POLITECNICO DI TORINO

Laurea di 1° Livello in Ingegneria Gestionale
Classe L-8 Ingegneria dell'Informazione

Simulatore Playoff NBA

Relatore
Prof. Fulvio Corno

Candidato
Ruben Berteletti

Anno Accademico
2018/2019

INDICE

Proposta di progetto 4

Descrizione dettagliate del problema affrontato 6

Descrizione del data-set utilizzato.....7

Descrizione ad alto livello delle strutture dati e degli algoritmi utilizzati 9

Diagramma delle classi principali 16

Videate dell'applicazione con risultati ottenuti e link al video dimostrativo del software 17

Valutazione dei risultati e conclusioni..... 20

Proposta di progetto

STUDENTE PROPONENTE

Berteletti Ruben (s236819)

TITOLO DELLA PROPOSTA

Simulatore playoff NBA

DESCRIZIONE DEL PROBLEMA PROPOSTO

L'applicazione si propone di permettere all'utente di scoprire il vincitore della stagione NBA. Partendo dai playoff verranno selezionate le squadre partecipanti e tramite l'appoggio sui dati reali della stagione regolare sulle prestazioni dei giocatori, verrà simulata partita per partita e serie per serie fino a decretare il campione della lega NBA, individuando anche il miglior giocatore delle finali.

DESCRIZIONE DELLA RILEVANZA GESTIONALE DEL PROBLEMA

L'attenzione per l'utilità gestionale è vista in modo particolare per l'utente "Team":

La squadra è a tutti gli effetti un'azienda che con il raggiungimento di risultati positivi, avrà maggiori introiti; può quindi essere utile avere tramite la simulazione, una stima realistica sul risultato finale della propria squadra e quindi valutare in anticipo gli investimenti derivanti dalla posizione: ogni step in più raggiunto dal team porta nelle casse della società diversi milioni di dollari tra premi, sponsorizzazioni e visibilità.

Di fatto si tratta di un'operazione di benchmarking del team basato sui risultati ottenuti nella stagione regolare, sulla quale lo staff può agire reagendo in anticipo attraverso delle migliori strategie.

L'applicazione può essere anche sfruttata dall'utente comune per il settore scommesse, fenomeno largamente diffuso negli ultimi anni.

DESCRIZIONE DEI DATA-SET PER LA VALUTAZIONE

L'applicazione fa uso di due dataset:

- Un primo dataset disponibile all'indirizzo https://www.kaggle.com/schmadam97/nba-regular-season-stats-20182019, in cui sono presenti tutte le statistiche di tutti i giocatori del circuito NBA nella stagione 2018/2019 (convertito in formato sql);

- Un dataset creato “ad hoc” e caricato su GitHub all’indirizzo https://github.com/rubenIng93/nba_teams_sql/blob/master/teams.sql, che riporta informazioni riguardo la conference di appartenenza di ogni squadra.

DESCRIZIONE PRELIMINARE DEGLI ALGORITMI COINVOLTI

L'algoritmo principale è una simulazione: una volta individuati i team che hanno raggiunto i playoff, verranno inseriti dall'utente nell'interfaccia in maniera da individuare tutti gli scontri, costituiti da una serie di partite al meglio delle sette.

Dato uno scontro viene avviata la simulazione su questo; contribuiscono al punteggio finale tutte le statistiche presenti nel data-set, come la percentuale di realizzazione canestri o le stoppate e gli assist.

L’applicazione comprende anche algoritmi di randomizzazione che svolgono la funzione da un lato di inserimento nella coda degli eventi della possibilità di infortunio, e dall’altro moderano le percentuali realizzative dei giocatori.

DESCRIZIONE PRELIMINARE DELLE FUNZIONALITÀ PREVISTE PER L’APPLICAZIONE SOFTWARE

L'applicazione inizialmente permette all'utente di inserire nel sistema i team che hanno raggiunto i playoff attraverso gli apposito combo box, delle query apposite al database permettono di dividere le squadre per conference.
Se gli algoritmi non rilevano errori in input, l’utente può avviare la simulazione di ogni conference scegliendo tra due modalità:

- Modalità “Serie per serie” se l’utente desidera un’esperienza più interattiva: viene fornito in output il risultato della prima serie ancora da giocare ogni volta che si preme il bottone di simulazione;
- Modalità “Simulazione veloce”: spuntando l’apposita casella, vengono simulate tutte le serie della conference fino a decretarne il vincitore.

Dopo aver simulato una serie, è possibile accedere alle rispettive statistiche cliccando sulla squadra vincente della serie.

Attraverso il pulsante “GO TO FINALS” si accede all’interfaccia delle finali assolute ed avviando l’ultima simulazione viene decretato il team campione ed eletto il miglior giocatore.

Descrizione dettagliata del problema affrontato

Nel concreto le squadre delle discipline sportive sono di fatto delle aziende, che se competono nelle massime serie, arrivano a fatturare cifre decisamente corpose: per dare alcuni esempi, in ambito calcistico la “Juventus” nell’anno 2018 è riuscita ad ottenere un ricavo di più di 500 milioni di euro (fonte: calcioefinanza.it); il team “Ferrari” che gareggia in Formula 1 ha ricavato nello stesso anno 205 milioni di euro (fonte: foxsport.it); spostandosi oltre oceano ed esaminando il circuito NBA, squadre come “Golden State Warriors” o “Los Angeles Lakers” hanno fatturato circa 400 milioni di dollari ed il loro valore complessivo supera abbondantemente i 3 miliardi (fonte: Forbes).

È allora chiaro che sono necessari degli strumenti in grado di prevedere con una certa attendibilità i risultati di fine stagione, in modo che le società possano avviare strategie preventive in modo da cercare di raggiungere gli obiettivi stagionali.

Ponendo il focus sul circuito NBA, l’applicazione si propone per raggiungere proprio questo scopo: Attraverso algoritmi specifici che analizzano come input le statistiche della stagione regolare dei giocatori, viene simulata partita per partita e per ognuna di queste vengono memorizzate le performance che possono essere consultate in qualsiasi momento dall’utente, in modo da permettere di avere una previsione sul risultato finale sia in termini di società, avendo un riscontro completo sulla posizione conquistata, sia in termini individuali per ogni giocatore .

Parte fondamentale del software è l’inserimento della componente aleatoria che garantisce risultati del tutto verosimili alla realtà, in quanto realisticamente parlando, non è sempre vero che il team che si è comportato meglio in stagione si laurei come campione della lega, anzi capita con frequenza non trascurabile di vedere come vincenti di una serie le cosiddette squadre *underdogs* e ancora, non è detto che i giocatori abbiano durante il percorso dei playoff le stesse percentuali di realizzazione che hanno mantenuto in precedenza.

È stato inoltre implementato con la randomizzazione, un algoritmo che regola gli infortuni durante i match.

D’altronde la componente random è in linea con l’interpretazione della squadra come azienda: in un procedimento produttivo è impossibile eliminare la componente di variabilità, se i risultati del team sono i prodotti, è quindi chiaro che debbano essere viziati da una componente casuale più o meno marcata.

Proprio a causa di questa componente aleatoria radicata nel software, è stato scelto di consentire all’utente di modificare come parametri di simulazione solo le squadre partecipanti, in quanto sono stati impostati i valori in input per gli algoritmi random solo dopo aver effettuato numerose prove ed ottenuto risultati attinenti alla realtà; una piccola variazione di questi, può dare origine a output assurdi.

Descrizione del data-set utilizzato

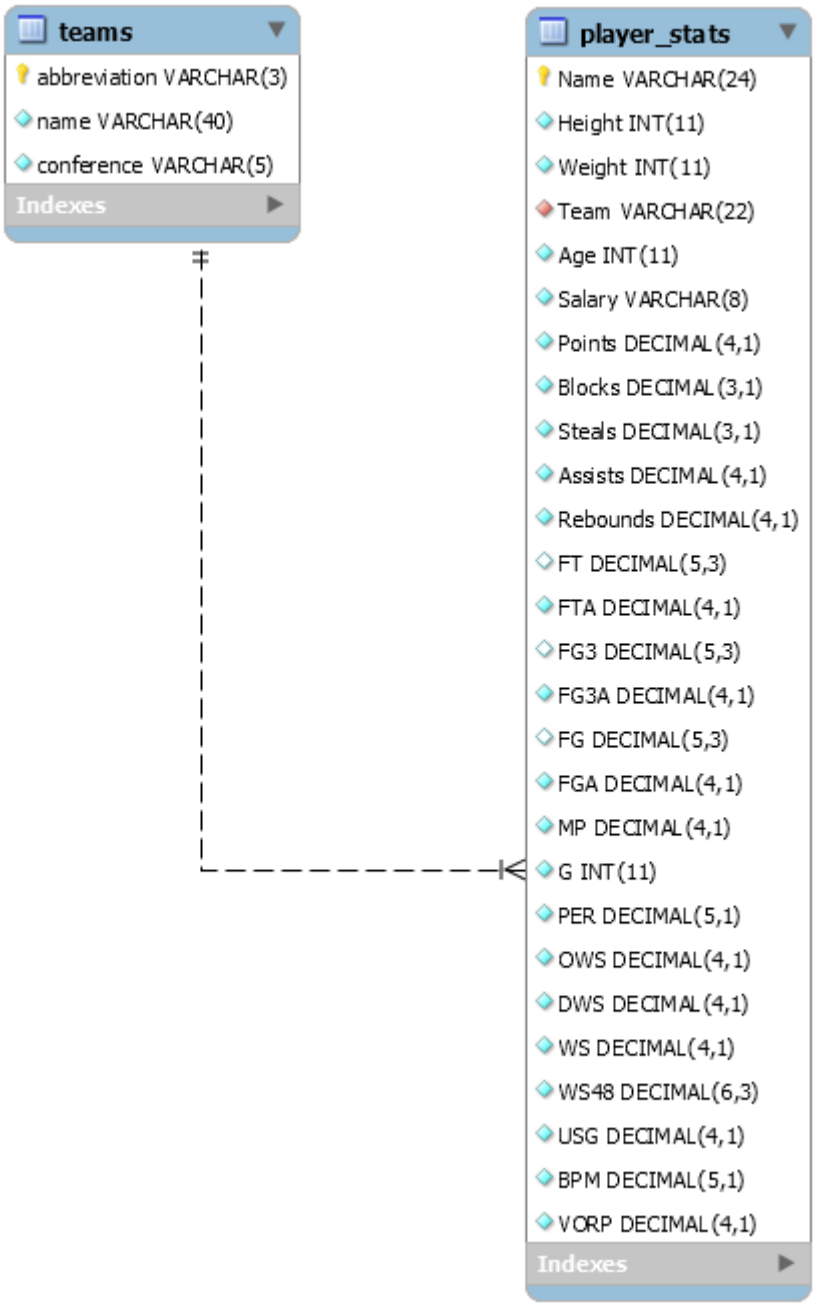
Come già detto l'applicazione si serve di due dataset:

1. Il dataset denominato "teams" che ha il compito di consentire tramite il pattern DAO (Data Access Object), di avere un'indicazione chiara di appartenenza ad una conference specifica. È costituito dai seguenti campi:
    - Abbreviation: abbreviazione della squadra in tre lettere es: "LAL";
    - Name: nome esteso della squadra;
    - Conference: indicazione di appartenenza, quindi "East" oppure "West";

2. Il dataset denominato player_stats che ingloba tutte le statistiche dei giocatori registrate nella stagione regolare, per lo scopo dell'applicazione vengono però considerate solo una porzione di queste, in particolare:
    - Name: nome del giocatore;
    - Team: squadra di appartenenza;
    - Points: media punti;
    - Blocks: media stoppate;
    - Assists: media assist;
    - Rebounds: media rimbalzi;
    - FT: percentuale di realizzazione nei tiri liberi;
    - FTA: media dei tentativi effettuati per i tiri liberi;
    - FG3: percentuale di realizzazione dei tiri da 3 punti;
    - FG3A: media dei tentativi effettuati per i tiri da 3;
    - FG: percentuale di realizzazione dei tiri da 2;
    - FGA: media dei tentativi effettuati per i tiri da 2.

    Tutti i valori medi si rifanno alla stagione regolare, in particolare alla stagione 2018 – 2019.

Importando i data-set descritti in un server SQL vengono generate le tabelle seguenti:



Struttura tabelle SQL.

Descrizione ad alto livello delle strutture dati e degli algoritmi utilizzati

DESCRIZIONE STRUTTURE DATI

L'applicazione è stata interamente realizzata mediante linguaggio Java, in particolare sfruttando gli applicativi *JavaFX* e modellata tramite il software SceneBuilder. Sono stati altresì implementati il pattern MVC (*Model View Controller*) per garantire una corretta divisione delle operazioni del software e quindi delegare al *Model* la logica applicativa e lasciare al *Controller* solo il compito di catturare l'interazione con l'utente ed esporre i risultati; il pattern DAO (*Data Access Object*) ed il pattern ORM (*Object Relational Mapping*).

Strutturalmente l'applicativo è stato diviso in tre packages in modo da tenere separate le diverse funzioni:

- <u>Il package Controller:</u> contenente tutte le classi e relativi files che si occupano di realizzare l'interfaccia grafica, di interagire con l'utente e di ricevere in input i dati da inviare in un secondo momento al *Model.* In particolare sono presenti le classi *Main*, i tre controller (*NBAController, StatsController e FinalsController*), i rispettivi file in formato *fxml* (*Table.fxml, StatsForGame.fxml, GoToFinals.fxml*) ed il file *application.css* per imprimere lo stile grafico;
- <u>Il package Model:</u> il vero "core" dell'applicazione in cui sono presenti tutte le classi necessarie all'elaborazione dei dati, raggruppabili in tre sottocategorie:
    - Le classi che si occupano della logica applicativa: *Model* e *Simulatore*;
    - Le classi di rappresentazione degli oggetti: *Evento*, *Match*, *Player*, *Team* e *Series* e *PlayerAVGStats*;
    - Le classi di test: *TestModel* e *TestSimulatore*.
- <u>Il package DB:</u> con il compito di gestire le connessioni al database mediante la classe *DBConnect* e di estrapolare i dati necessari al package Model attraverso la classe DAO *NBADao.*

DESCRIZIONE DEGLI ALGORITMI UTILIZZATI

All'avvio dell'applicazione saranno già disponibili alla selezione le squadre divise per conference, recuperate dal database attraverso una query specifica e inserite nei *combobox* attraverso il metodo del Controller `setModel(Model model, Stage stage).` Per avviare la simulazione delle serie di una conference, l'utente deve assicurarsi di aver selezionato tutte le squadre e di non aver selezionato più di una volta la stessa squadra, in caso contrario, alla pressione del bottone "*Simulazione Ovest*" o "*Simulazione Est*" verrà individuato l'errore attraverso due algoritmi di

controllo basati su due semplici cicli *for each* e comparirà il messaggio di errore relativo nella text area sottostante.

Nel caso di input corretto il *controller* passerà i dati al *model*, in cui si svilupperà la simulazione:

Il simulatore è qui pensato come un singolo match tra due squadre diverse, pertanto ogni comando `new Simulatore()` verranno resettate tutte le statistiche prodotte dalla simulazione precedente. Da qui nasce l'esigenza di organizzare nel model delle strutture dati adatte a salvare al termine di ogni *partita/simulazione* i risultati ottenuti, che saranno poi usati come punto di partenza per le simulazioni delle fasi più avanzate del tabellone, anche perché per rispecchiare la realtà una squadra prevale sulla sua diretta avversaria ed avanza alla fase successiva, solo dopo avere vinto una *serie al meglio delle 7 partite*; è quindi compito della classe `Series` salvare le informazioni riguardo i singoli matches, l'opzione adottata è quella di usare una lista: `List<Match> matches`, ed ogni *Match* porta con se tutte i risultati della singola simulazione.

La classe `Simulatore` basa il suo funzionamento sulla struttura dati `PriorityQueue`, che a sua volta necessita della definizione della classe `Evento`, che è responsabile di schedulare gli avvenimenti attraverso i propri attributi.

Evento è stata strutturata come segue:

```
public enum TipoEvento{

    FREE_THROW_ATTEMPT,
    THREE_POINTS_ATTEMPT,
    FIELD_GOAL_ATTEMPT,


    INJURED

}

private Integer time;
private TipoEvento type;
private Team team;
private Player player;
```

incapsulando informazioni riguardo al minutaggio (`time`), con la scelta di adottare una rappresentazione del tempo in *interi* in modo da non coinvolgere la libreria *java.time,* non necessaria in quanto non viene compiuta nessuna elaborazione consistente sullo scorrere del tempo; riguardo al tipo dell'evento (`type`); riguardo alla squadra (`team`) e riguardo al giocatore (`player`). Come dall'estratto di codice soprastante sono state individuate 4 tipologie di eventi, 3 per schedulare i tentatiti di tiro ed una per gestire gli infortuni.

Indispensabile al corretto funzionamento del simulatore è il metodo `public int compareTo()`, che come parametro di confronto utilizza il tempo evento.

Come di norma avviene, il simulatore è costituito da due parti fondamentali:

- <u>L'inizializzazione</u>: in cui vengono preparate le strutture dati per salvare i risultati e viene caricata la coda degli eventi; di seguito un estratto di codice del metodo `public void init(Team home, Team away)` per caricare i tentativi da 3 punti.

```
Integer attempt3 = (int) ((((rand.nextInt(this.PROB_TENTATIVO_RANDOM) - rand.nextInt(this.PROB_TENTATIVO_FISSA))
        * homeP.getThreePointsAttempts()) / 100));
for(int i = 0; i < attempt3; i++) {

    if(homeP.getTeam().equals(home.getName())) {

        this.eventList.add(new Evento(null, TipoEvento.THREE_POINTS_ATTEMPT,home, homeP));

    }else {

        this.eventList.add(new Evento(null, TipoEvento.THREE_POINTS_ATTEMPT,away, homeP));

    }

    for(PlayerAVGStats pas : match.getPlayerStats()) {//aggiornamento tentativi
        if(pas.getName().equals(homeP.getName())) {
            pas.setThreeAttempts(pas.getThreeAttempts() + 1);
        }
    }

}
```

  Con questo procedimento vengono estratti i valori di tentativo di tiro giocatore per giocatore, randomizzati attraverso i parametri ***PROB_TENTATIVO_FISSA*** e ***PROB_TENTATIVO_RANDOM***. Vengono creati nuovi eventi passando al costruttore `null` come parametro del tempo, inserendoli poi in una lista e non direttamente nella priority queue in modo da poter invocando il metodo `Collections.shuffle(eventList)` randomizzare le azioni (vale lo stesso principio per gli altri tipo di evento), sarà poi compito del seguente ciclo andare a settare il minutaggio dell'evento e di aggiungerlo alla coda.

```
for(Evento ev : this.eventList) {
    if(matchTime <= matchDurations) {
        matchTime = matchTime +
                ((rand.nextInt(this.TEMPO_RANDOM_AZIONE)) + this.TEMPO_COSTANTE_AZIONE);
        ev.setTime(matchTime);
        this.queue.add(ev);

    }
}
```

  Viene infine posta attenzione in fase di inizializzazione, alla gestione degli infortuni, escludendo dall'algoritmo di caricamento eventi, gli eventuali giocatori indisponibili ed andando a decrementare le giornate di riposo.

- <u>L'esecuzione</u>: in cui estraendo evento per evento dalla `PriorityQueue<Evento>`, attraverso il costrutto `switch`, corrisponderà un diverso ramo dell'algoritmo.

  Nel caso di tipo evento ***FIELD_GOAL_ATTEMPT***, ***THREE_POINTS_ATTEMPT*** oppure ***FREE_THROW_ATTEMPT***, il codice è del tutto analogo se non per il punteggio assegnato in caso di buon esito del tentativo, verrà pertanto di seguito esaminato nel dettaglio il solo estratto di "`case` ***THREE_POINTS_ATTEMPT***".

```java
case THREE_POINTS_ATTEMPT:
    if(ev.getPlayer().getThreePointsPercentage() > rand.nextDouble()) {
        String name = "";
        for(PlayerAVGStats pas : match.getPlayerStats()) {
            if(pas.getName().equals(ev.getPlayer().getName())) {
                pas.setPoint(pas.getPoint() + 3);
                pas.setThreeDone(pas.getThreeDone() + 1);
                name = pas.getName();
            }
            if(!name.equals("") && !pas.getName().equals(name)) {//MODELLAZIONE ASSIST
                if(team.equals(match.getHome())) {
                    Player random = hPlayers.get(rand.nextInt(this.hPlayers.size()));
                    if(!team.getInjured().containsKey(random.getName())) {
                        for(PlayerAVGStats pas2 : match.getPlayerStats()) {
                            if(pas2.getName().equals(random.getName())
                                    && pas2.getAssist() < (random.getAssists() * (1 + rand.nextDouble()))) {
                                if(this.PROB_RANDOM_ASSIST > rand.nextDouble())
                                    pas2.setAssist(pas2.getAssist() + 1);
                            }
                        }
                    }
                }else {
                    Player random = aPlayers.get(rand.nextInt(this.aPlayers.size()));
                    if(!team.getInjured().containsKey(random.getName())) {
                        for(PlayerAVGStats pas2 : match.getPlayerStats()) {
                            if(pas2.getName().equals(random.getName())
                                    && pas2.getAssist() < (random.getAssists() * (1 + rand.nextDouble()))) {
                                if(this.PROB_RANDOM_ASSIST > rand.nextDouble())
                                    pas2.setAssist(pas2.getAssist() + 1);
                            }
                        }
                    }
                }
            }
        }

        if(team.getName().equals(this.match.getHome().getName())) {
            this.homePoints = this.homePoints + 3;
        }else {
            this.awayPoints = this.awayPoints + 3;
```

Questa prima parte di codice regola il **buon esito del tentativo** di tiro da tre punti.
L'algoritmo recupera la percentuale di realizzazione stagionale del giocatore in esame e nel caso in cui fosse superiore ad un numero random compreso tra 0.0 e 1.0 (fornito dalla funzione `rand.nextDouble()`) si avrà un esito positivo, ed a quel punto verranno aggiornate le statistiche del match in quesitone.
Sarà inoltre salvato il nome del giocatore che ha fatto canestro per consentire all'algoritmo di non prenderlo in considerazione nella modellazione degli assist: questa fase che viene suddivisa per team; viene selezionato random un giocatore appartenente alla stessa squadra del giocatore che ha realizzato il canestro e se la media stagionale di assist del primo, maggiorata di una percentuale randomica non viene superata, gli viene attribuito l'assist. È inoltre presente il controllo infortuni: nel caso in cui il team abbia nella sua lista di giocatori infortunati il player che è stato selezionato casualmente per l'assist, non verrà preso in considerazione.
Infine, dopo aver aggiornato le statistiche dei giocatori, l'algoritmo si occupa con l'ultima sezione del codice di aggiornare il punteggio della partita.

```java
}else {
    if(team.equals(this.match.getHome())) {
        if(rand.nextDouble() < this.PROB_RIMBALZO) {
            Player random = aPlayers.get(rand.nextInt(this.aPlayers.size()));
            if(!match.getAway().getInjured().containsKey(random.getName())) {
                for(PlayerAVGStats pas : match.getPlayerStats()) {
                    if(pas.getName().equals(random.getName())) {
                        pas.setRebounds(pas.getRebounds() + 1);
                    }
                }
            }
        }else if(rand.nextDouble() < this.PROB_STOPPATA) {//stoppata
            Player random = aPlayers.get(rand.nextInt(this.aPlayers.size()));
            if(!match.getAway().getInjured().containsKey(random.getName())) {
                for(PlayerAVGStats pas : match.getPlayerStats()) {
                    if(pas.getName().equals(random.getName())
                            && pas.getBlock() < (random.getBlocks() + this.PROB_PLUS_RANDOM_STOPPATA * rand.nextDouble())) {
                        pas.setBlock(pas.getBlock() + 1);
                    }
                }
            }
        }
    }else {
            if(rand.nextDouble() < this.PROB_RIMBALZO) {
                Player random = hPlayers.get(rand.nextInt(this.hPlayers.size()));
                if(!match.getHome().getInjured().containsKey(random.getName())) {
                    for(PlayerAVGStats pas : match.getPlayerStats()) {
                        if(pas.getName().equals(random.getName())) {
                            pas.setRebounds(pas.getRebounds() + 1);
                        }
                    }
                }
            }else if(rand.nextDouble() < this.PROB_STOPPATA) {
                Player random = hPlayers.get(rand.nextInt(this.hPlayers.size()));
                if(!match.getHome().getInjured().containsKey(random.getName())) {
                    for(PlayerAVGStats pas : match.getPlayerStats()) {
                        if(!random.getInjured() && pas.getName().equals(random.getName())
                                && pas.getBlock() < (random.getBlocks() + this.PROB_PLUS_RANDOM_STOPPATA * rand.nextDouble())) {
                            pas.setBlock(pas.getBlock() + 1);
                        }
                    }
                }
            }
    }
}
break;
```

Questa sezione, invece, si occupa del **caso di tentativo fallito**.
Analogamente alla modellazione assist, anche qui l'algoritmo prosegue lungo due "strade" diverse a secondo del team di appartenenza del giocatore.
In caso di tentativo fallito vengono prese in considerazioni tre possibilità in ordine di probabilità:

1. Che un giocatore avversario random prenda il rimbalzo, nel caso in cui non sia infortunato e nel caso in cui risulti verificata la disuguaglianza `rand.nextDouble() < this.`***`PROB_RIMBALZO`*** con ***`PROB_RIMBALZO`*** settata al 80%;
2. Che un giocatore avversario stoppi il tiro, nel caso in cui non sia infortunato e nel caso in cui risulti verificata la disuguaglianza `rand.nextDouble() < this.`***`PROB_STOPPATA`*** con ***`PROB_STOPPATA`*** settata al 50%;
3. Che non accada nulla.

Vengono quindi aggiornata le relative statistiche.
Le disuguaglianze sono state introdotte e settate come sopra descritto a seguito di numerose prove in modo da rendere l'output il più realistico possibile.
Come sopra già esposto, il simulatore si occupa del singolo match; **è invece il *Model* che gestisce la simulazione di una serie** tra due team e ne memorizza i risultati. Di seguito il codice dell'algoritmo.

```java
public Team SimulationWinner(Team home, Team away) {

    Series series = new Series(home, away);

    this.winA = 0;
    this.winH = 0;

    for(int i = 1; i <= 7 ; i++) {
        if(winA == 4 || winH == 4) {
            break;
        }else{
            sim.init(home, away);
            sim.run();

            if(sim.getMatch().getWinner().equals(home)) {
                this.winH++;
                series.setWinHome(series.getWinHome() + 1);
                series.getMatches().add(sim.getMatch());
            }else {
                this.winA++;
                series.setWinAway(series.getWinAway() + 1);
                series.getMatches().add(sim.getMatch());
            }

            this.matchs.add(sim.getMatch());
            if(this.eastWinner != null && this.westWinner != null) {
                this.finalStats.addAll(sim.getMatch().getPlayerStats());
            }


        }


    }

    if(winA > winH) {
        this.winnerTeamMap.put(away.getAbbreviation(), away);
        series.setWinner(away);
        if(this.WestTeams.contains(home)) {
            this.seriesMapWest.put(idSeriesWest, series);
            this.idSeriesWest ++;
        }else {
            this.seriesMapEast.put(idSeriesEast, series);
            this.idSeriesEast ++;
        }

        return away;
    }

    else {
        this.winnerTeamMap.put(home.getAbbreviation(), home);
        series.setWinner(home);
        if(this.WestTeams.contains(home)) {
            this.seriesMapWest.put(idSeriesWest, series);
            this.idSeriesWest ++;
        }else {
            this.seriesMapEast.put(idSeriesEast, series);
            this.idSeriesEast ++;
        }
        return home;
    }
}
```

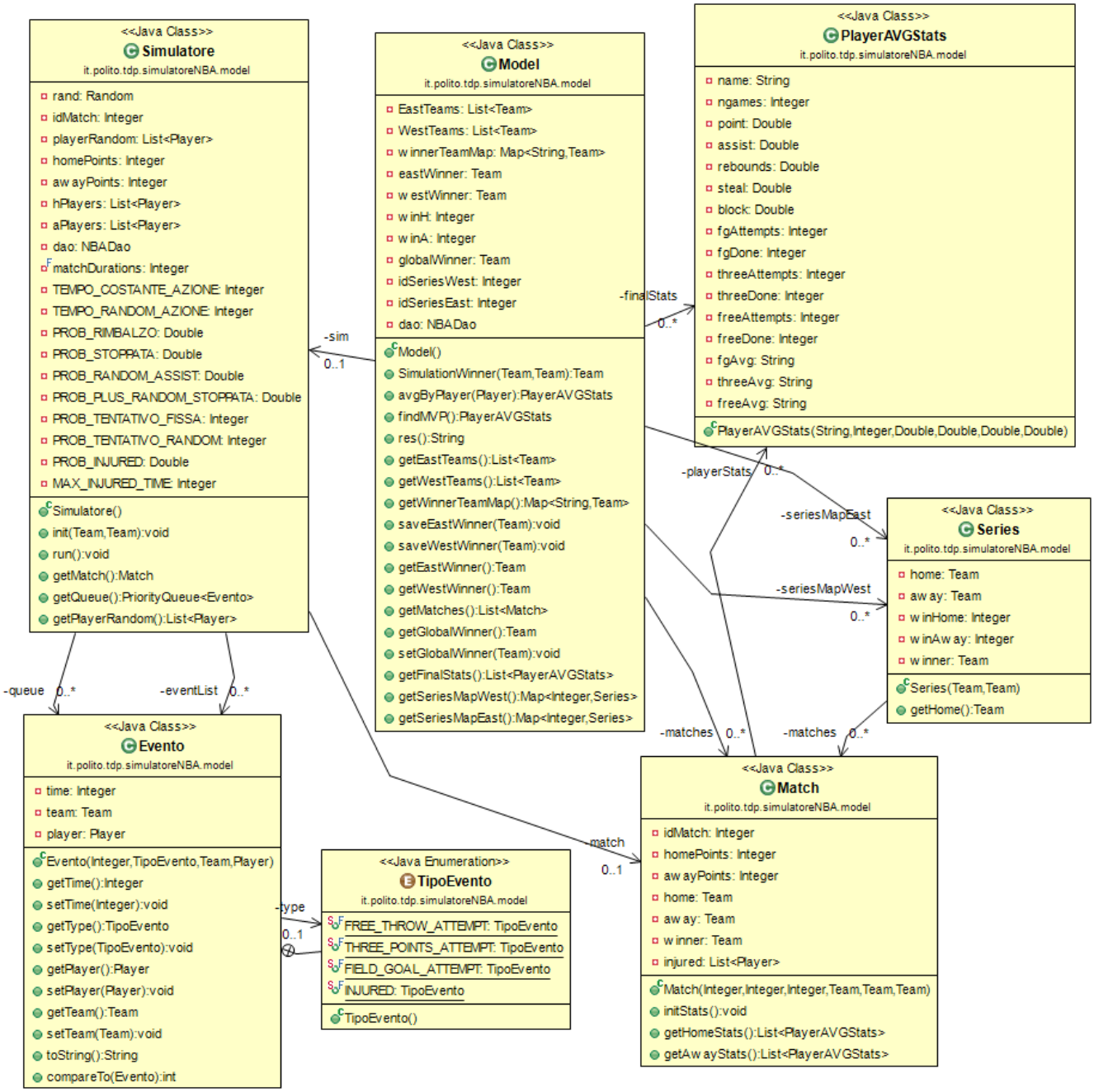
È costituito da un ciclo for che si ripete fino a che non siano state giocate al massimo 7 partite, oppure una della due squadre ne abbia vinte 4.
Ad ogni iterazione viene inizializzato ed eseguito un nuovo simulatore ed in base ai risultati di quest'ultimo, vengono aggiornati i risultati della serie separando tra team casalingo e team ospite;

viene memorizzato il match in una lista e nel caso di `eastWinner` e `westWinner` già decretati, vengono salvate le statistiche dei giocatori in modo da permettere in un secondo momento di calcolarne la media grazie ad un metodo apposito del *Model*.
Una volta terminato il ciclo, e quindi decretato un vincitore, il metodo salva infine in una mappa (ne sono state create due, una per ogni conference) il risultato della serie ed incrementa l'`idSeries` utilizzato come chiave.

Meritevole di menzione è infine il metodo utilizzato per adattare lo `StatsController` in base alla serie selezionata, con i dati relativi.
È basato su di un *MouseEvent*: a seconda del *Text Field* cliccato, ne viene recuperato l'id (assegnato tramite SceneBuilder) e viene passato nel metodo `setModel()` come parametro al controller, che poi tramite una serie di condizioni `if`, recupererà dalle mappe delle serie i dati necessari. Di seguito l'estratto del codice.

```
void doViewStats(MouseEvent event) {

    txtLog.clear();

    try {

        FXMLLoader loader = new FXMLLoader(getClass().getResource("StatsForGame.fxml"));
        BorderPane root = (BorderPane) loader.load();
        Scene scene = new Scene(root);

        if(event.getSource() instanceof TextField) {
            TextField source = (TextField) event.getSource();

            if(source.getText().equals("")) {
                txtLog.appendText("Prima di accedere alle statistiche, è necessario simulare la partita.");
                return;
            }

            String idField = source.getId();

            StatsController controller = loader.getController();
            controller.setModel(model, stage, idField);

            Stage s= new Stage();

            s.setScene(scene);
            scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
            s.setTitle("Statistiche Serie");
            s.show();
        }

    } catch(Exception e) {
        e.printStackTrace();
    }

}
```

Diagramma delle classi principali



Per le classi `PlayerAVGStats`, `Match` e `Series`, sono stati omessi i metodi getter & setter.

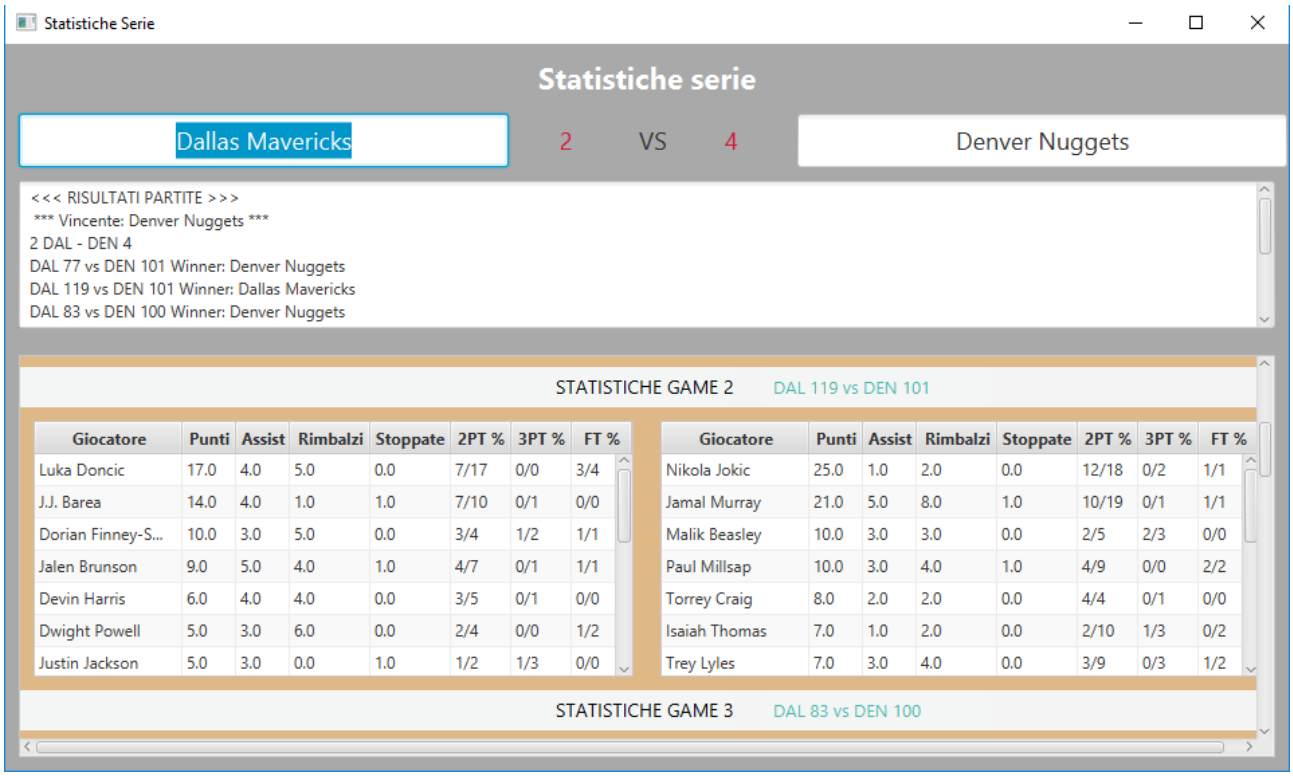
Videate dell'applicazione con risultati ottenuti e link al video dimostrativo del software

Verranno ora esposte una serie di videate dell'applicazione in funzione, ordinate cronologicamente in base all'esperienza dell'utente.

Videata Schermata "Home"

Simulazione della Western Conference terminata (tramite simulazione veloce) e Eastern Conference parzialmente avviata; è la prima schermata che l'utente incontra avviando l'applicazione.

Videata schermata “Statistiche Serie”

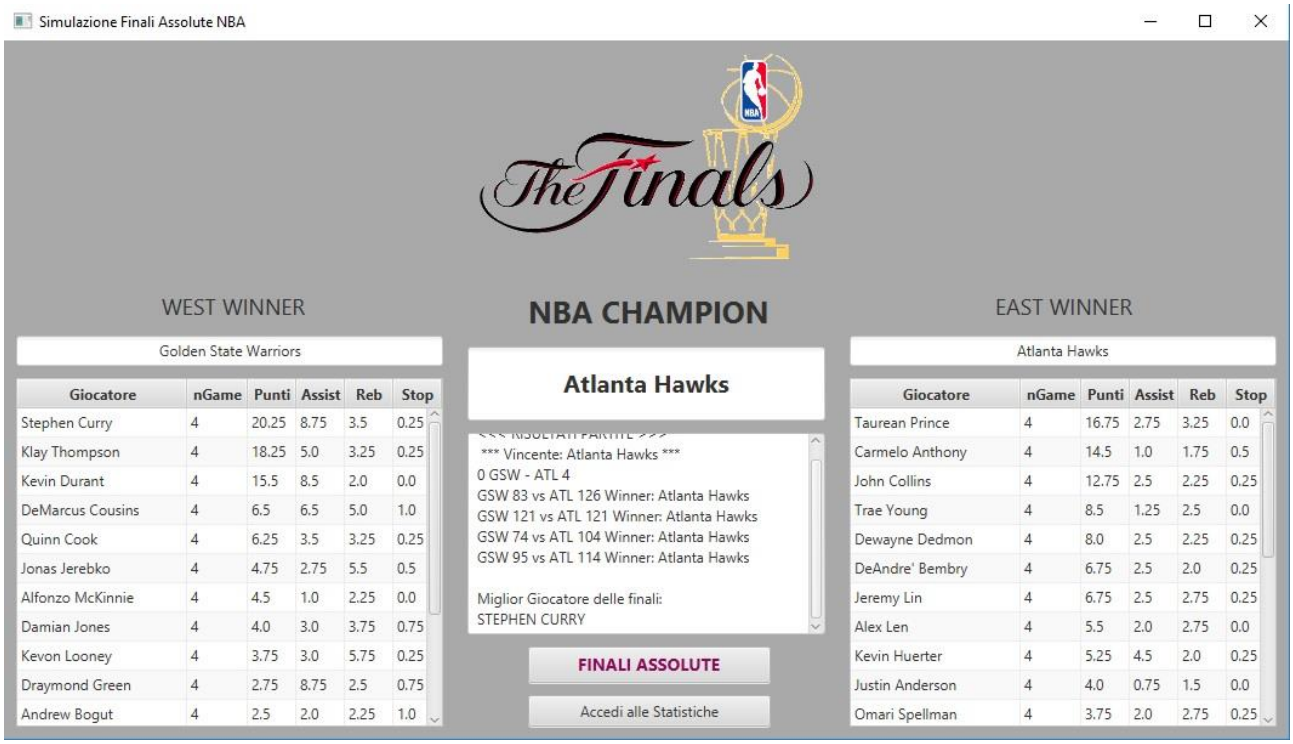


Giocatore	Punti	Assist	Rimbalzi	Stoppate	2PT %	3PT %	FT %
Luka Doncic	17.0	4.0	5.0	0.0	7/17	0/0	3/4
J.J. Barea	14.0	4.0	1.0	1.0	7/10	0/1	0/0
Dorian Finney-S...	10.0	3.0	5.0	0.0	3/4	1/2	1/1
Jalen Brunson	9.0	5.0	4.0	1.0	4/7	0/1	1/1
Devin Harris	6.0	4.0	4.0	0.0	3/5	0/1	0/0
Dwight Powell	5.0	3.0	6.0	0.0	2/4	0/0	1/2
Justin Jackson	5.0	3.0	0.0	1.0	1/2	1/3	0/0

Giocatore	Punti	Assist	Rimbalzi	Stoppate	2PT %	3PT %	FT %
Nikola Jokic	25.0	1.0	2.0	0.0	12/18	0/2	1/1
Jamal Murray	21.0	5.0	8.0	1.0	10/19	0/1	1/1
Malik Beasley	10.0	3.0	3.0	0.0	2/5	2/3	0/0
Paul Millsap	10.0	3.0	4.0	1.0	4/9	0/0	2/2
Torrey Craig	8.0	2.0	2.0	0.0	4/4	0/1	0/0
Isaiah Thomas	7.0	1.0	2.0	0.0	2/10	1/3	0/2
Trey Lyles	7.0	3.0	4.0	0.0	3/9	0/3	1/2

Accessibile cliccando l’abbreviazione della squadra vincente di una serie (se in NBAController) oppure tramite il pulsante “Accedi alle Statistiche” (se in FinalsController). Riporta risultati totali e parziali della serie ed i dati per ogni giocatore e per ogni game giocato, in particolare nella text area vengono visualizzati i risultati senza scendere nel dettaglio, mentre nello Scroll Pane sottostante compare una serie di tabelle che esaminano quantitativamente le statistiche.

Videata “Finali Assolute”



È la videata finale, dopo aver avviato l’ultima simulazione. Vengono visualizzati nelle sue tabelle i valori medi delle statistiche dei giocatori calcolati sulla serie delle finali assolute; nella text area viene anche fornito come output il giocatore incoronato MVP delle finals.

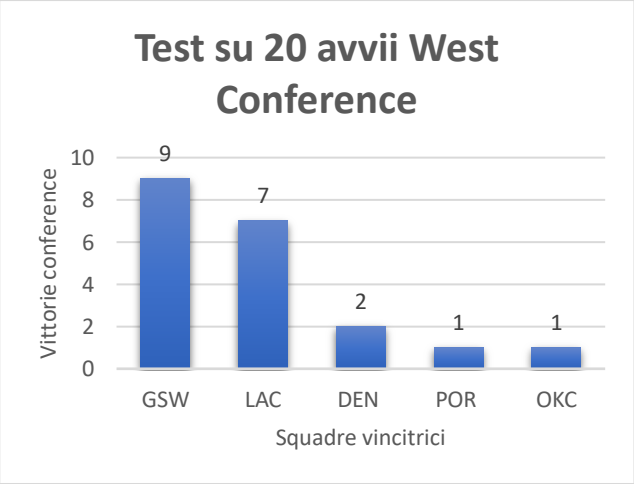
Link al video YouTube: https://www.youtube.com/watch?v=l4M1tCTtI_s

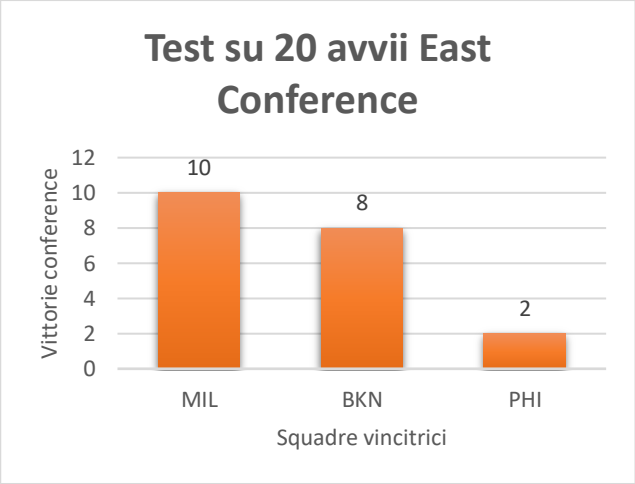
Valutazione dei risultati ottenuti e conclusioni

Come tempistiche l'applicazione ha dei tempi pressochè istantanei, in quanto gli algoritmi utilizzano cicli su liste con condizioni filtro che ne bloccano l'iterazione eccessiva ed inoltre ci si appoggia su un database di dimensioni relativamente contenute facendo uso della libreria *Hikari.*

Analizzando invece i dati ottenuti, si può avere una previsione qualitativa dell'andamento delle squadre lungo il percorso dei playoff; come già detto la componente randomica è radicata nella logica applicativa, quindi i risultati devono essere interpretati con le dovute cautele.

Dopo la premessa sopra esposta si ravvisa una confrontabilità totale dei risultati con i valori reali dei playoff giocati nella stagione di riferimento del database (2019), valori che testimoniano quindi una modesta affidabilità dell'applicazione, sia in termini di statistiche dei giocatori, sia in termini di risultati delle partite/ serie; come ci si può aspettare aumentando il numero di ripetizioni della simulazione, alla lunga le squadre che nella stagione regolare hanno avuto un impatto migliore, ottengono un piazzamento migliore. È stato eseguito un test sfruttando l'interfaccia Home strutturato su 20 ripetizioni andando a selezionare le squadre che nella stagione 2018 – 2019 si siano effettivamente qualificate per i playoff. I risultati sono riportati nei grafici seguenti:





Sia per la conference dell'ovest, che per la conference dell'est, ne risulta che dopo un numero modesto di ripetizioni della simulazione, vincano la propria conference le squadre che si siano qualificate ai playoff con in risultato migliore (Golden State Warriors per la conference ovest e Milwaukee Bucks per la conference est). Si nota inoltre che la componente randomica garantisce una certa variabilità negli output, in questo caso in modo più evidente per la conference dell'ovest.

Si precisa che nella simulazione, riferendosi ai team sono stati utilizzati i termini *home* e *away*, ma questi sono solo stati usati dal programmatore per avere un riferimento chiaro nella gestione degli algoritmi; una possibile successiva implementazione potrebbe essere proprio quella di tenere in conto il fattore "partita in casa" o "partita in trasferta" ammorbidendo le disuguaglianze che sfruttando gli algoritmi random nel caso di match casalingo.

Quest'opera è stata rilasciata con licenza Creative Commons Attribuzione - Non commerciale - Condividi allo stesso modo 4.0 Internazionale.

Copia della licenza consultabile al sito web: http://creativecommons.org/licenses/by-nc-sa/4.0/