# Assignment 1: Smart positioning

## 1. Analisi

L’assignement in questione proponeva il design e l’implementazione di un sistema che fosse in grado di posizionare in uno spazio a 2 dimensioni alcune particelle puntiformi, le quali si muovono nel piano influenzate dalle forze che sentono dalle altre particelle. Questo comportamento doveva poi essere esteso da una GUI in grado, oltre di visualizzare le particelle, di fermare e far riprendere il calcolo delle posizioni con i pulsanti start e stop. Il processo deve essere guidato anche dal numero di step di calcolo.

Le criticità su cui ci si è soffermati con più attenzione sono state:

* La suddivisione in più thread del lavoro
* L’aggiornamento dei dati
* Il controllo del flusso sia per il numero di step richiesti, che per la pressione dei pulsanti start e stop.

## 2. Soluzione proposta

A fronte delle criticità analizzata al punto precedente, si è scelto di implementare una applicazione in MVC dotata di un controller che permette di reagire all’input dell’utente e di avviare la computazione. Dal

E’ il controller infatti che permette di avviare il thread **MainWorker** il quale si fa carico della suddivisione del lavoro e avvia i **ParticleWorker**, che calcolano le forze, le applicano, e si mettono in pausa.

Attraverso un monitor, **StopFlag**, passato a tutti i thread, viene controllato lo stato del sistema, e qualora fosse a false si interrompe la computazione di tutti i thread.

Un altro monitor, **Counter**, si fa carico del conteggio del numero di step eseguiti, e viene controllato dal MainWorker ad ogni iterazione.

Per la sincronizzazione dei thread, invece, si è scelto di utilizzare altre 2 strutture:

* Un monitor, **ProceedMonitor**, che permette di mettere in attesa i ParticleWorker al termine della loro computazione, e di risvegliare questi da parte del MainWorker all’iterazione successiva. Questo comportamento evita di ricreare nuovamente i thread, mettendoli in pausa e risvegliandoli all’occorrenza.
* Una barriera a 2 step chiamata Barrier, che mantiene le informazioni sul numero di thread utilizzati *nthread*, sul numero di ParticleWorker che hanno terminato la computazione *count* ed espone 2 metodi.   
  Il primo, waitAllDone(), permette di restare in attesa fin quando tutti i thread non abbiamo terminato la computazione della nuova posizione delle particelle, per poi resettare il contatore.   
  Il secondo, inc() viene invocato da ogni ParticleWorker al termine della computazione, incrementando il contatore e risvegliando MainWorker.

MainWorker suddivide il lavoro prendendo il numero di particelle e suddividendolo per il numero di thread, create in base al numero di processori disponibili. ParticleWorker invece, calcola le forze sentite da ciascuna particella assegnatagli, le somma e le applica, mettendo la nuova posizione ottenuta dalla particella in un array temporaneo. Sarà il MainWorker, a ciascuna iterazione, a prendere queste posizioni, aggiornarle nella GUI e trasformandole nelle posizioni delle singole particele.

Questo comportamento permette di far lavorare tutti i ParticleWorker sulle posizioni calcolate allo step precedente, e allo stesso tempo permette di aggiornarle senza dover attendere il calcolo delle forze da parte di tutti i ParticleWorker.

Le coordinate delle particelle sono state calcolate su un piano dalle dimensioni indicate dal **Boundary**, il quale viene visualizzato dalla **ParticleView** all’interno di un pannello dedicato.

## 3. Test e Speedup

Per l’effettuazione dei test sulle performance e il calcolo dello speedup è stato create un package *performance* contenenete due sottopackage *parallel* e *seq*.   
Il primo contiene 2 versioni semplificate dei thread della versione parallela, senza referenza alla view e funzionalità di start/stop mentre nel secondo è presente una classe con la versione sequenziale dell’algoritmo.

I test riportati di seguito sono stati effettuati su un computer con cpu 4 core con hyper threading.

Todo: inserire commenti a riguardo