Politecnico di Torino

Corso di Laurea in Ingegneria Gestionale
Classe L8 - Ingegneria dell'Informazione
A.A. 2020/2021
Sessione di Laurea Settembre 2021

Politecnico di Torino

Applicazione per la gestione di una squadra NBA

Relatore:
Prof.Fulvio Corno

Candidato:
Davide Balossino
256558

INDICE

1.PROPOSTA DI PROGETTO

Studente proponente

s256558 Davide Balossino

Titolo della proposta

Applicazione per la gestione di una squadra NBA

Descrizione del problema proposto

Il problema che si vuole risolvere è quello della gestione di una squadra NBA, rispettando alcuni vincoli come nella realtà: in primis il limite del salary cap, ovvero il fatto che il totale degli stipendi dei giocatori sotto contratto con una franchigia non possa superare una certa soglia, se non in alcune eccezioni descritte meglio nell'ultima sezione; un'altra condizione da tener presente è il numero minimo e massimo di giocatori per squadra.

L'applicativo permette di cercare giocatori in base alle caratteristiche desiderate, e individuare quali atleti della propria rosa cedere per poter arrivare ad acquisire il cestista ambito. Infine è presente la possibilità di liberarsi di alcuni dei propri giocatori e sostituirli con i migliori cestisti per ottimizzare la rosa in base alle caratteristiche cercate.

Descrizione della rilevanza gestionale del problema

Il problema della gestione di una squadra riguarda direttamente i general manager delle franchigie NBA; ma anche più semplicemente chiunque si cimenti a giocare ai videogiochi della saga "NBA 2K" per PlayStation o XBOX, in cui ci si ritrova a vestire i panni del general manager della squadra scelta con l'obbiettivo di migliorarla e condurla alla vittoria del titolo finale.

Tale problema è quindi strettamente gestionale, le franchigie NBA possono ormai essere considerate a tutti gli effetti delle vere aziende, con valori che, come nel caso dei New York Knicks, arrivano oltre i 4 miliardi di dollari (dato Forbes del 2019). Il successo della squadra dipende in primis da chi ha il compito di assemblarla: il general manager.

Descrizione dei data-set per la valutazione

I dati relativi agli stipendi dei diversi giocatori derivano dal data-set al link:

https://www.kaggle.com/isaienkov/nba2k20-player-dataset

I dati relativi invece alle statistiche avanzate dei singoli giocatori derivano dal data-set al link:

https://www.kaggle.com/nicklauskim/nba-per-game-stats-201920

Entrambi i data-set fanno riferimento alla stagione passata, 2019/20, che è stata conclusa lo scorso 12 ottobre.

In entrambi i casi sono presenti alcune colonne che al fine di questo lavoro contengono dati superflui, e sono quindi state eliminate.

Inoltre nel secondo data-set sono presenti più righe (652 in totale) rispetto al primo (430). Questo perché nel caso di giocatori che abbiano cambiato squadra in corso durante la stagione, nel secondo data-set vengono riportate su più righe sia le medie tenute in ciascuna squadra, sia in totale. Per lo svolgimento del programma si prendono in analisi le medie totali, che fanno quindi riferimento a tutta la stagione, ed il giocatore viene considerato parte della squadra in cui si trovava a fine stagione.

Le righe superflue del secondo data-set saranno quindi rimosse.

Il formato originale dei dati era CVS, sono stati poi importati in HeidiSQL in modo da poterli gestire come visto a lezione.

Inoltre si è utilizzato un dataset relativo alle squadre, che era stato in precedenza creato da un altro studente e presente al link:
https://github.com/rubenIng93/nba_teams_sql/blob/master/teams.sql .

Descrizione preliminare degli algoritmi coinvolti

Il programma si compone di diverse funzionalità: sarà possibile svolgere operazioni di ricerca, a seguito di un input da parte dell'utente, per trovare i giocatori più idonei ai parametri impostati, ad esempio sarà possibile cercare uno scorer (colui che segna molti punti) oppure un assistman e così via. Per presentare liste di giocatori ordinate in basa alla richiesta fatta ci si baserà sulle statistiche dell'ultima stagione 2019/20 conclusa pochi mesi fa.

Il tool implementa anche un algoritmo di ricorsione utile per individuare i giocatori di cui liberarsi in modo tale da poterne ingaggiare uno nuovo: l'utilità di tale metodo è spiegata meglio nella sezione successiva.

L'applicazione si propone inoltre di risolvere il "Knapsack problem", ossia problema dello zaino, e per farlo si baserà sull'euristica di greedy: gli oggetti saranno ordinati in base al loro costo unitario, ed esaminati in ordine decrescente. L'oggetto corrente viene inserito se e solo se il suo peso non supera la capacità residua, nel caso di questa applicazione rappresentato dal limite del salary cap.

Per ogni giocatore sarà generato un costo unitario tenendo conto delle statistiche nella specialità indicata dall'utente (ad esempio punti o rimbalzi o assist ecc), dell'età e dello stipendio.

Tale euristica sarà implementata anche attraverso l'uso della ricorsione.

Descrizione preliminare delle funzionalità previste per l'applicazione software

L'applicazione offrirà le funzioni necessarie per gestire al meglio una franchigia NBA nei panni di un general manager: sia nella realtà sia limitatamente al mondo del gaming.

In primo luogo l'utente dovrà scegliere la squadra di cui curare gli interessi.

Una volta presa tale decisione può sfruttare le diverse funzioni messe a disposizione.

Per gestire una squadra è necessario conoscere e osservare attentamente i giocatori in tutta la lega, sarà infatti permesso di impostare vari parametri di indagine per trovare gli atleti che, sulla base delle loro statistiche, soddisfino maggiormente la ricerca fatta.

Se un giocatore viene ritenuto particolarmente interessante, o lo si desidera ottenere per un qualunque altro motivo, sarà possibile chiedere all'applicativo di individuare i cestisti della propria squadra da cedere in modo da permettersi, rispettando il salary cap, l'ingaggio del giocatore voluto. Verranno presentate le diverse soluzioni possibili; il tutto rispettando il numero minimo e massimo di giocatori che si possono avere in rosa, rispettivamente 12 e 17.
Nel caso in cui, la squadra possa direttamente permettersi l'ingaggio del cestista desiderato senza doversi prima liberare di alcuni suoi giocatori, l'utente sarà avvisato di tale opportunità.

Il rispetto del tetto salariale massimo è un punto importante per rendere l'esperienza realistica. Il salary cap come detto in precedenza rappresenta il totale degli stipendi dei giocatori e deve essere sotto una certa soglia uguale per tutte le squadre. Vi sono però alcune eccezioni per cui una squadra può trovarsi al di sopra del salary cap, questo è possibile che avvenga solo quando un team si trova a dover rinnovare un giocatore già appartenente alla sua rosa, a cui potrebbe essere offerto un certo compenso anche a costo di sforare il limite salariale.
Tuttavia una volta superato il salary cap la squadra potrà ingaggiare nuovi giocatori solo a condizione che essi abbiano uno stipendio pari al minimo salariale fissato dalla NBA.
Se invece la squadra volesse acquisire un giocatore con un ingaggio superiore al minimo dovrà prima liberarsi di uno (o più) suoi giocatori di modo tale che il 125% della somma dei loro stipendi non superi l'ingaggio del cestista che si vuole ottenere.

Un'ultima funzione messa a disposizione dell'utente permette di scegliere in prima istanza i giocatori della propria rosa ritenuti superflui e di cui liberarsi. A seguire sarà possibile impostare i criteri di ricerca per nuovi giocatori (ad esempio si potrà impostare di volere uno scorer e un assistman, dando a uno dei due la priorità) e verrà proposto dall'applicativo (risolvendo il problema dello zaino) il o i migliori cestisti possibili.

2.DESCRIZIONE DETTAGLIATA DEL PROBLEMA

Le società sportive hanno assunto negli ultimi anni, sia dal punto di vista economico che gestionale, la rilevanza e la struttura organizzativa che caratterizzano le grandi aziende.
Secondo Forbes nel 2021, nonostante la pandemia, il valore medio delle 50 squadre sportive più importanti è salito del 9.9% rispetto all'anno precedente; e sempre secondo i dati forniti dalla rivista statunitense ben 3 franchigie NBA si trovano nelle prime 7 posizioni dei club sportivi con maggiore valore; in particolare a guidare la NBA ci sono i New York Knicks con un valore di 5 miliardi USD. Alla sesta posizione si trovano i Golden State Warriors, la cui quotazione, nei soli ultimi 5 anni, è aumentata del 147%.
Si tratta quindi di vere e proprie aziende in continua crescita, e non c'è da stupirsi se queste investono in maniera sempre più consistente sull'analisi dei dati.
Nella NBA oggi tutte le franchigie hanno un gruppo di data analyst, di grande importanza in qualunque decisione riguardo la squadra.

Ponendo l'attenzione sul mondo NBA, l'applicazione ha lo scopo di aiutare nell'analisi dei giocatori, andando a individuare, sulla base delle statistiche, quali potrebbero essere più funzionali per la propria squadra.
Per rendere l'applicazione realistica si è tenuto conto del salary cap.

Come in tutte le più importanti leghe americane, anche nella National Basketball Association, è presente il salary cap.
Questo indica il totale degli stipendi dei giocatori della propria squadra e deve essere sotto una certa soglia, fissato per la stagione 2019-2020 a 109.14 milioni USD.
Tuttavia, in NBA si parla di soft cap, ossia flessibile. A differenza dell'Hard Cap, dove il limite del Salary non può essere superato per nessun motivo e non sono ammesse eccezioni, il Soft Cap permette alle franchigie di poter mettere sotto contratto giocatori e/o effettuare scambi eccedendo il limite del Salary Cap, ma solo rispettando certe condizioni.
Come accennato prima, è quindi possibile rinnovare il contratto di un proprio giocatore, anche ad una cifra che porti a sforare il Cap massimo.
Una volta che ci si trova sopra alla cifra di 109.14 milioni non è però possibile ingaggiare nuovi cestisti a meno che questi non guadagnino il minimo salariale, mentre rimane possibile scambiare i propri giocatori con altri team, a patto però che la somma dei salari in entrata non superi il 125% di quelli in uscita.
Il minimo salariale cambia in base all'esperienza del giocatore nella lega: più questa è alta, maggiore sarà il minimo salariale a cui il giocatore avrà diritto, come illustrato nella tabella proposta di seguito.

Years of Experience	Salary
0	$898,310
1	$1,445,697
2	$1,620,564
3	$1,678,854
4	$1,737,145
5	$1,882,867
6	$2,028,594
7	$2,174,318
8	$2,320,044
9	$2,331,593
10+	$2,564,753

Vi possono essere casi in cui il salario sia inferiore a 898.310$, questo perché il giocatore in realtà appartiene alla squadra di G-League (ovvero la lega di sviluppo, dove far giocare i giovani o le riserve. Tuttavia nel corso della stagione, è possibile che uno di questi firmi un contratto con la prima squadra, venendo quindi considerato come componente del roster); qualunque salario inferiore a 898.310$ è comunque considerato come minimo salariale.

Per quanto riguarda l'applicativo, nella funzionalità che permette di cedere uno o più dei propri giocatori, il tool fornirà automaticamente, sulla base delle specifiche dell'utente, i migliori cestisti possibili per sostituirlo/i, andando a risolvere quello che è definito come "il problema dello zaino", rispettando i limiti salariali.

Questo problema è infatti caratterizzato dalla presenza di un contenitore (lo zaino) e da una quantità di oggetti aventi ciascuno un peso ed un valore. Lo scopo è quello di riempire lo zaino con gli oggetti a disposizione per ottenere la maggior valutazione possibile senza però oltrepassare il limite di peso sostenibile dallo zaino stesso.

Il vincolo del salary cap è alla base anche del metodo che permette di acquistare nuovi giocatori. Una volta selezionati gli atleti desiderati, sarà l'applicazione a indicare, sulla base di un metodo ricorsivo, le diverse possibilità di cestisti appartenenti alla propria squadra da cedere, per potersi permettere l'ingaggio di quelli desiderati. Qualora la situazione del team lo permetta, o i giocatori selezionati abbiano uno stipendio pari al limite salariale, il metodo fornirà in output un messaggio all'utente riguardo alla possibilità di mettere sotto contratto tali cestisti senza bisogno di doverne cedere alcuno.

Oltre ai due algoritmi ricorsivi sopra descritti, è possibile cercare un atleta per analizzarne le statistiche, oppure listare i giocatori sulla base dei parametri scelti dall'utente.

3.DESCRIZIONE DEL DATA-SET UTILIZZATO

Per realizzare l'applicativo sono stati utilizzati diversi dataset. Tutti facenti riferimento alla stagione 2019-20.

1) https://www.kaggle.com/isaienkov/nba2k20-player-dataset.
   Questo è stato utilizzato principalmente per ricavare i dati riguardanti lo stipendio dei diversi giocatori, ma anche per trarre le informazioni su peso e altezza.

2) https://www.kaggle.com/nicklauskim/nba-per-game-stats-201920
   A questo link si trovano invece cinque diversi dataset. Di questi ne sono stati considerati due: nba_2020_advanced.cvs e nba_2020_per_game.cvs.
   Il primo presenta le statistiche avanzate per ogni giocatore. Dei vari campi ne sono stati presi in considerazione solo alcuni, ovvero:
   - Nome del giocatore
   - Posizione
   - Età
   - Squadra
   - Box Plus-Minus: indica il rendimento di un giocatore mediante il conteggio della differenza tra punti fatti e subiti dalla sua squadra durante la permanenza in campo del giocatore stesso
   - VORP: acronimo di Value Over Replacement Player ed è una statistica dipendente dal Plus-Minus: si prende il valore di quest'ultima per creare una scala di valutazione univoca per tutti i giocatori

   Per quanto riguarda il secondo dataset, questo riporta altre voci statistiche, che sono state tutte incluse ed utilizzate nel programma.

   Per entrambi è stata aggiunta manualmente una ulteriore colonna "INJ" per segnalare se il giocatore fosse infortunato, e non avesse quindi potuto far registrare alcuna statistica per quell'anno.

3) Il dataset denominato "teams" è stato utilizzato per specificare le informazioni riguardo alle squadre.
   È costituito dai seguenti campi:
   - Abbreviation: abbreviazione della squadra in tre lettere.
   - Name: nome esteso della squadra.
   - Conference: indicazione di appartenenza, quindi "East" oppure "West".

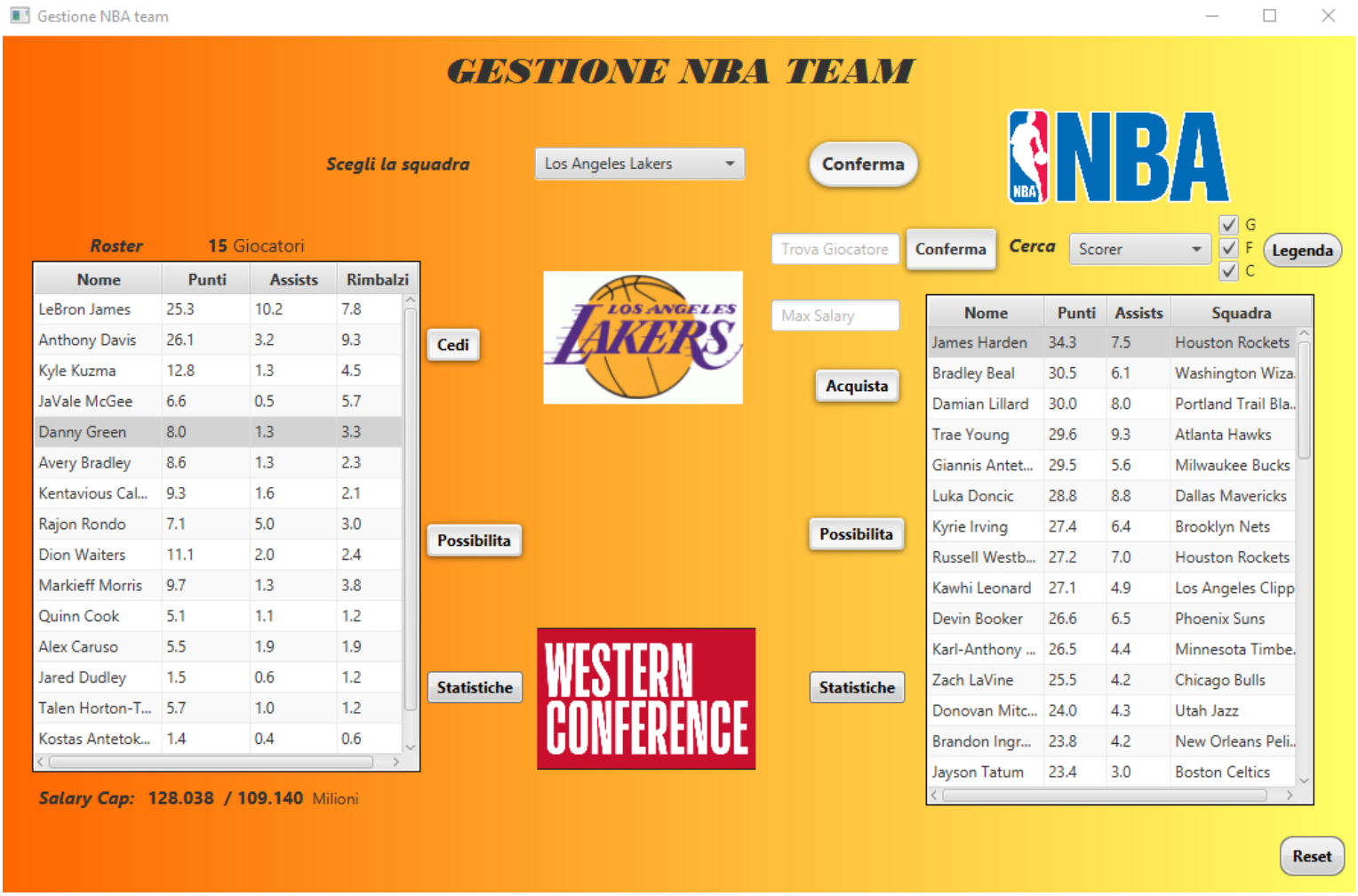
Il dataset presente al primo link e i due dataset presenti al secondo link sono stati in parti modificati per fare corrispondere il nome dei giocatori che in alcuni casi risultava diverso (es. JR Smith in un dataset e J.R. Smith negli altri due).

4.Descrizione ad alto livello delle strutture dati e degli algoritmi utilizzati

L'applicazione è stata realizzata tramite linguaggio Java, sfruttando anche l'applicativo SceneBuilder e il linguaggio CSS per quanto riguarda la parte grafica.
Il programma Java è stato realizzato implementando sia il pattern MVC per garantire una corretta divisione delle operazioni e quindi delegare al *Model* la logica applicativa.
L'applicativo è quindi diviso in tre packages: controller, db, model.

ALGORITMI

Quando viene avviata l'applicazione, come primo passo, l'utente deve selezionare quale fra le 30 squadre NBA gestire. Una volta presa tale decisione vi sono diverse possibilità.



Nella parte a sinistra viene presentato il roster del team scelto e la situazione salariale.
È poi possibile selezionare un cestista della squadra e aggiungerlo tramite il bottone "Cedi" alla lista dei giocatori da cedere; finita la selezione, premere poi su "Possibilità".
Si apre dunque una nuova schermata dove l'utente può scegliere le caratteristiche in base a cui l'applicativo ricerca i giocatori con cui sostituire quelli inseriti nella lista delle cessioni.
Il controller passa i parametri utili al model, dove è implementata la logica applicativa nel seguente metodo.

Viene passato al metodo “trovaMiglioriGiocatori” una lista di Archetipi, che è riempita in base ai parametri forniti in input dall’utente, e lo spazio salariale disponibile.

```
public List<Giocatore> trovaMiglioriGiocatori(List<Archetipo> scelti, Integer spazioSalariale){

    trovati.clear();
    ruoloFattoRimbalzi=null;
    ruoloFattoAssist=null;
    ruoloFattoScorer=null;


    List<String> ruoli=new ArrayList<String>();
    List<String> ordine=new ArrayList<String>();
    for(Archetipo a:scelti) {
        if(a.getTipo().equals("Scorer")) {
            ruoli.add(a.getRuolo());
            ordine.add("Scorer");
        }

        if(a.getTipo().equals("Assistman")) {
            ruoli.add(a.getRuolo());
            ordine.add("Assistman");
        }

        if(a.getTipo().equals("Rimbalzista")) {
            ruoli.add(a.getRuolo());
            ordine.add("Rimbalzista");
        }

        if(a.getTipo().equals("Tiratore da 3")) {
            ruoli.add(a.getRuolo());
        if(a.getTipo().equals("Tiratore da 2")) {
            ruoli.add(a.getRuolo());
            ordine.add("Tiratore da 2");
        }

        if(a.getTipo().equals("Uomo squadra")) {
            ruoli.add(a.getRuolo());
            ordine.add("Uomo squadra");
        }

    }
    List<Giocatore> parziale=new ArrayList<Giocatore>();

    faiRicorsione(spazioSalariale,parziale,ruoli,ordine,0,ruoli.size());

    return trovati;
}
```

La classe Archetipo è composta da due stringhe, una ad indicare il ruolo del giocatore e l’altra le caratteristiche (es. scorer, assistman).
Nelle prime righe vengono ripulite ed inizializzate alcune variabili per essere sicuri che non contengano valori derivanti da una precedente chiamata dal metodo.
Dopo di che si va ad analizzare la lista di Archetipi chiamata “scelti”, che era stata passata come parametro.
Viene poi invocato il metodo privato “faiRicorsione”: a cui viene passato, tra i vari parametri, lo spazio salariale a disposizione per i nuovi giocatori, e due liste di stringhe che rappresentano rispettivamente i ruoli selezionati e il loro ordine, fondamentale per stabilire la priorità.

```
        for(Giocatore g:listaTiratoriDa3(spazioSalariale,ruoli.get(livello))) {
            if(!parziale.contains(g)) {
                if(parziale.size()<numero && (g.getPosizione().contains(ruoli.get(livello)) || ruoli.get(livello).contains(g.getPosizione()))) {
                    parziale.add(g);
                    faiRicorsione(spazioSalariale-g.getSalary(),parziale,ruoli,ordine,livello+1,numero);
                    if(parziale.size()==numero) {
                        break;
                    }
                }
            }
        }
    }
}

    if(parziale.size()==numero) {
        break;
    }

    if(ordine.get(livello).equals("Tiratore da 2")) {
        for(Giocatore g:listaTiratoriDa2(spazioSalariale,ruoli.get(livello))) {
            if(!parziale.contains(g)) {
                if(parziale.size()<numero && (g.getPosizione().contains(ruoli.get(livello)) || ruoli.get(livello).contains(g.getPosizione()))) {
                    parziale.add(g);
                    faiRicorsione(spazioSalariale-g.getSalary(),parziale,ruoli,ordine,livello+1,numero);
                    if(parziale.size()==numero) {
                        break;
                    }
                }
            }
        }
    }
}

      if(parziale.size()==numero) {
          break;
      }

      if(ordine.get(livello).equals("Uomo squadra")) {
          for(Giocatore g:listaUominiSquadra(spazioSalariale,ruoli.get(livello))) {
              if(!parziale.contains(g)) {
                  if(parziale.size()<numero && (g.getPosizione().contains(ruoli.get(livello)) || ruoli.get(livello).contains(g.getPosizione()))) {
                      parziale.add(g);
                      faiRicorsione(spazioSalariale-g.getSalary(),parziale,ruoli,ordine,livello+1,numero);
                      if(parziale.size()==numero) {
                          break;
                      }
                  }
              }
          }
      }
  }
}

}

                    faiRicorsione(spazioSalariale-g.getSalary(),parziale,ruoli,ordine,livello+1,numero);
                    if(parziale.size()==numero) {
                        break;
                    }
                }
            }
        }
    }
}

    if(parziale.size()==numero) {
        break;
    }

    if(ordine.get(livello).equals("Tiratore da 3")) {
```

Per ogni livello si va ad analizzare in quale caso ricada il tipo specificato dall'utente (scorer, assistman, rimbalzista ecc.); in base a questo si ottiene una lista di giocatori ordinata in modo decrescente secondo la statistica corrispondente, e, se il primo giocatore della lista non è ancora stato aggiunto e gioca nel ruolo richiesto, allora verrà aggiunto alla lista "parziale", da cui poi si otterrà il risultato finale.

```
private String ruoloFattoScorer;
public List<Giocatore> listaScorer(Integer spazioSalariale, String posizione){

    if(ruoloFattoScorer==null || !ruoloFattoScorer.contains(posizione)) {
    lscorer=giocatoriDao.getListaGiocatoriAccessibiliPunti(spazioSalariale,squadraSelezionata,posizione);

    Collections.sort(lscorer,new Comparator<Giocatore>(){

        @Override
        public int compare(Giocatore o1, Giocatore o2) {
            return -Float.compare(o1.getPesoScorer(), o2.getPesoScorer());
        }

    });
    ruoloFattoScorer=posizione;}
    return lscorer;
}
```

Il metodo sopra rappresentato chiarisce l'algoritmo per ottenere una lista di giocatori ordinata in base al tipo (in questo caso è stato riportato quello facente riferimento al tipo Scorer, ma è analogo per tutti).
Si richiama il metodo "getListaGiocatoriAccessibiliPunti", a cui viene fornito lo spazio salariale, la squadra selezionata dall'utente e un ruolo; in questo modo fornirà una lista di giocatori con un salario inferiore o pari allo spazio salariale disponibile, non appartenenti alla propria squadra e che giochino nella posizione richiesta (si tenga conto che i giocatori con stipendio pari al minimo salariale possono essere ingaggiati e sono quindi considerati anche se hanno un compenso maggiore di quello che è lo spazio salariale rimasto, come spiegato prima).
Si può vedere la presenza della variabile "ruoloFattoScorer" che permette di memorizzare il ruolo a cui fa riferimento la lista, in questa maniera, qualora venga richiesta un'altra lista di Scorer per il medesimo ruolo, non sarà interrogato nuovamente il database ma sarà fornita la lista già presente, e sarà poi compito della ricorsione scegliere i giocatori con un salario adatto.
In "getListaGiocatoriAccessibiliPunti" viene anche impostato un peso in base all'età del giocatore e al suo stipendio. Inizialmente è posto uguale alla media del giocatore nella categoria considerata (in questo caso alla media punti del giocatore, se si parlasse invece di assistman allora si considererebbe la sua media assist e così via).

```
if(g.getEta()<25) {
    g.setPesoScorer(g.getPesoScorer()+mediaPunti/2);
}
if(g.getEta()>30) {
    g.setPesoScorer(g.getPesoScorer()-mediaPunti/2);
}
if(g.getSalary()<mediaSalary) {
    g.setPesoScorer(g.getPesoScorer()+mediaPunti/2);
}
if(g.getSalary()>20*Math.pow(10, 6)) {
    g.setPesoScorer(g.getPesoScorer()-mediaPunti/2);
}
```

Media punti e salary indicano la media punti e salariale di tutti i cestisti della lega.

Un altro algoritmo fondamentale del programma permette invece di individuare quali giocatori della propria squadra sia necessario cedere per potersi permettere di ingaggiare nuovi cestisti, selezionati dall'utente. Viene passato come parametro una lista di giocatori,

ovvero quelli che l'utente ha selezionato come incedibili, e che non verranno quindi considerati nelle possibili soluzioni.

```
public List<List<Giocatore>> trovaPossibilita(List<Giocatore> incedibili) {
    List <Giocatore> tot=getRoster(getSquadraSelezionata());
    Integer numeroGiocatoriARoster=tot.size();
    for(Giocatore g:incedibili) {
        if(tot.contains(g))
            tot.remove(g);
    }
    Double costo=(double)getSalaryCapDegliAcquisti();
    Integer livelloSalariale=getSalaryCap(getSquadraSelezionata());
    Integer spazioRosterMinimo=numeroGiocatoriARoster+daAcquistare.size()-17;
    if(spazioRosterMinimo<0)
        spazioRosterMinimo=0;
    Integer spazioRosterMassimo=numeroGiocatoriARoster+daAcquistare.size()-12;
    if(spazioRosterMassimo>3)
        spazioRosterMassimo=3;
    if((livelloSalariale+costo<=this.limiteSalariale && spazioRosterMinimo==0) || (costo==0 && spazioRosterMinimo==0)) {
        return null;}

    else {
        if(livelloSalariale>=this.limiteSalariale) {
            costo=costo*100/125;
        }
        if(livelloSalariale<limiteSalariale) {
            costo=(livelloSalariale+costo)-limiteSalariale;
            costo=costo*100/125;
        }
        List<Giocatore> parziale=new ArrayList<Giocatore>();


       List<List<Giocatore>> results=new ArrayList<List<Giocatore>>();
       faiRicorsioneAcquisti(parziale,tot,0,spazioRosterMassimo,spazioRosterMinimo,costo,0,results);

       return results;
    }
}
```

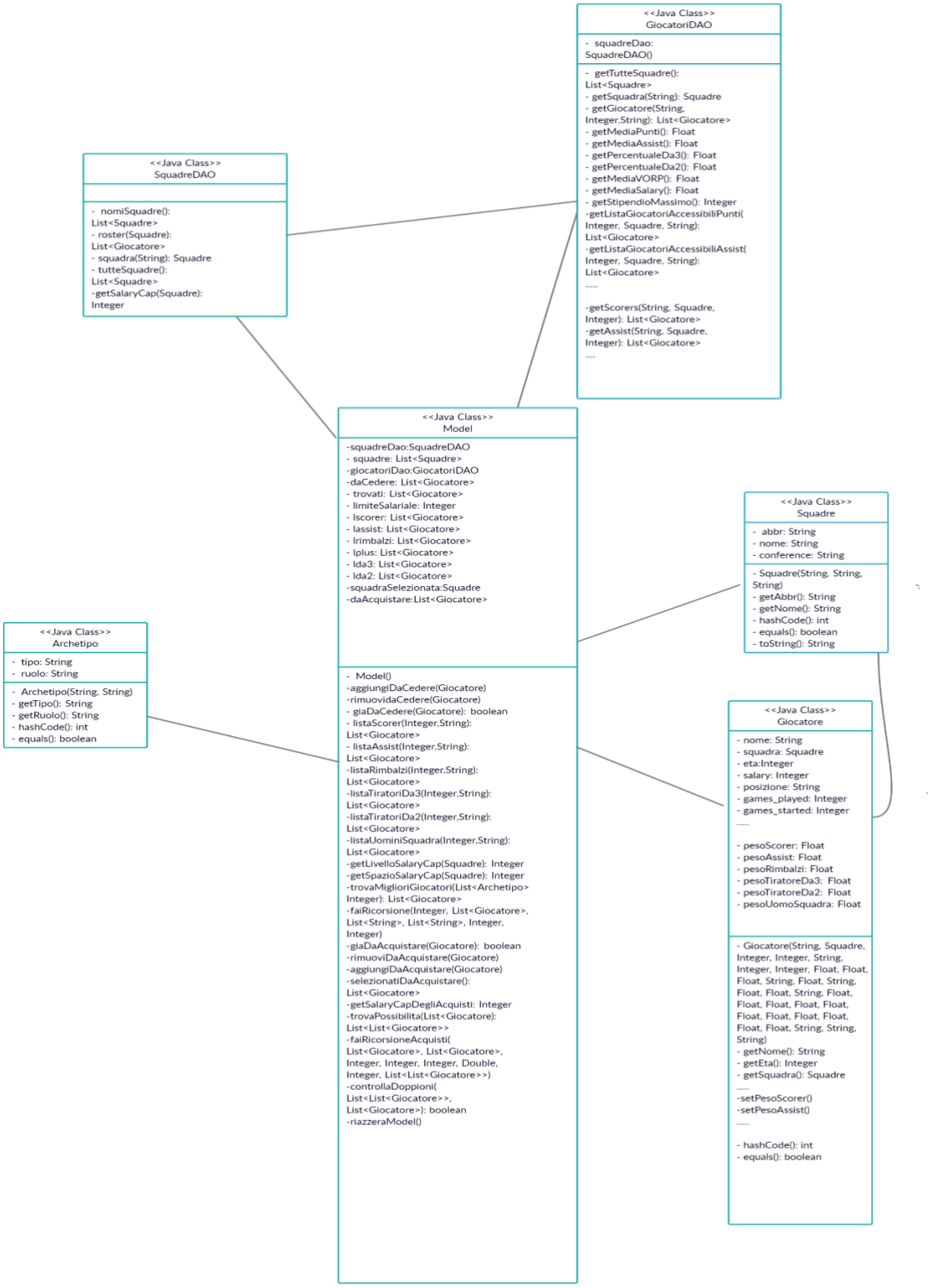
```
private void faiRicorsioneAcquisti(List<Giocatore> parziale, List<Giocatore> tot,Integer livello,Integer nMax,Integer nMin,Double costo,Integer liberato
    if(nMax==livello) {
        if(liberato>=costo && nMin<=livello) {
        if(controllaDoppioni(results,parziale)==false) {

            results.add(new ArrayList<Giocatore>(parziale));
            return;
        }
        else
            return;
     }
        return;
    }

    if(liberato>=costo && livello>=nMin) {
        if(nMax>=livello) {
        if(controllaDoppioni(results,parziale)==false) {
             results.add(new ArrayList<Giocatore>(parziale));
             return; //per evitare ad esempio mi dia curry,curry+minimo,curry+minimo+minimo
         }
         }
    }


    for(Giocatore g:tot) {
        if(!parziale.contains(g)) {
            parziale.add(g);
            faiRicorsioneAcquisti(parziale,tot,livello+1,nMax,nMin,costo,liberato+g.getSalary(),results);
            parziale.remove(g);
        }
    }
```

Come risultato l'algoritmo presenta un insieme di liste di giocatori, in modo da poter esplorare tutte le diverse possibilità.
Tale risultato è ottenuto grazie al metodo "faiRicorsioneAcquisti", il quale, basandosi su un approccio ricorsivo, permette di ricavare le possibili combinazioni.
Si può vedere come questo metodo richiami a sua volta "controllaDoppioni", per evitare che vengano fornite diverse sequenze ma equivalenti nella pratica (ad esempio se è sufficiente cedere il giocatore A, ovviamente risulterà adeguato anche cedere A+B, e A+B+C e così via).

Qualora non vi sia bisogno di cedere propri giocatori, poiché la squadra può permettersi i salari degli atleti desiderati senza superare il salary cap, o perché questi guadagnano il minimo salariale, viene segnalato in output all'utente.
Inoltre viene sempre posta l'attenzione al fatto che il roster del team debba essere composto da un numero compreso fra 12 e 17 atleti.

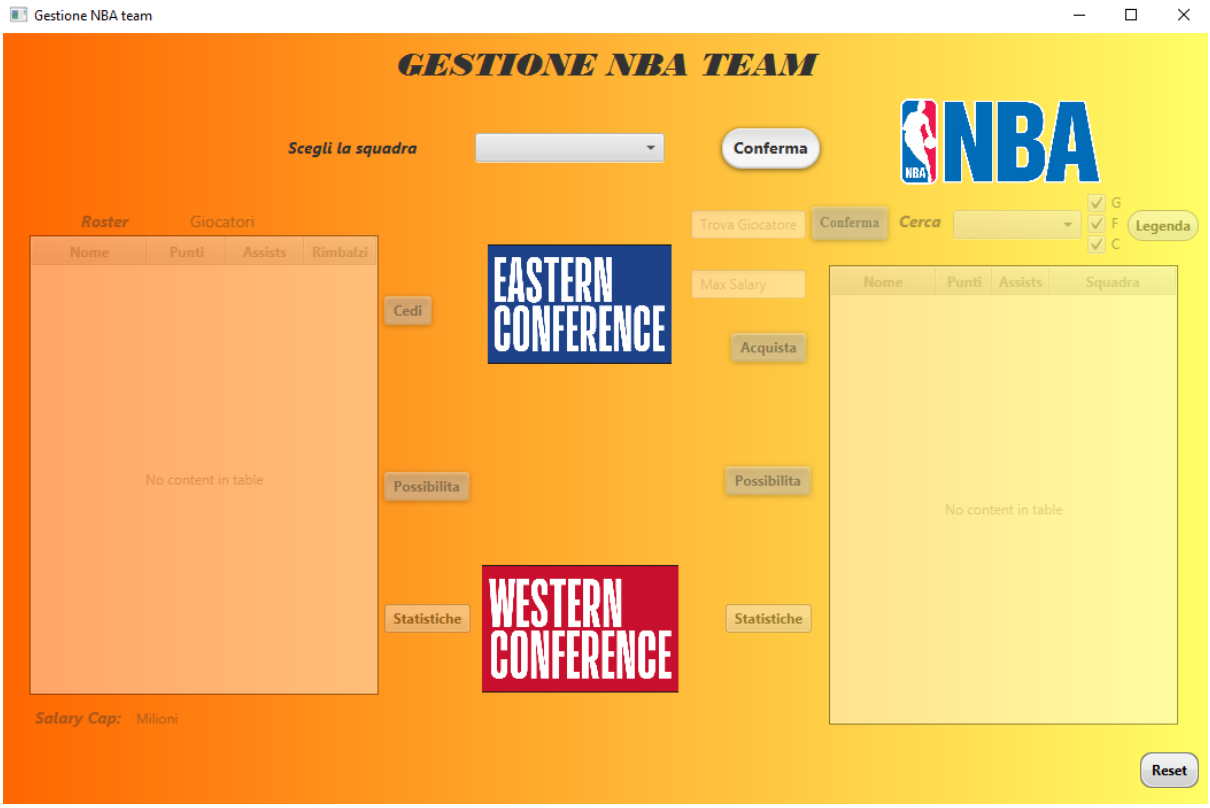
5. DIAGRAMMA DELLE CLASSI PRINCIPALI

Nel seguente diagramma delle classi più legate alla parte algoritmica sono stati omessi alcuni semplici metodi di setter e getter.

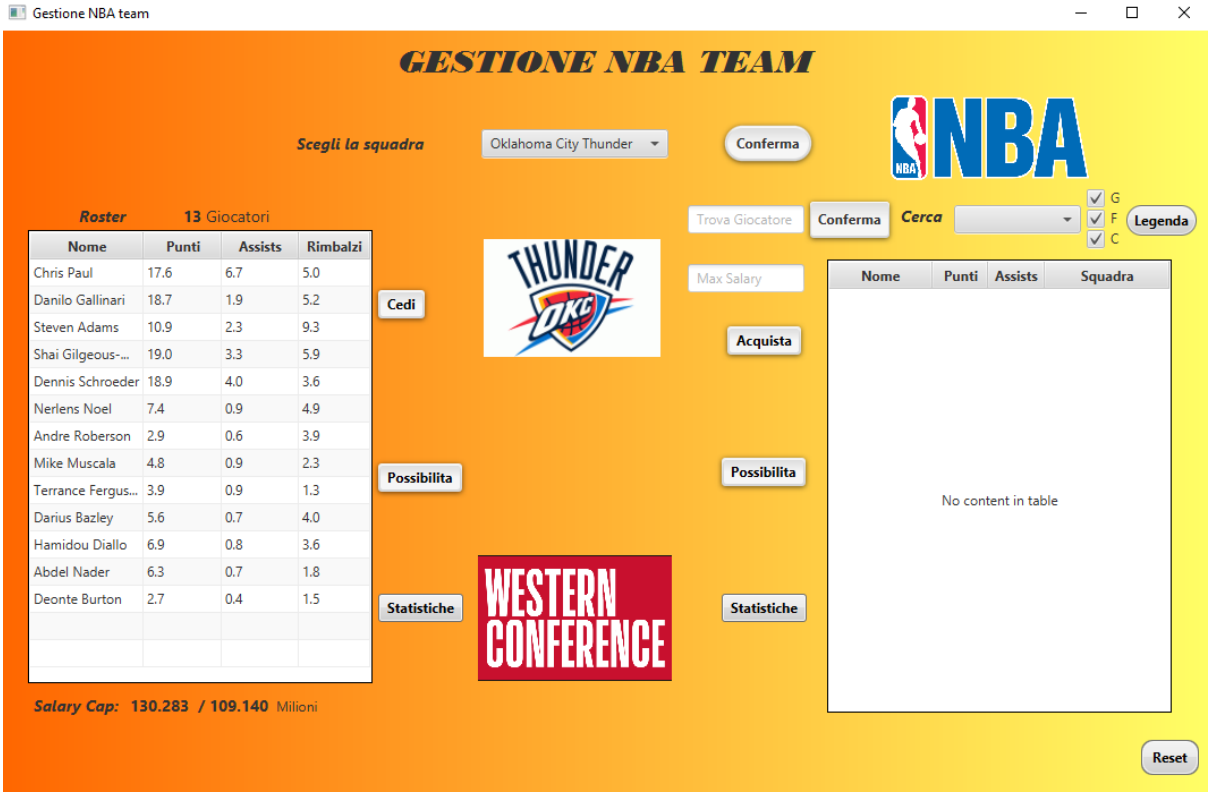


6. VIDEATE DELL'APPLICAZIONE

All'avvio dell'applicazione si apre la seguente schermata dove poter scegliere la propria squadra.

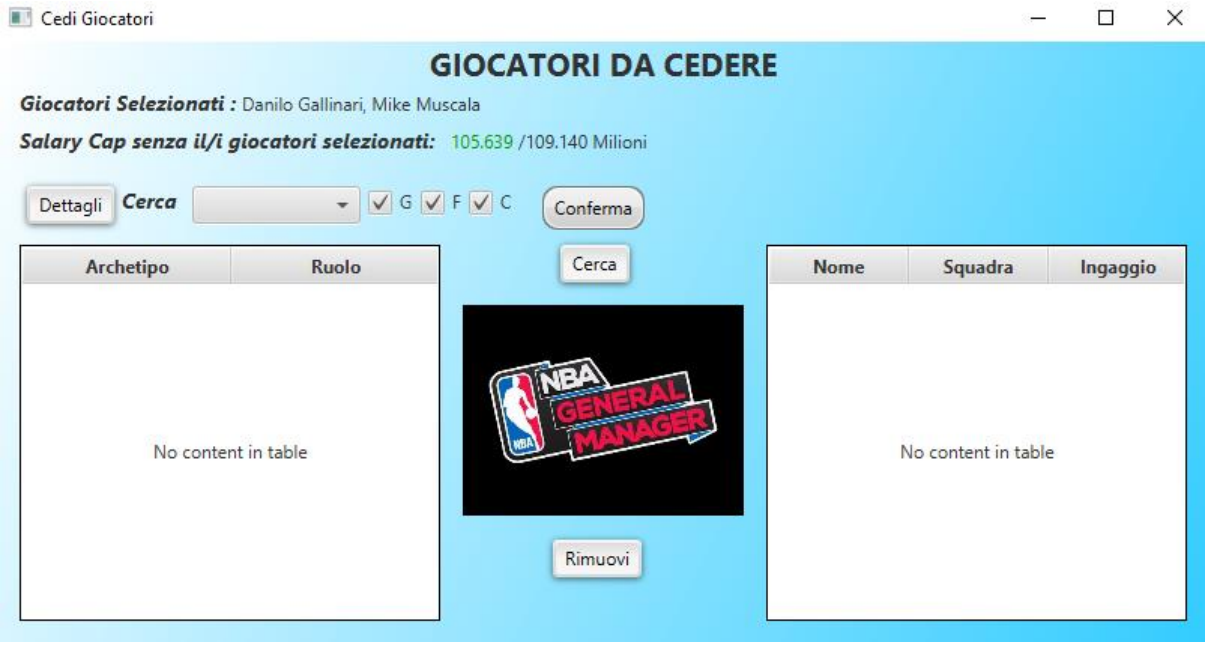


Dopo la scelta sulla sinistra viene mostrato il roster e la situazione salariale del team.

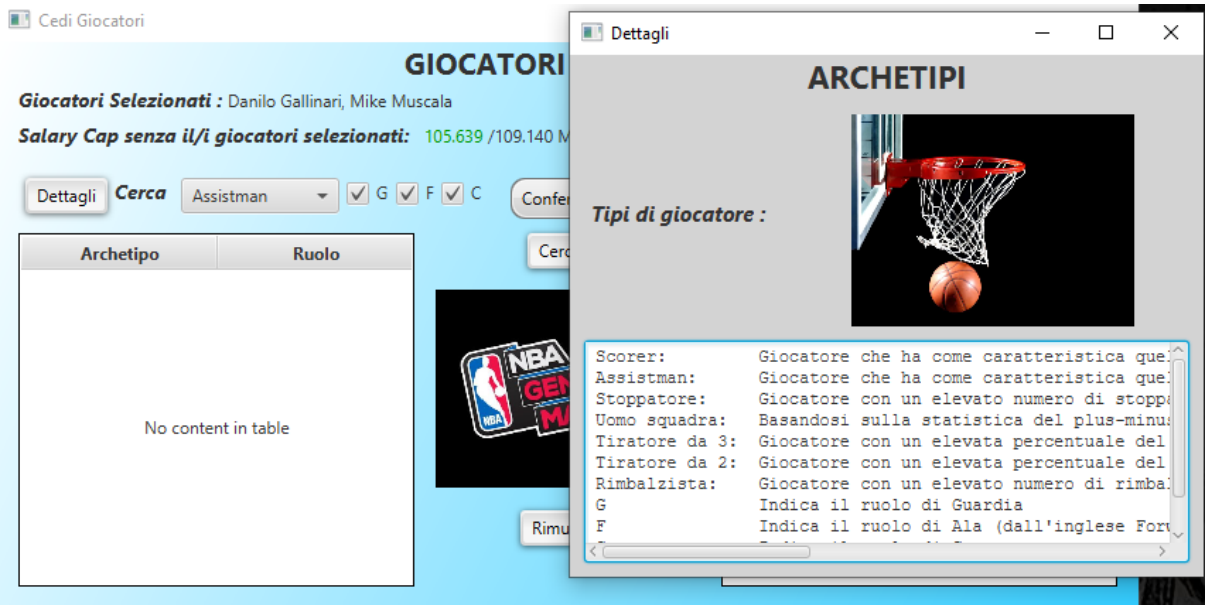


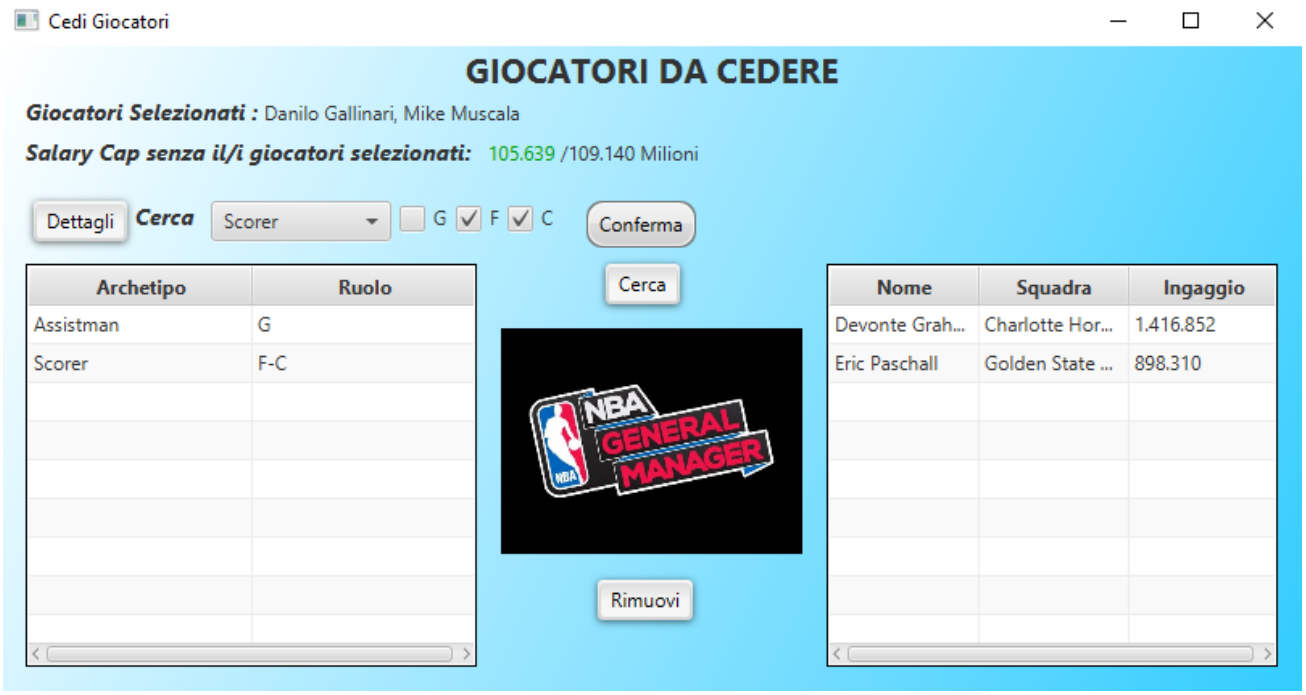
A questo punto è possibile selezionare alcuni giocatori della propria squadra da cedere, aggiungerli alla lista tramite “Cedi”, e successivamente, premendo il pulsante “Possibilita”, andare a trovare con chi sostituirli. Non è possibile selezionare più di tre giocatori da cedere, anche perché ciò sarebbe irrealistico.

Si apre quindi una nuova schermata dove poter selezionare i parametri dei giocatori da cercare.

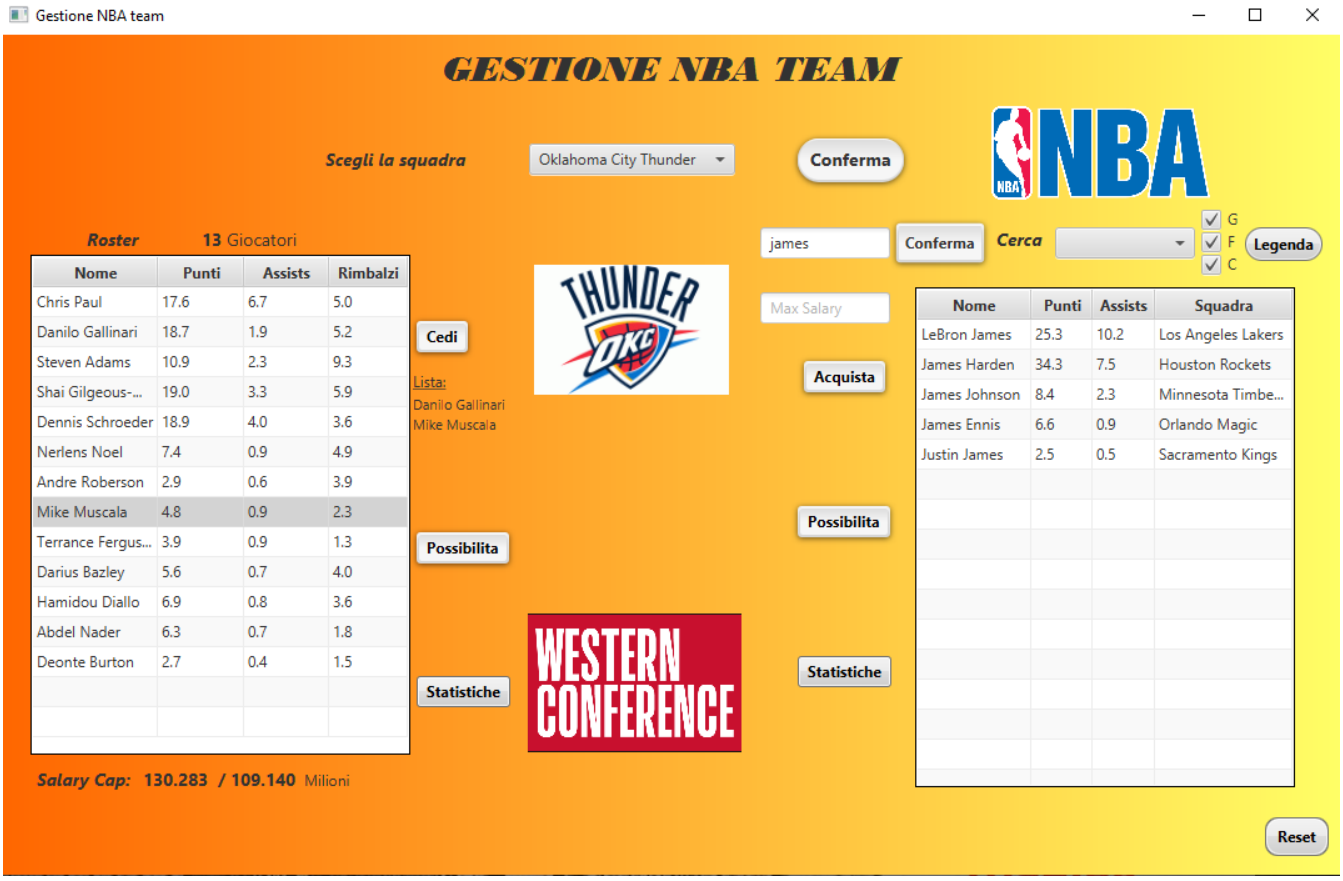


Si possono selezionare fino a tre diversi archetipi, dopo di che, selezionando “Cerca” viene generata dal computer in output la miglior lista dei giocatori secondo i parametri di input e rispettando il salary cap. Nel caso non si abbia conoscenza del significato di alcuni termini (es. Scorer, Assistman o anche i ruoli G,F,C), cliccando su “Dettagli” si aprirà una nuova finestra dove questi sono spiegati.

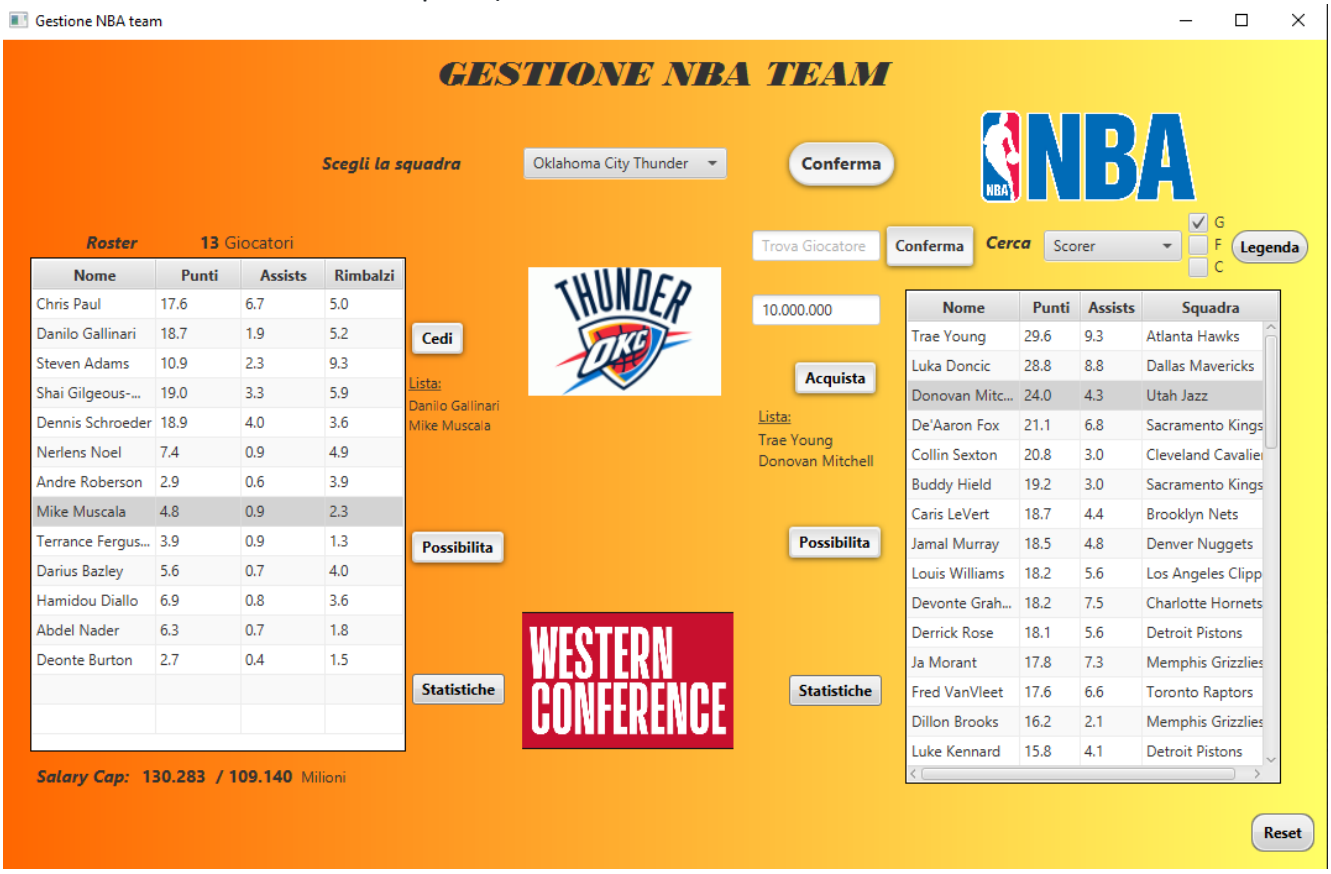




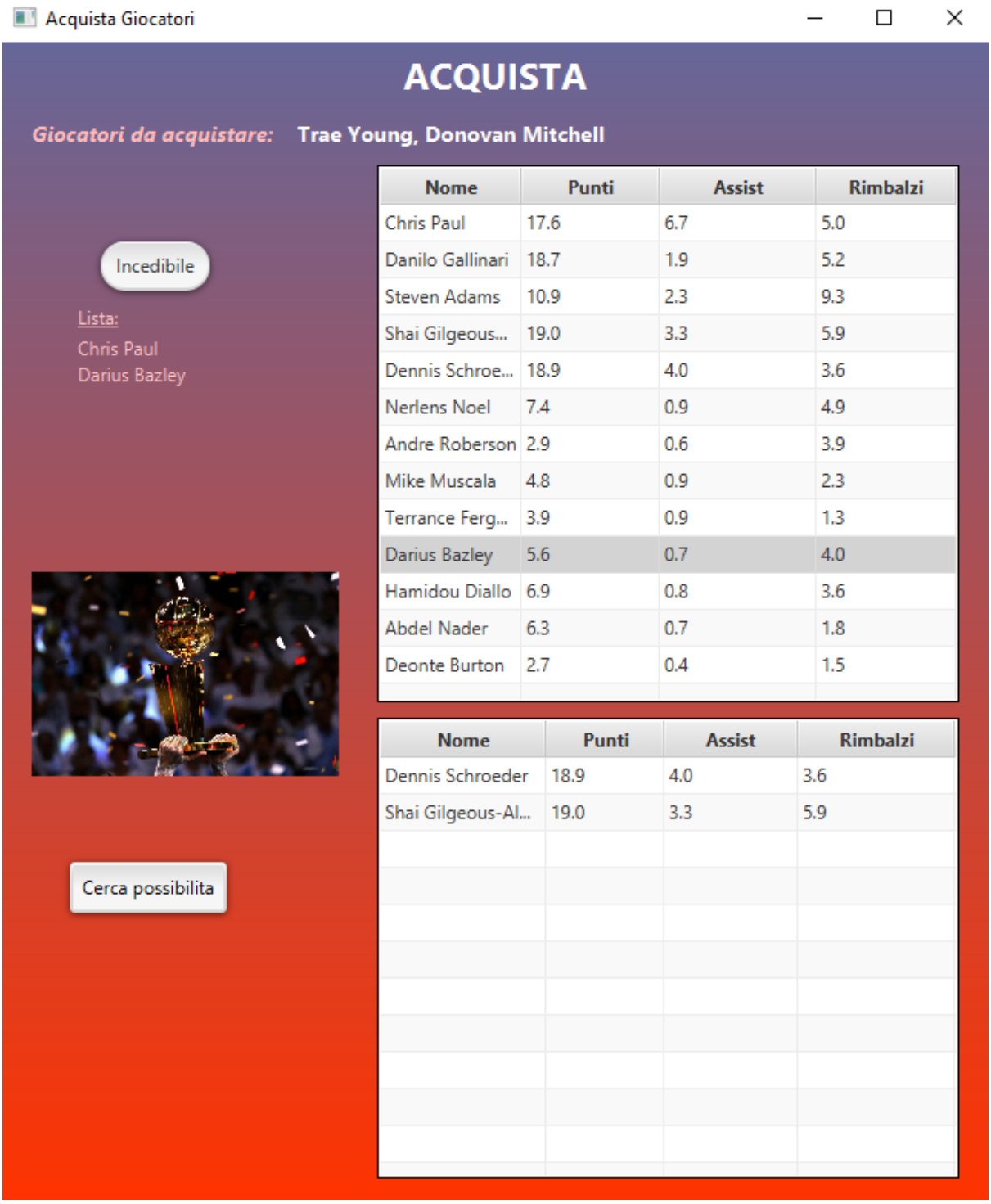
Tornando alla schermata principale, si può vedere come sulla destra sia possibile effettuare varie ricerche.
È possibile semplicemente cercare un giocatore per nome, oppure impostando anche un salario massimo e il ruolo.



Se non si vuole cercare inserendo il nome, è possibile selezionare la categoria di giocatore che si desidera, e sarà presentata in output una lista di cestisti, ordinati in maniera decrescente secondo la statistica corrispondente (es. se si seleziona Scorer, i giocatori saranno ordinati in base ai punti).



Si possono selezionare fino a tre dei giocatori presentati e aggiungerli alla lista degli acquisti, come fatto nella videata sopra.
A questo punto premendo su “Possibilita” si conferma la lista e si apre una nuova schermata.



In alto è possibile selezionare alcuni dei propri giocatori aggiungendoli alla lista degli incedibili: in questa maniera, quando si va a selezionare “Cerca possibilita”, l’applicativo presenterà in output le possibili combinazioni di cestiti da cedere, senza però introdurre quelli che sono stati inseriti nella lista degli incedibili.
Nel caso si voglia una combinazione diversa di atleti da cedere, basta premere di nuovo “Cerca possibilita” e, se presente, ne verrà presentata una diversa.

Link al video Youtube: https://youtu.be/LlAYycsrcpQ

7. RISULTATI SPERIMENTALI OTTENUTI

METODO = CEDI	SELEZIONATI	SPAZIO SALARIALE OCCUPATO(IN MILIONI)	TEMPO (s)
1)	Assistman/G-F-C Tiratore da 2/G-F-C Uomo Squadra/G-F-C	69.252	1.217
2)	Assistman/G-F Tiratore da 2/C Uomo Squadra/G	69.252	0.345
3)	Scorer/G-F-C Rimbalzista/G-F-C Tiratore da 3/G-F-C	27.175	1.248
4)	Assistman/G-F	69.252	0.304
5)	Scorer/G-F-C Rimbalzista/G-F-C Tiratore da 3/G-F-C	131.645	0.207
6)	Assistman/G-F-C	131.645	0.063
7)	Uomo Squadra/G-F-C Tiratore da 2/G-C	105.707	0.205
8)	Uomo Squadra/G-F-C Tiratore da 2/G-C Scorer/G-F-C	105.707	0.242
9)	Rimbalzista/G-F-C Tiratore da 2/G-F-C Tiratore da 3/G-F-C Assistman/G-F-C Scorer/G-F-C Scorer/G-F-C Uomo Squadra/G-F-C Rimbalzista/G-F-C	50.897	1.811

METODO= ACQUISTA	SOMMA SALARI DA ACQUISTARE(Milioni)	SALARY CAP SQUADRA (Milioni)	NUMERO INCEDIBILI	SOMMA SALARI DEGLI INCEDIBILI (Milioni)	COMBINAZIONI POSSIBILI	TEMPO (s)
1)	38.199	103.702	0	0	30	0.258
2)	38.199	103.702	1	34.45	24	0.037
3)	38.199	103.702	5	48.397	16	0.021
4)	68.02	140.102	0	0	20	0.023
5)	68.02	140.102	3	48.792	0	0.021
6)	68.02	140.102	5	84.768	0	0.02
7)	5.038	112.713	0	0	67	0.323
8)	93.862	112.713	2	5.63	0	0.018
9)	5.038	76.405	5	30.657	Non necessario	0.018

Nelle tabelle qui sopra sono stati riportati i risultati solo per i due metodi principali che sono quelli per acquistare e cedere giocatori. I tempi di esecuzione per gli altri sono immediati.

8. CONCLUSIONI

L’applicativo risulta di facile utilizzo ed intuitivo. I risultati ottenuti sono soddisfacenti, anche se per evitare che il costo computazionale diventasse troppo oneroso, si è introdotto un limite di giocatori da cedere/acquistare (non più di tre cestisti).
Il tool si caratterizza per la sua utilità e adattabilità in casi pratici: l’app può risultare utile anche per un appassionato di videogiochi che si cimenti nello storico “NBA 2K”, trovandosi così a vestire i panni del general manager e dovendo guidare la propria squadra al successo.

Mi ritengo molto soddisfatto per aver avuto la possibilità di lavorare a questo progetto unendo l’aspetto della programmazione (argomento che più mi ha appassionato in questi tre anni di università) con la mia grande passione per sport, ed in particolare per la pallacanestro.