

# **POLITECNICO DI TORINO**

Laurea di 1° Livello in Ingegneria Gestionale  
Classe L-8 Ingegneria dell'Informazione  
A.A. 2019-2020



## **Simulatore Serie A post Covid-19**

Relatore

Prof. Fulvio Corno

Candidato

Omar Panebianco



# INDICE

1. Proposta di progetto	4
2. Descrizione del problema affrontato	6
3. Data-set utilizzati	7
4. Strutture dati e algoritmi	11
5. Diagramma delle classi principali	17
6. Videate dell'applicazione e video dimostrativo	18
7. Valutazione dei risultati e conclusioni	20

# **1. PROPOSTA DI PROGETTO**

## **1.1 Studente proponente**

s234164 Panebianco Omar

## **1.2 Titolo della proposta**

Simulatore Serie A post Covid-19

## **1.3 Descrizione del problema proposto**

La pandemia di Covid-19 in Italia ha compromesso il regolare svolgimento del campionato di calcio di Serie A. Quest'ultimo è stato sospeso fino a tempo indeterminato dal Consiglio Federale della FIGC in seguito al DPCM del 9 Marzo 2020. Di recente, il Governo Italiano e la FIGC hanno concordato la ripresa del campionato. Nel caso in cui sia istituito un nuovo lockdown il campionato non verrà terminato e la classifica finale della stagione 2019-20 sarà stilata tramite l'utilizzo di un algoritmo.

L'applicazione ha lo scopo di simulare le 124 partite di Serie A non ancora disputate in seguito alla prima sospensione decretando la classifica finale del campionato.

## **1.4 Descrizione della rilevanza gestionale del problema**

Il calcio è un business che vale miliardi di Euro. L'annullamento del campionato comporterebbe per le società la perdita di ingenti somme di denaro. Come sostenuto dagli organizzatori, è necessario decretare una classifica finale - seppur virtuale - per la corretta distribuzione dei premi e per permettere alle società di programmare la stagione successiva.

### **1.5 Descrizione dei data-set per la valutazione**

L'applicazione utilizzerà dei data-set costruiti appositamente. Uno di questi conterrà i dati statistici dettagliati relativi ad ogni squadra di Serie A aggiornati all'ultima giornata disputata che costituiranno le variabili fondamentali per l'algoritmo di simulazione. Tali dati verranno reperiti dal portale WhoScored (<https://it.whoscored.com>).

Un altro data-set conterrà invece il calendario delle 124 partite di Serie A non disputate suddivise per giornata. Il calendario verrà reperito dal sito ufficiale della Lega Serie A (<https://www.legaseriea.it>).

### **1.6 Descrizione preliminare degli algoritmi coinvolti**

L'algoritmo principale dell'applicazione è la simulazione. Sulla base dei dati statistici ricavabili dal primo data-set, l'applicazione effettuerà una simulazione per ognuna delle 124 partite non disputate aggiornando la classifica del campionato di giornata in giornata.

In particolare il risultato di ogni partita verrà decretato considerando le statistiche in casa e in trasferta o assolute di entrambe le squadre tra cui la media dei tiri in porta effettuati a partita e la percentuale di realizzazione degli stessi ma anche la media dei tiri in porta subiti a partita e la percentuale di reti incassate per tiro in porta subito.

### **1.7 Descrizione preliminare delle funzionalità previste per l'applicazione**

All'avvio, l'applicazione mostrerà la classifica della Serie A aggiornata alla 26a giornata. Per avviare la simulazione all'utente verrà data la scelta di utilizzare il *fattore casa* ed un secondo fattore chiamato *fattore punti*. Inoltre all'utente verrà offerta la possibilità di assistere alla simulazione di ogni giornata e la simulazione del turno successivo avverrà su comando dell'utente. Alternativamente l'utente può scegliere di visualizzare direttamente la classifica finale.

## 2. DESCRIZIONE DEL PROBLEMA AFFRONTATO

L'applicazione si propone, mediante l'impiego di algoritmi di simulazione, di prevedere il risultato delle partite del campionato di calcio di Serie A non ancora disputate in seguito alla sospensione del mese di Marzo decretando così la classifica finale del campionato.

Il sistema calcio può essere paragonato sotto molti aspetti ad una vera e propria azienda. Le cifre che ogni anno fluiscono nelle casse delle squadre di Serie A superano complessivamente il miliardo di Euro. Alcune di queste sono addirittura quotate in borsa.

L'annullamento di un campionato comporterebbe quindi per le società coinvolte, per i detentori dei diritti televisivi e per tutte le attività e le persone all'interno del sistema, perdite economiche non indifferenti.

Per la stagione 2019-2020, la necessità di avere a disposizione la classifica finale per la corretta distribuzione dei premi e per permettere alle società di programmare la stagione successiva ha spinto la Lega Serie A a prendere in considerazione l'utilizzo di un algoritmo che potesse simulare le partite non disputate a causa della pandemia di Covid-19 qualora non fosse possibile terminare il campionato sul campo.





La simulazione degli ultimi turni di campionato, infatti, renderebbe possibile l'assegnazione dello scudetto, determinerebbe le squadre qualificate in Champions League e le squadre qualificate in Europa League per la stagione successiva e individuerebbe le squadre costrette alla retrocessione in Serie B.

### 3. DATA-SET UTILIZZATI

L'applicazione, per funzionare in modo appropriato, necessita del supporto di data-set specifici difficilmente reperibili in un formato compatibile con il linguaggio di programmazione utilizzato. Si è resa dunque indispensabile la costruzione apposita di ben quattro data-set.



Il primo data-set utilizzato contiene il calendario delle 124 partite di Serie A non ancora disputate suddivise per giornata. Il secondo contiene invece le principali informazioni sulle squadre del campionato.

Il terzo ed il quarto data-set contengono, infine, i dati statistici dettagliati rispettivamente in casa ed in trasferta relativi ad ogni squadra di Serie A aggiornati all'ultimo turno di campionato disputato prima della sospensione.

TABLES	Field	Type
 partite	id_partita	INT
 squadre	giornata	INT
 statistiche_casa	id_squadra_casa	INT
 statistiche_trasferta	id_squadra_trasferta	INT
	gol_squadra_casa	INT
	gol_squadra_trasferta	INT

Il data-set denominato *partite* è costituito da 6 campi:





- id\_partita                      identificativo della partita
- giornata                        numero del turno di gioco
- id\_squadra\_casa                identificativo della squadra in casa
- id\_squadra\_trasferta           identificativo della squadra in trasferta
- gol\_squadra\_casa                numero reti della squadra in casa
- gol\_squadra\_trasferta           numero reti della squadra in trasferta

TABLES	Field	Type
 partite	id_squadra	INT
 <b>squadre</b>	nome_squadra	VARCHAR
 statistiche_casa	pg_casa	INT
 statistiche_trasferta	v_casa	INT
	n_casa	INT
	p_casa	INT
	gf_casa	INT
	gs_casa	INT
	pg_trasferta	INT
	v_trasferta	INT
	n_trasferta	INT
	p_trasferta	INT
	gf_trasferta	INT
	gs_trasferta	INT

Il data-set denominato *squadre* è costituito da 14 campi:

- id\_squadra                      identificativo della squadra
- nome\_squadra                  nome della squadra
- pg\_casa                          partite giocate in casa
- v\_casa                           vittorie in casa
- n\_casa                           pareggi in casa
- p\_casa                           sconfitte in casa
- gf\_casa                          reti fatte in casa
- gs\_casa                          reti subite in casa
- pg\_trasferta                    partite giocate in trasferta
- v\_trasferta                      vittorie in trasferta
- n\_trasferta                      pareggi in trasferta
- p\_trasferta                      sconfitte in trasferta
- gf\_trasferta                    reti fatte in trasferta
- gs\_trasferta                    reti subite in trasferta







TABLES	Field	Type
 partite	id_squadra	INT
 squadre	tiri_totali_fatti	DOUBLE
 statistiche_casa	tiri_azione_fatti	DOUBLE
 statistiche_trasferta	tiri_piazzato_fatti	DOUBLE
	tiri_rigore_fatti	DOUBLE
	gol_azione_realizzati	DOUBLE
	gol_piazzato_realizzati	DOUBLE
	gol_rigore_realizzati	DOUBLE
	tiri_totali_concessi	DOUBLE
	tiri_azione_concessi	DOUBLE
	tiri_piazzato_concessi	DOUBLE
	tiri_rigore_concessi	DOUBLE
	gol_azione_subiti	DOUBLE
	gol_piazzato_subiti	DOUBLE
	gol_rigore_subiti	DOUBLE
	prob_autogol_a_favore	DOUBLE
	prob_autogol_contro	DOUBLE
	falli_fatti	DOUBLE
	falli_subiti	DOUBLE
	prob_espulsione	DOUBLE

Il data-set denominato *statistiche\_casa* è costituito da 20 campi:

- id\_squadra                      identificativo della squadra
- tiri\_totali\_fatti                media tiri totali fatti
- tiri\_azione\_fatti                media tiri su azione fatti
- tiri\_piazzato\_fatti              media tiri su calcio piazzato fatti
- tiri\_rigore\_fatti                media tiri su calcio di rigore fatti
- gol\_azione\_realizzati          media gol su azione fatti
- gol\_piazzato\_fatti              media gol su calcio piazzato fatti
- gol\_rigore\_fatti                media gol su calcio di rigore
- tiri\_totali\_concessi            media tiri totali subiti
- tiri\_azione\_concessi          media tiri su azione subiti
- tiri\_piazzato\_concessi        media tiri su calcio piazzato subiti
- tiri\_rigore\_concessi          media tiri su calcio di rigore subiti
- gol\_azione\_subiti              media gol su azione subiti
- gol\_piazzato\_subiti            media gol su calcio piazzato subiti
- gol\_rigore\_subiti              media gol su calcio di rigore subiti

- `prob_autogol_a_favore`                      probabilità di un autogol a favore
- `prob_autogol_contro`                      probabilità di un autogol a sfavore
- `falli_fatti`                                      media falli fatti
- `falli_subiti`                                    media falli subiti
- `prob_espulsione`                            probabilità di espulsione per fallo fatto

TABLES		Field	Type
	<code>partite</code>	<code>id_squadra</code>	INT
	<code>squadre</code>	<code>tiri_totali_fatti</code>	DOUBLE
	<code>statistiche_casa</code>	<code>tiri_azione_fatti</code>	DOUBLE
	<code>statistiche_trasferta</code>	<code>tiri_piazzato_fatti</code>	DOUBLE
		<code>tiri_rigore_fatti</code>	DOUBLE
		<code>gol_azione_realizzati</code>	DOUBLE
		<code>gol_piazzato_realizzati</code>	DOUBLE
		<code>gol_rigore_realizzati</code>	DOUBLE
		<code>tiri_totali_concessi</code>	DOUBLE
		<code>tiri_azione_concessi</code>	DOUBLE
		<code>tiri_piazzato_concessi</code>	DOUBLE
		<code>tiri_rigore_concessi</code>	DOUBLE
		<code>gol_azione_subiti</code>	DOUBLE
		<code>gol_piazzato_subiti</code>	DOUBLE
		<code>gol_rigore_subiti</code>	DOUBLE
		<code>prob_autogol_a_favore</code>	DOUBLE
		<code>prob_autogol_contro</code>	DOUBLE
		<code>falli_fatti</code>	DOUBLE
		<code>falli_subiti</code>	DOUBLE
		<code>prob_espulsione</code>	DOUBLE

Il data-set denominato *statistiche\_trasferta* è costituito dagli stessi 20 campi del data-set *statistiche\_casa* ma i valori concernano i dati statistici in trasferta di ogni squadra.

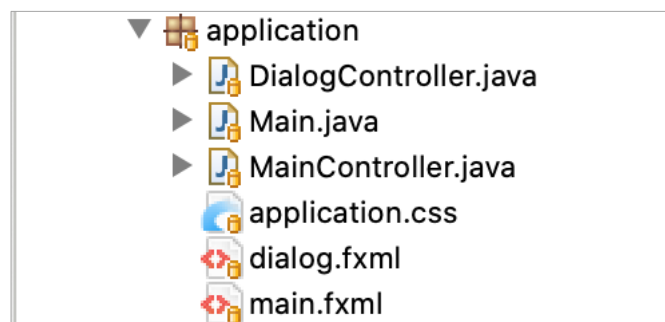
## 4. STRUTTURE DATI E ALGORITMI

### 4.1 Descrizione delle strutture dati utilizzate

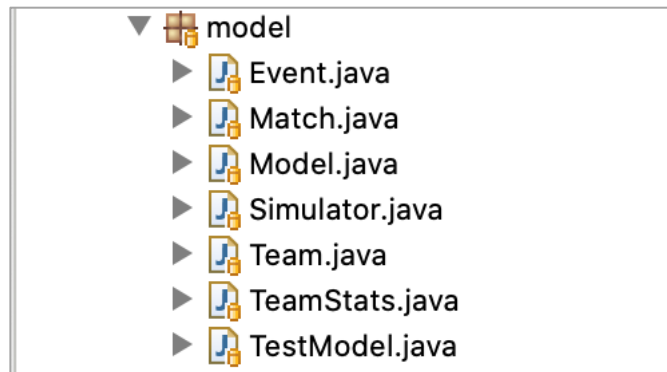
L'applicazione è realizzata mediante l'utilizzo del linguaggio di programmazione Java ed in particolare mediante l'utilizzo degli applicativi JavaFX. Sono stati altresì implementati il pattern MVC (Model View Controller), per garantire una corretta divisione delle operazioni all'interno del software delegando al Model la logica applicativa mentre il Controller ha il compito di gestire l'interazione con l'utente e mostrare a video i risultati, e il pattern DAO (Data Access Object) per la gestione delle interazioni con il database.

Strutturalmente l'applicazione è suddivisa nei seguenti tre package:

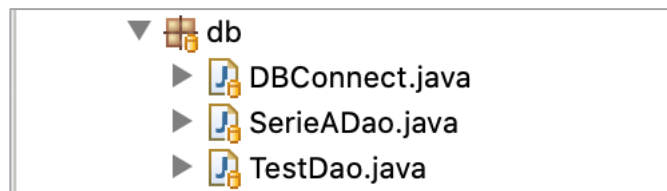
- Il package *application* contiene tutte le classi e i file che si occupano di realizzare l'interfaccia grafica e di interagire con l'utente ricevendo i parametri in input da inviare al model e restituendo i risultati.



- Il package *model* è il cuore dell'applicazione e contiene tutte le classi necessarie all'elaborazione dei dati. Il package contiene anche una classe dotata di *main* per testare gli algoritmi.



- Il package *db* contiene invece le classi utili alla gestione delle connessioni al database e all'estrapolazione dei dati necessari per l'implementazione del model.



È presente anche un quarto package denominato *img* che contiene tutte le immagini utilizzate dall'interfaccia grafica in formato *.png*.

## 4.2 Descrizione degli algoritmi utilizzati

Ogni qualvolta si avvia una simulazione tramite apposito pulsante viene richiamato il metodo `startSimulation()` della classe *Model* che aggiorna i parametri di simulazione personalizzabili in input – qualora siano stati modificati quelli impostati di default – e verifica quale sia il tipo di simulazione scelto dall'utente.

La scelta del tipo di simulazione determina il numero di ripetizioni dell'algoritmo di simulazione principale situato nella classe *Simulator*. Per ogni partita da simulare, infatti, il simulatore viene inizializzato, aggiornato con i dati delle squadre coinvolte e successivamente avviato.

I risultati di ogni singola partita, suddivisi per giornata, vengono poi memorizzati in una mappa denominata `simulatedDays` all'interno della classe *Model*.

Il simulatore utilizza una `PriorityQueue` per la gestione degli eventi di una partita inizializzata in modo tale da estrarre gli stessi eventi in modo casuale. Si possono verificare dieci tipi di evento diversi per match:

```
public enum EventType {  
    HOME_OPEN_PLAY,  
    AWAY_OPEN_PLAY,  
    HOME_FREE_KICK,  
    AWAY_FREE_KICK,  
    HOME_PENALTY,  
    AWAY_PENALTY,  
    HOME_OWN_GOAL,  
    AWAY_OWN_GOAL,  
    HOME_FOUL,  
    AWAY_FOUL  
}
```

L'inizializzazione del simulatore è molto complessa. Oltre a resettare tutte le variabili di output, il metodo `init(Match match)` è responsabile del corretto inserimento di tutti gli eventi nella *queue* e della loro probabilità di realizzazione.

Il numero di occasioni a favore della squadra in casa viene determinato effettuando una media pesata del numero di tiri fatti a partita dalla squadra in casa e del numero di tiri subiti a partita dalla squadra in trasferta. Il numero di occasioni a favore della squadra in trasferta segue la stessa logica. Analogamente è anche il calcolo del numero di falli fatti e subiti da entrambe le squadre.

```
homeShots = (match.getHomeTeam().getHomeStats().getTotalMadeShots()  
            + match.getAwayTeam().getAwayStats().getTotalConcededShots()) / 2;  
awayShots = (match.getAwayTeam().getAwayStats().getTotalMadeShots()  
            + match.getHomeTeam().getHomeStats().getTotalConcededShots()) / 2;  
homeFouls = (match.getHomeTeam().getHomeStats().getCommittedFouls()  
            + match.getAwayTeam().getAwayStats().getTakenFouls()) / 2;  
awayFouls = (match.getAwayTeam().getAwayStats().getCommittedFouls()  
            + match.getHomeTeam().getHomeStats().getTakenFouls()) / 2;
```

Il metodo inserisce quindi tanti eventi in coda fino a raggiungere il numero medio pesato di occasioni per squadra appena calcolati. Viene inoltre calcolata la probabilità di realizzazione di ogni singolo evento: ad esempio il *successRate* di un *HOME\_OPEN\_PLAY*, ovvero la probabilità che la squadra di casa realizzi una rete in seguito ad un tiro su azione, è determinato effettuando una media pesata tra il numero di reti fatte su azione per tiro su azione dalla squadra in casa e il numero di reti subite su azione per tiro su azione dalla squadra in trasferta.

```
for(int i = 0; i < (homeShots * (1 + pointsFactor)); i++) {
    double homeOpenPlayMadeShots = match.getHomeTeam().getHomeStats().getOpenPlayMadeShots();
    double awayOpenPlayConcededShots = match.getAwayTeam().getAwayStats().getOpenPlayConcededShots();
    double mixedOpenPlayShots = (homeOpenPlayMadeShots + awayOpenPlayConcededShots) / 2;
    double openPlayShotProbability = mixedOpenPlayShots / homeShots;

    double homeFreeKickMadeShots = match.getHomeTeam().getHomeStats().getFreeKickMadeShots();
    double awayFreeKickConcededShots = match.getAwayTeam().getAwayStats().getFreeKickConcededShots();
    double mixedFreeKickShots = (homeFreeKickMadeShots + awayFreeKickConcededShots) / 2;
    double openFreeKickProbability = mixedFreeKickShots / homeShots;

    double homePenaltyMadeShots = match.getHomeTeam().getHomeStats().getPenaltyMadeShots();
    double awayPenaltyConcededShots = match.getAwayTeam().getAwayStats().getPenaltyConcededShots();
    double mixedPenaltyShots = (homePenaltyMadeShots + awayPenaltyConcededShots) / 2;

    randomDouble = random.nextDouble();

    if(randomDouble < openPlayShotProbability) {
        double homeMadeGoalProbability = match.getHomeTeam().getHomeStats().getOpenPlayMadeGoals() / mixedOpenPlayShots;
        double awayConcededGoalProbability = match.getAwayTeam().getAwayStats().getOpenPlayConcededGoals() / mixedOpenPlayShots;
        double successRate = (homeMadeGoalProbability + awayConcededGoalProbability) / 2;

        queue.add(new Event(EventType.HOME_OPEN_PLAY, (successRate * (1 + pointsFactor))));
    } else if(randomDouble < (openPlayShotProbability + openFreeKickProbability)) {
        double homeMadeGoalProbability = match.getHomeTeam().getHomeStats().getFreeKickMadeGoals() / mixedFreeKickShots;
        double awayConcededGoalProbability = match.getAwayTeam().getAwayStats().getFreeKickConcededGoals() / mixedFreeKickShots;
        double successRate = (homeMadeGoalProbability + awayConcededGoalProbability) / 2;

        queue.add(new Event(EventType.HOME_FREE_KICK, (successRate * (1 + pointsFactor))));
    } else {
        double homeMadeGoalProbability = match.getHomeTeam().getHomeStats().getPenaltyMadeGoals() / mixedPenaltyShots;
        double awayConcededGoalProbability = match.getAwayTeam().getAwayStats().getPenaltyConcededGoals() / mixedPenaltyShots;
        double successRate = (homeMadeGoalProbability + awayConcededGoalProbability) / 2;

        queue.add(new Event(EventType.HOME_PENALTY, successRate));
    }
}
```

Analogamente, vengono inseriti tanti eventi in coda fino a raggiungere il numero medio pesato di falli a favore per squadra. In questo caso il *successRate* rappresenta la probabilità che l'evento di un fallo di gioco provochi un'espulsione. Viene inoltre inserito in coda un evento di tipo *OWN\_GOAL* – autogol – per squadra.

```

double homeConcededOwnGoalProbability = match.getHomeTeam().getHomeStats().getConcededOwnGoalProbability();
double awayMadeOwnGoalProbability = match.getAwayTeam().getAwayStats().getMadeOwnGoalProbability();
double homeOwnGoalsuccessRate = (homeConcededOwnGoalProbability + awayMadeOwnGoalProbability) / 2;

queue.add(new Event(EventType.HOME_OWN_GOAL, (homeOwnGoalsuccessRate * (1 - pointsFactor))));

double awayConcededOwnGoalProbability = match.getAwayTeam().getAwayStats().getConcededOwnGoalProbability();
double homeMadeOwnGoalProbability = match.getHomeTeam().getHomeStats().getMadeOwnGoalProbability();
double awayOwnGoalsuccessRate = (awayConcededOwnGoalProbability + homeMadeOwnGoalProbability) / 2;

queue.add(new Event(EventType.AWAY_OWN_GOAL, (awayOwnGoalsuccessRate * (1 + pointsFactor))));

for(int i = 0; i < (homeFouls * (1 - pointsFactor)); i++) {
    double successRate = match.getHomeTeam().getHomeStats().getRedCardProbability();
    queue.add(new Event(EventType.HOME_FOUL, (successRate * (1 - pointsFactor))));
}

for(int i = 0; i < (awayFouls * (1 + pointsFactor)); i++) {
    double successRate = match.getAwayTeam().getAwayStats().getRedCardProbability();
    queue.add(new Event(EventType.AWAY_FOUL, (successRate * (1 + pointsFactor))));
}

```

I moltiplicatori *homeFactor* e *pointsFactor* incidono sul numero e sulla percentuale di realizzazione delle occasioni da rete e dei falli commessi e subiti di entrambe le squadre coinvolte nella simulazione. Maggiore è il loro valore, maggiore è l'impatto sui dati coinvolti.

Il *redCardMultiplier*, invece, determina un aumento del solo numero di occasioni da rete per la squadra che ne beneficia e una diminuzione dello stesso per la squadra che la subisce in seguito al verificarsi di un espulsione.

Il metodo `run()` avvia la simulazione estraendo uno alla volta i vari eventi dalla coda e li gestisce determinando in modo casuale la realizzazione degli stessi in base al già discusso *successRate*. Il metodo aggiorna quindi i parametri della simulazione.

```

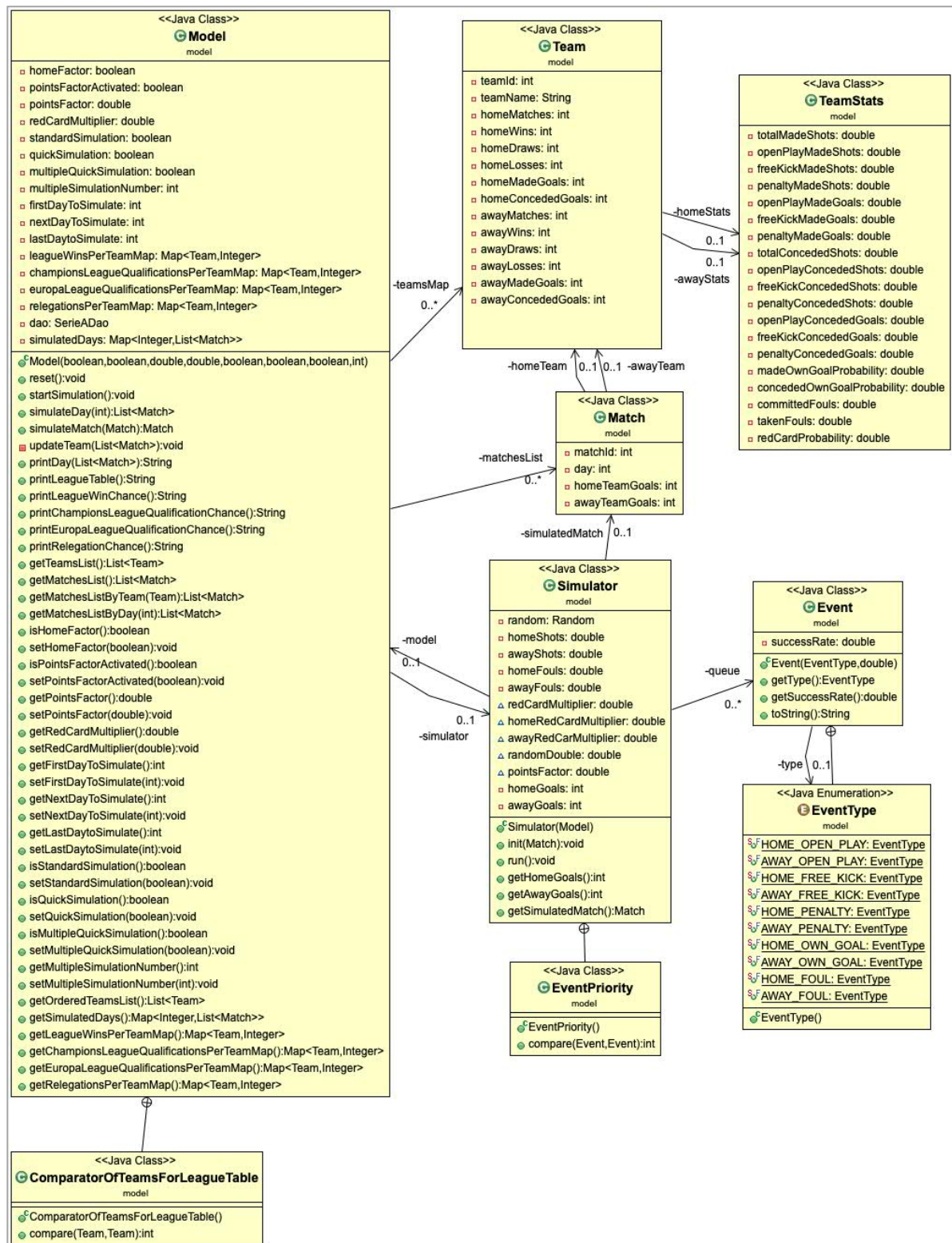
public void run() {
    while(!queue.isEmpty()) {
        Event event = queue.poll();
        randomDouble = random.nextDouble();

        switch(event.getType()) {
            case HOME_OPEN_PLAY:
                if(randomDouble < (event.getSuccessRate() * homeRedCardMultiplier))
                    homeGoals ++;
                break;
            case AWAY_OPEN_PLAY:
                if(randomDouble < (event.getSuccessRate() * awayRedCarMultiplier))
                    awayGoals ++;
                break;
            case HOME_FREE_KICK:
                if(randomDouble < (event.getSuccessRate() * homeRedCardMultiplier))
                    homeGoals ++;
                break;
            case AWAY_FREE_KICK:
                if(randomDouble < (event.getSuccessRate() * awayRedCarMultiplier))
                    awayGoals ++;
                break;
            case HOME_PENALTY:
                if(randomDouble < (event.getSuccessRate() * homeRedCardMultiplier))
                    homeGoals ++;
                break;
            case AWAY_PENALTY:
                if(randomDouble < (event.getSuccessRate() * awayRedCarMultiplier))
                    awayGoals ++;
                break;
            case HOME_OWN_GOAL:
                if(randomDouble < (event.getSuccessRate() * awayRedCarMultiplier))
                    awayGoals ++;
                break;
            case AWAY_OWN_GOAL:
                if(randomDouble < (event.getSuccessRate() * homeRedCardMultiplier))
                    homeGoals ++;
                break;
            case HOME_FOUL:
                if(randomDouble < event.getSuccessRate()) {
                    awayRedCarMultiplier += redCardMultiplier;
                    homeRedCardMultiplier -= redCardMultiplier;
                }
                break;
            case AWAY_FOUL:
                if(randomDouble < event.getSuccessRate()) {
                    homeRedCardMultiplier += redCardMultiplier;
                    awayRedCarMultiplier -= redCardMultiplier;
                }
                break;
        }
    }
    simulatedMatch.setHomeTeamGoals(homeGoals);
    simulatedMatch.setAwayTeamGoals(awayGoals);
}

```



## 5. DIAGRAMMA DELLE CLASSI PRINCIPALI



## 6. VIDEATE DELL'APPLICAZIONE E DIMOSTRAZIONE

The screenshot shows the 'Serie A Simulator' application window. On the left is the 'Settings' panel, and on the right is the 'League Table'.

**Settings Panel:**

- Home Factor:** ☒
- Points Factor:** ☒ (Slider: 0.1 to 0.9, set at 0.5)
- Red Card Multiplier:** (Slider: 0.05 to 0.5, set at 0.25)
- Simulation Type:**
  - ☒ Standard Simulation
  - ☐ Quick Simulation
  - ☐ Multiple Quick Simulation
- Number of Simulations:** 100
- Start Simulation:** Button
- n.b.:** simulator might take much time by using a big number of simulations.

**League Table:**

POS	TEAM	PTS	PM	W	D	L	GF	GA	HPM	HW	HD	HL	HGF	HGA	APM	AW	AD	AL	AGF	AGA
1	JUVENTUS	63	26	20	3	3	50	24	13	12	1	0	31	10	13	8	2	3	19	14
2	LAZIO	62	26	19	5	2	60	23	14	11	3	0	39	10	12	8	2	2	21	13
3	INTER	54	25	16	6	3	49	24	12	7	4	1	23	10	13	9	2	2	26	14
4	ATALANTA	48	25	14	6	5	70	34	12	6	2	4	33	19	13	8	4	1	37	15
5	ROMA	45	26	13	6	7	51	35	13	6	3	4	26	19	13	7	3	3	25	16
6	NAPOLI	39	26	11	6	9	41	36	13	5	2	6	17	18	13	6	4	3	24	18
7	MILAN	36	26	10	6	10	28	34	13	4	5	4	13	15	13	6	1	6	15	19
8	VERONA	35	25	9	8	8	29	26	12	6	3	3	17	12	13	3	5	5	12	14
9	PARMA	35	25	10	5	10	32	31	13	6	1	6	17	12	12	4	4	4	15	19
10	BOLOGNA	34	26	9	7	10	38	42	13	4	5	4	18	20	13	5	2	6	20	22
11	SASSUOLO	32	25	9	5	11	41	39	13	6	1	6	28	22	12	3	4	5	13	17
12	CAGLIARI	32	25	8	8	9	41	40	13	5	2	6	25	23	12	3	6	3	16	17
13	FIorentina	30	26	7	9	10	32	36	13	3	5	5	13	17	13	4	4	5	19	19
14	UDINESE	28	26	7	7	12	21	37	13	5	4	4	10	12	13	2	3	8	11	25
15	TORINO	27	25	8	3	14	28	45	12	4	2	6	11	22	13	4	1	8	17	23
16	SAMPDORIA	26	25	7	5	13	28	44	14	4	4	6	16	22	11	3	1	7	12	22
17	GENOA	25	26	6	7	13	31	47	12	4	1	7	14	18	14	2	6	6	17	29
18	LECCE	25	26	6	7	13	34	56	13	2	5	6	19	26	13	4	2	7	15	30
19	SPAL	18	26	5	3	18	20	44	12	2	2	8	11	20	14	3	1	10	9	24
20	BRESCIA	16	26	4	4	18	22	49	13	1	3	9	13	25	13	3	1	9	9	24

**Next Day:** DAY 25

- TORINO - PARMA
- VERONA - CAGLIARI
- ATALANTA - SASSUOLO
- INTER - SAMPDORIA

Videata iniziale dell'applicazione

The screenshot shows the 'Serie A Simulator' application window during a simulation. The 'Settings' panel is on the left, and the 'League Table' and 'Simulated Day' are on the right.

**Settings Panel:**

- Home Factor:** ☒
- Points Factor:** ☒ (Slider: 0.1 to 0.9, set at 0.5)
- Red Card Multiplier:** (Slider: 0.05 to 0.5, set at 0.25)
- Simulation Type:**
  - ☒ Standard Simulation
  - ☐ Quick Simulation
  - ☐ Multiple Quick Simulation
- Number of Simulations:** 100
- Start Simulation:** Button
- Simulate Next Day:** Button
- End Simulation:** Button
- n.b.:** simulator might take much time by using a big number of simulations.

**League Table:**

POS	TEAM	PTS	PM	W	D	L	GF	GA	HPM	HW	HD	HL	HGF	HGA	APM	AW	AD	AL	AGF	AGA
1	JUVENTUS	92	37	29	5	3	78	33	18	16	2	0	46	16	19	13	3	3	32	17
2	LAZIO	90	37	28	6	3	98	33	19	15	4	0	58	13	18	13	2	3	40	20
3	INTER	88	37	27	7	3	85	29	19	14	4	1	48	13	18	13	3	2	37	16
4	ATALANTA	68	37	20	8	9	94	51	18	9	3	6	47	28	19	11	5	3	47	23
5	ROMA	66	37	19	9	9	79	48	19	10	4	5	43	24	18	9	5	4	36	24
6	PARMA	57	37	17	6	14	46	42	19	9	2	8	23	18	18	8	4	6	23	24
7	NAPOLI	53	37	15	8	14	56	53	18	7	3	8	23	26	19	8	5	6	33	27
8	VERONA	50	37	13	11	13	41	44	19	8	4	7	22	22	18	5	7	6	19	22
9	MILAN	50	37	14	8	15	48	56	18	6	5	7	21	25	19	8	3	8	27	31
10	SASSUOLO	46	37	13	7	17	53	57	18	7	3	8	36	31	19	6	4	9	17	26
11	UDINESE	45	37	12	9	16	44	48	19	9	4	6	26	17	18	3	5	10	18	31
12	BOLOGNA	44	37	12	8	17	53	62	18	6	5	7	24	27	19	6	3	10	29	35
13	CAGLIARI	42	37	10	12	15	55	64	19	5	3	11	30	39	18	5	9	4	25	25
14	TORINO	41	37	12	5	20	45	73	19	5	4	10	18	36	18	7	1	10	27	37
15	FIorentina	37	37	8	13	16	42	57	19	3	8	8	19	26	18	5	5	8	23	31
16	SPAL	35	37	10	5	22	36	60	18	3	4	11	19	32	19	7	1	11	17	28
17	LECCE	34	37	8	10	19	48	82	18	3	6	9	24	36	19	5	4	10	24	46
18	BRESCIA	33	37	9	6	22	40	66	18	4	4	10	22	31	19	5	2	12	18	35
19	SAMPDORIA	32	37	9	5	23	38	83	19	6	4	9	23	33	18	3	1	14	15	50
20	GENOA	29	37	7	8	22	38	76	18	4	2	12	18	34	19	3	6	10	20	42

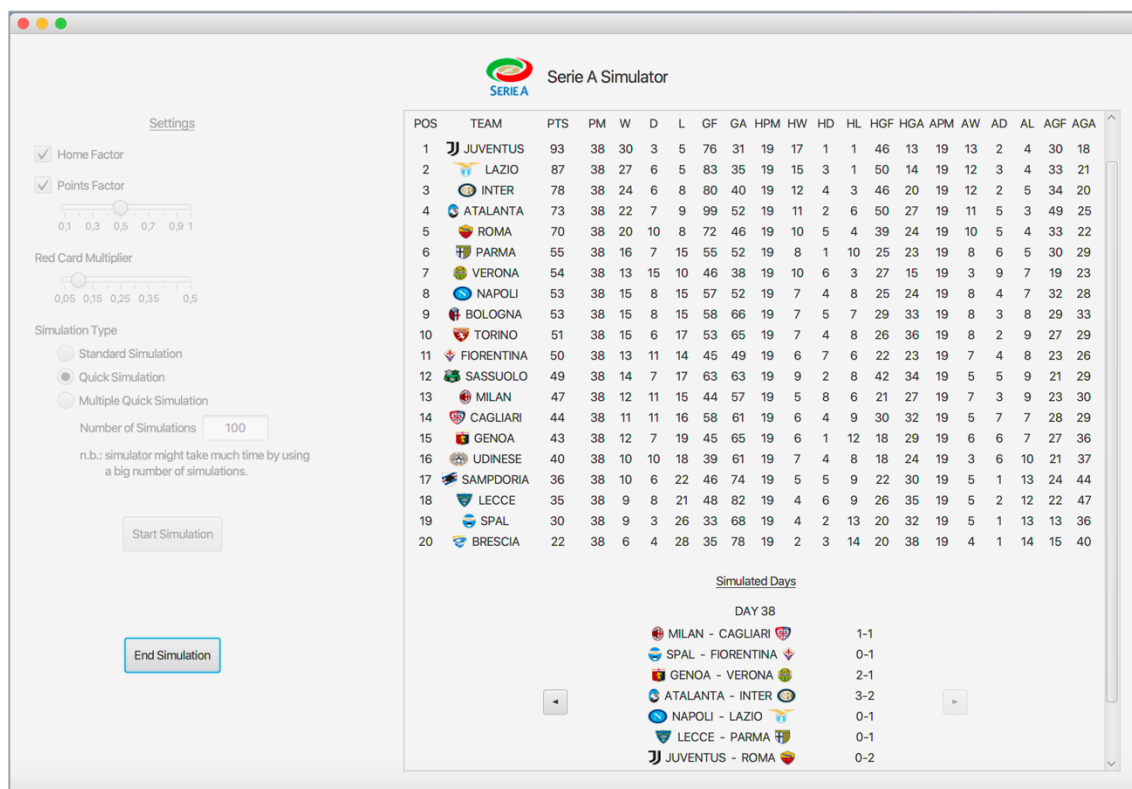
**Simulated Day:** DAY 37

- PARMA - ATALANTA 1-2
- FIorentina - BOLOGNA 1-1
- LAZIO - BRESCIA 3-0
- SASSUOLO - GENOA 2-0
- CAGLIARI - JUVENTUS 0-3
- UDINESE - LECCE 4-0
- SAMPDORIA - MILAN 2-1

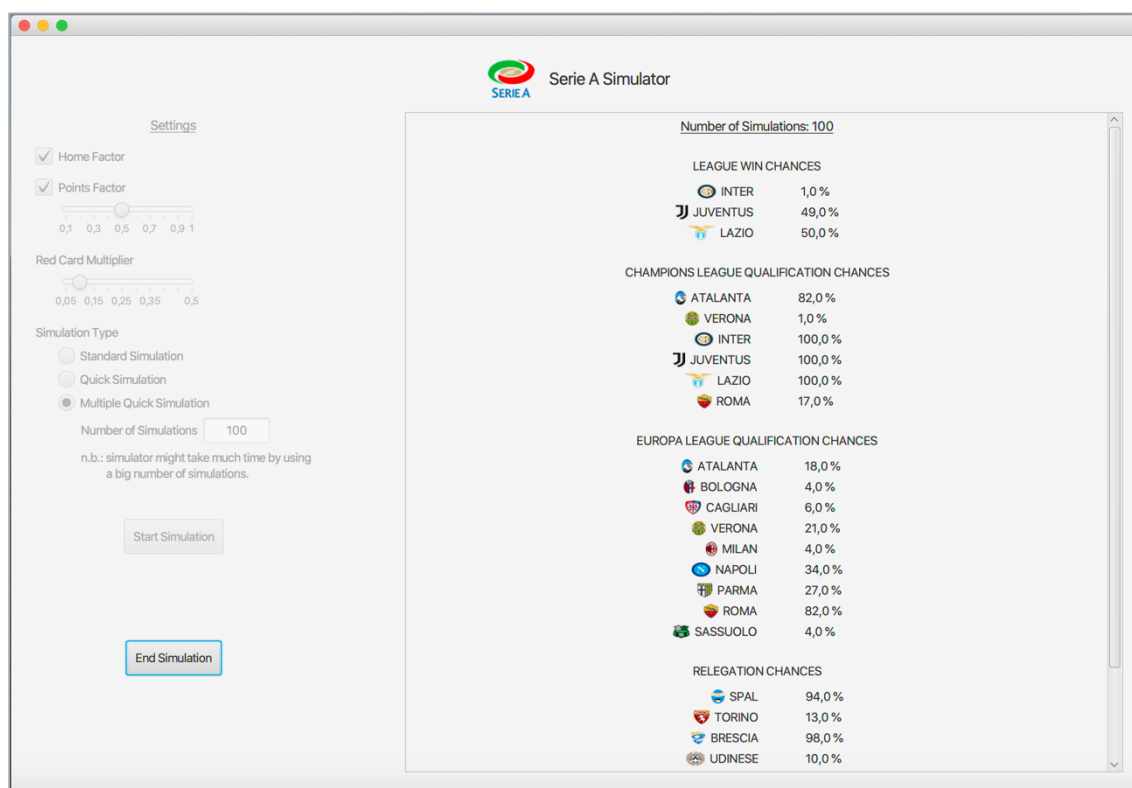
**Next Day:** DAY 38

- MILAN - CAGLIARI
- SPAL - FIorentina
- GENOA - VERONA
- ATALANTA - INTER
- NAPOLI - LAZIO
- LECCE - PARMA
- JUVENTUS - ROMA

Videata dell'applicazione durante una Standard Simulation



Videata dell'applicazione al termine di una Quick Simulation



Videata dell'applicazione al termine di una Multiple Quick Simulation

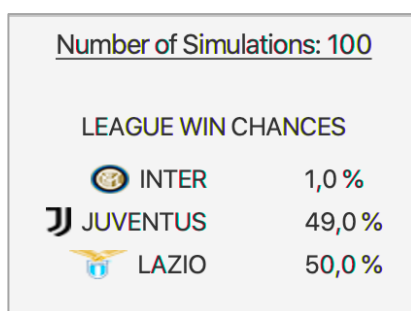
Video dimostrativo su Youtube: <https://youtu.be/0ihUSEDus1w>

## 7. VALUTAZIONE DEI RISULTATI E CONCLUSIONI

La simulazione veloce multipla può fornire un'idea sulla bontà algoritmica dell'applicazione. Appare chiaro che un algoritmo di simulazione per un campionato di calcio non potrebbe mai garantire la certezza assoluta del risultato così come è indiscutibile il risultato sul campo di giuoco.

Ciò viene evidenziato particolarmente quando si confronta il cammino di due squadre molto vicine in classifica. Sul campo sono i dettagli a fare la differenza e questi dettagli sono variabili casuali difficilmente computerizzabili.

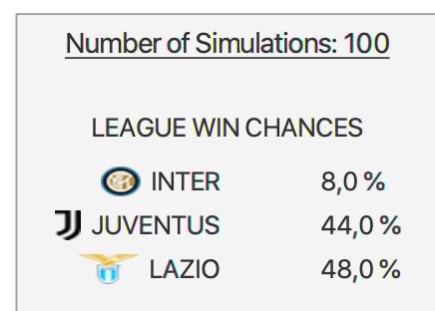
Si rende dunque fondamentale sfruttare appieno i dati a disposizione sulle squadre – tra l'altro, di non semplice reperibilità – e riuscire a dare un peso specifico ai risultati ottenuti. In base a quest'esperienza, si può affermare che, una volta scelti i parametri di simulazione, quest'ultima è tanto più accurata quanto è maggiore il numero di ripetizioni.



*homeFactor: attivo*

*pointsFactor: 0.5*




*redCardMultiplier: 0.1*



*homeFactor: attivo*

*pointsFactor: 0*




*redCardMultiplier: 0.1*

Number of Simulations: 100		
LEAGUE WIN CHANCES		
 INTER		7,0 %
 JUVENTUS		40,0 %
 LAZIO		53,0 %

*homeFactor: attivo*

*pointsFactor: 0.2*

*redCardMultiplier: 0.1*

Number of Simulations: 100		
LEAGUE WIN CHANCES		
 INTER		3,0 %
 JUVENTUS		48,0 %
 LAZIO		49,0 %

*homeFactor: attivo*

*pointsFactor: 0.5*

*redCardMultiplier: 0.25*

Effettuando cento simulazioni differenti si può notare come, anche variando i parametri di simulazione modificabili dall'utente nel tab *Settings*, i risultati ottenuti sono del tutto paragonabili. Il che può dimostrare una discreta affidabilità dell'applicazione.



Quest'opera è distribuita con Licenza [Creative Commons Attribuzione - Non commerciale - Condividi allo stesso modo 4.0 Internazionale](https://creativecommons.org/licenses/by-nc-sa/4.0/).