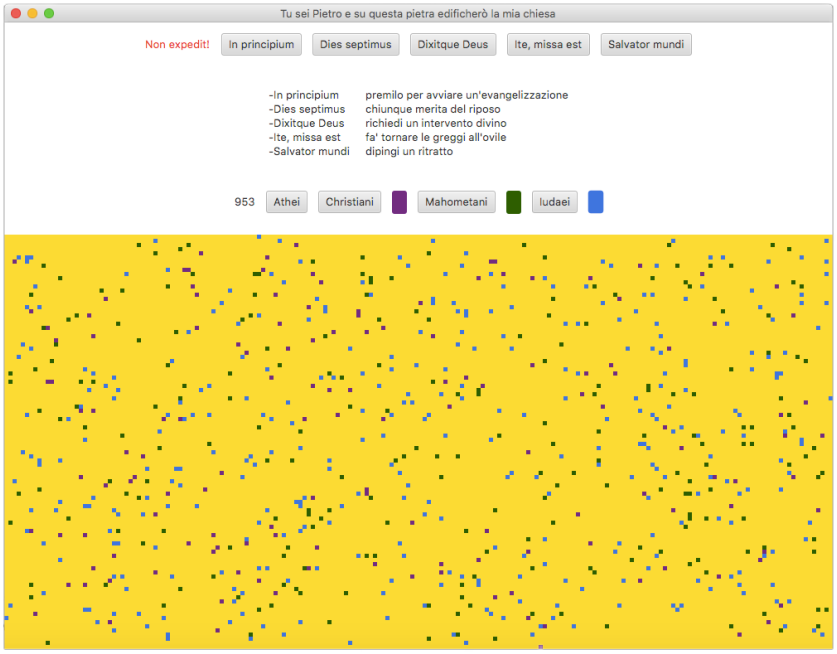
1 Introduzione

L'ambiente sviluppato simula lo scontro tra differenti fazioni rappresentate da celle, queste simulazioni prendono apertamente ispirazione dagli scontri - sempre frequenti nel passato come nel presente - tra i fedeli delle tre religioni abramitiche; l'interfaccia grafica stessa presenta riferimenti espliciti alla Chiesa Romana, ciò però non impone all'utente di dover obbligatoriamente giocare dalla parte dei cristiani. I fedeli interagiscono questi fedeli tra loro nella seguente maniera:

- Cristiani
  - Se un cristiano è a contatto con 3 o più cristiani muore.
  - A contatto con uno o due musulmani un perisce se però sono più di 3 (compreso) si giunge ad un accordo e il cristiano diviene un apostata.
  - Come sopra ma non è contemplata la conversione.
- Musulmani
  - Un musulmano che incontra 3 o più musulmani muore.
  - Se viene incontro a uno o due cristiani perisce, se sono 3 o più si converte.
  - Come prima.
- Ebrei
  - Se il numero di cristiani è pari a 1 o 2 questi hanno la meglio, se però il numero è maggiore si ha conversione.
  - Se si incontrano un numero di musulmani tra 1 e 3 questi prevalgono altrimenti si ha conversione.
  - Se il numero di ebrei è 3 o superiore si ha uno scontro interno in cui il fedele in minoranza ha la peggio.

2 GUI

Come si può vedere in figura l'interfaccia grafica si sviluppa su tre livelli (quattro includendo il testo contenente le istruzioni). In alto si hanno bottoni con differenti funzioni, dei bottoni che permettono la scelta dei fedeli da inserire sul campo e infine un pannello che funziona da tela (non è un oggetto di tipo `Canvas` ma `Pane`). Per poter colorare gli elementi grafici è stato usato anche il `CSS` (i bottoni atomici non potrebbero essere colorati altrimenti).



3 Codice

Il codice è diviso in due pacchetti, un pacchetto `gui` contenente la sola classe `gui` che gestisce la parte grafica e un pacchetto `dev` contenente la classe astratta `Player` con le sue sottoclassi `Christian`, `Muslim` e `Jew` e un'ultima classe `Util` che contiene variabili che ricorrono spesso sia in `gui` che in `dev`. Per gestire il multithreading con l'interfaccia è risultato necessario ricorrere alle classi generiche `Service` e `Task` - entrambe implementano l'interfaccia `Worker` - poiché in `JavaFX` l'interfaccia grafica è un thread la cui modifica da parte di un altro genera una `ConcurrentModificationException`, ciò si evita usando un oggetto della classe `Task` che genera un thread che si distacca dalla normale esecuzione e poi grazie a `Service` può comunicare con l'interfaccia grafica. Per leggere il colore di ogni cella inizialmente si scattava uno snapshot da cui si poteva leggere il colore di un pixel, processo però troppo lento e due cicli for annidati svolgono il lavoro in maniera più efficiente anche grazie all'utilizzo di una `HashMap` che prende le coordinate di un oggetto e l'oggetto stesso, eliminando dalla lista quegli oggetti che escono dai limiti del campo.
L'utilizzo di `Pane` ha inizialmente causato qualche problema riguardo la gestione della posizione di `Rectangle`, per ovviare a ciò è stata utilizzata la formula $c = c - (c \mod l)$ con l lunghezza del lato della `Shape`, portando però a perdere fluidità di movimento