Politecnico di Torino

Corso di Laurea in Ingegneria Gestionale
Classe L-8
A.A. 2022/2023
Sessione di Laurea Ottobre 2023

Tesi di Laurea Triennale

Analisi statistiche UEFA Champions League 2021-22 (per singolo giocatore e per singola squadra) e creazione di un dream team

Relatore:
Professor Fulvio Corno

Candidato:
Alessio Vantaggi, 282425

Sommario

1 - Proposta di progetto

1.1 Studente proponente

282425 Vantaggi Alessio

1.2 Titolo della proposta

Analisi statistiche UEFA Champions League 2021-22 (per singolo giocatore e per singola squadra) e creazione di un *dream team*.

1.3 Descrizione del problema proposto

L'analisi e l'interpretazione dei dati riveste un ruolo molto importante. Infatti, oltre che ad essere immagazzinati, i dati devono essere anche contestualizzati ed interpretati. Il principale obiettivo è quello di facilitare l'interpretazione dei dati, mediante confronti e organizzazione in ranking, in modo da favorire una corretta visualizzazione. Grazie alla normalizzazione delle statistiche e alla suddivisione in ranking è possibile evidenziare punti di forza e punti di debolezza di ciascun giocatore e di ciascuna squadra.
È possibile infine richiedere la creazione di un *dream team*, che contempli le opzioni scelte dall'utente.

1.4 Descrizione della rilevanza gestionale del problema

Dal punto di vista gestionale l'analisi dei dati riveste un ruolo cruciale, in quanto grazie ad essi si è in grado di fare delle previsioni, ma anche delle scelte per l'organizzazione futura. Lo studio delle performance della passata stagione aiuta anche a capire quali siano stati i punti di forza (e di debolezza) di una determinata squadra o di un singolo giocatore. In questo modo sarà possibile intervenire, in maniera mirata, sulla rosa e/o intensificare l'allenamento per una determinata caratteristica di un calciatore. La creazione di un *dream team* permette, infine, di stabilire una sorta di benchmark da seguire come riferimento.

1.5 Descrizione dei data-set per la valutazione

Il data-set utilizzato per le statistiche dei giocatori è il seguente:
https://www.kaggle.com/datasets/azminetoushikwasi/ucl-202122-uefa-champions-league
che dovrà essere incrociato con il seguente data-set, relativo al salario a inizio stagione (2022) del calciatore:
https://www.kaggle.com/datasets/bryanb/fifa-player-stats-database?select=FIFA22_official_data.csv

Infine si utilizzerà il seguente data-set per estrarre il valore economico del giocatore, raggiunto alla fine della competizione:

https://www.kaggle.com/datasets/bryanb/fifa-player-stats-database?select=FIFA23_official_data.csv

Il **database** completo sarà così organizzato:

- Una tabella con tutti i **giocatori presenti nel data-set**:
  (**IDGiocatore**, Name, position, #Age (2022), Nationality, Wage (2022), Value (2023))
- Una tabella con le **statistiche**:
  (**IDGiocatore**, **club**, #saved, #conceded, #cleansheets, #balls_recoverd, #tackles, #t_won, #pass_attempted, #pass_completed, #assist, #total_attempts, #on_target, #goals, #dribbles, #minutes_played, #match_played)
- Una tabella con il **numero di partite giocate da una squadra**:
  (**club**, #total_match_played)

1.6 Descrizione preliminare degli algoritmi coinvolti

Lo scopo del programma è quello di mostrare statistiche *normalizzate* di un determinato giocatore o di una determinata squadra, mediante un algoritmo di ricerca ed elaborazione del ranking finale, che permetterà di visualizzare le statistiche, sia a livello numerico, che percentuale. La percentuale sarà poi usata per stilare una classifica in modo da comprendere i punti di forza e di debolezza del giocatore selezionato dall'utente. Se invece viene selezionata una squadra, verranno mostrate le statistiche complessive di quel team,

ottenute facendo la media dei calciatori appartenenti al club e la relativa posizione nella classifica generale, per ogni specifica caratteristica.
Grazie a questa vista si potranno evidenziare punti di forza e di debolezza per ogni reparto della squadra selezionata.
L'ultimo algoritmo riguarda la creazione di una sorta di *dream team*, che verrà stabilito dal sistema sulla base delle opzioni fornite dall'utente ed implementato mediante una *ricorsione*.

1.7 Descrizione preliminare delle funzionalità previste per l'applicazione software

L'applicazione finale dovrà essere composta di tre funzioni principali:

1 - La prima funzione è caratterizzata da un menù a tendina contenente tutti i giocatori presenti all'interno del data-set. Una volta selezionato un singolo giocatore, sarà possibile visionare tutte le caratteristiche relative a quel calciatore. Le statistiche differiranno a seconda che si tratti di un portiere o di un giocatore di movimento.

- Per quanto riguarda i portieri verranno visualizzati: saved, saved / (conceded + saved), conceded, conceded / minutes played, cleansheets.

- Se si tratta invece di un giocatore di movimento le statistiche verranno divise in tre sezioni:
  - Difensive → verranno visualizzati: balls recoverd, balls recoverd / minutes played, tackles, tackles won, tackles won / tackles.
  - Passaggi → verranno visualizzati: pass attempted, pass completed, pass completed / pass attempted, assist, assist / minutes played.
  - Offensive → verranno visualizzati: shot attempts, shot on target, shot on target / shot attempts, goals, goals / shot attempts, goals / minutes played, dribbles, dribbles / minutes played.

2 - La seconda funzione è caratterizzata da un menù a tendina, contenente tutte le squadre che hanno partecipato alla Uefa Champions League 2021-22. Una volta selezionata una squadra, sarà possibile visionare tutte le caratteristiche/statistiche relative all'intero team, che saranno suddivise in reparti:
  - Portieri → verranno visualizzati: saved, saved / (conceded + saved), conceded, conceded / match played, cleansheets.
  - Difensive → verranno visualizzati: balls recoverd, balls recoverd / match played, tackles, tackles won, tackles won / tackles.
  - Passaggi → verranno visualizzati: pass attempted, pass completed, pass completed / pass attempted, assist, assist / match played.
  - Offensive → verranno visualizzati: shot attempts, shot on target, shot on target / shot attempts, goals, goals / shot attempts, goals / match played, dribbles, dribbles / match played.

Infine, per ogni statistica è presente anche la relativa posizione nella classifica generale di tutte le squadre che hanno partecipato alla competizione.

3 – La terza funzione consiste nella creazione di un *dream team* di 11 giocatori (1 portiere e 10 giocatori di movimento) per una determinata caratteristica/statistica, selezionata dall'utente, con dei limiti imposti dall'utente.

Per prima cosa verrà proposto all'utente la scelta del modulo di gioco, tra quelli di seguito disponibili:
- 3-4-3
- 3-5-2
- 4-3-3
- 4-4-2
- 4-5-1
- 5-3-2
- 5-4-1

A questo punto l'utente dovrà effettuare la scelta della caratteristica/statistica di riferimento per il portiere, tra quelle di seguito disponibili:

- saved / (conceded + saved)
- conceded / minutes played
- clean sheets / match played

Dopo di che l'utente dovrà effettuare la scelta della caratteristica/statistica di riferimento per i 10 giocatori di movimento. Ecco di seguito quelle disponibili:

- balls recoverd / minutes played
- tackles won / tackles
- pass completed / pass attempted
- assist / minutes played
- shot on target / shot attempts
- goals / shot attempts
- goals / minutes played
- dribbles / minutes played

Successivamente l'utente dovrà inserire:

1. il budget massimo complessivo (per *value* e *wage*) della squadra
2. la percentuale di salario e di valore economico dedicate al portiere (il resto del budget sarà dedicato ai restanti 10 giocatori di movimento).
3. numero minimo di minuti giocati (il valore inserito dovrà essere ≥ 540 min)
4. un'età minima dei giocatori
5. un'età massima dei giocatori
6. numero massimo di giocatori appartenenti alla stessa squadra (1, 2, 3)

Oltre alla ricerca mediante l'inserimento di un budget massimo (per salario e per valore economico) è presente un'altra opzione, che permette di ottenere il *dream team* senza i vincoli sopra indicati ai punti 1, 2, 4, 5, 6, mentre per il punto 3 il vincolo è prefissato (e non modificabile) al valore minimo, di default, di 540 minuti.

2 - Descrizione dettagliata del problema affrontato

Al giorno d'oggi, in un'epoca sempre più tecnologica, la raccolta dei dati riveste un ruolo fondamentale.
I dati, però, oltre ad essere immagazzinati, devono essere anche elaborati e contestualizzati, per renderli confrontabili tra di loro e per favorirne, in questo modo, la comprensione, la valutazione e la visualizzazione.
Proprio per questo motivo, ho deciso di realizzare la mia applicazione, che fosse in grado di raccogliere tutti i dati disponibili della UEFA Champions League 2021-2022, di trasformarli e renderli più immediati all'utente finale, per favorire eventuali valutazioni e considerazioni future.
La prima caratteristica implementata, all'interno dell'applicazione, è stata la normalizzazione dei dati: in questo modo ogni statistica del giocatore viene divisa per il numero di minuti giocati, ovvero per il numero totale di tentativi effettuati, ovvero per il numero di partite effettuate. In questo modo è stato possibile confrontare i giocatori o le squadre, all'interno del dataset, che avevano avuto minutaggi differenti tra di loro.
Prendendo spunto dall'applicazione realizzata possiamo vedere come il giocatore: "Alaba" della squadra Real Madrid CF sia al 21º per *balls recoverd*, ma solo al 293º per *balls recoverd/minutes played*.
Questa discordanza viene ben spiegata dal fatto che il giocatore abbia preso parte alla quasi totalità delle partite disputate dal suo team.
La classifica così stilata, con le caratteristiche normalizzate, ci permette di analizzare meglio l'andamento dei giocatori e delle squadre.
All'interno dell'applicazione, oltre alle statistiche normalizzate, viene presentato anche il relativo ranking, per ogni caratteristica, sia normalizzata che non normalizzata, un ulteriore elemento che favorisce la comprensione e la giusta collocazione del giocatore; proprio per questo, nell'output delle caratteristiche viene specificato anche il numero totale di giocatori presenti nel database.

La creazione del **dream team con vincoli**, infine, permette di stabilire una sorta di benchmark di riferimento. Infatti, dopo aver completato i campi *budget totale value* e *budget totale wage* è possibile rendersi conto a che livelli massimi si può arrivare, in determinate statistiche, sfruttando quei budget a disposizione.
In ultimo la creazione del **dream team senza vincoli** permette di scovare i giocatori migliori, in ogni ruolo, per la statistica selezionata.

Per quanto riguarda l'utilizzo concreto dell'applicazione, quest'ultima potrebbe tornare utile a molte società calcistiche a partire dalla dirigenza, passando per i direttori sportivi, gli allenatori ed arrivando infine ai calciatori.

Se prendiamo in considerazione gli allenatori, questi ultimi potrebbero selezionare dal menù a tendina la propria squadra ed analizzare l'andamento, valutando così quale reparto è stato il migliore e quale reparto è stato il peggiore. In questo modo è possibile stilare una scaletta prestabilita per colmare le lacune presentate durante la partecipazione alla competizione europea.
Inoltre gli allenatori potrebbero selezionare anche le statistiche relative ad una squadra avversaria, per scovare eventuali punti deboli o tentare di studiare una strategia, che riesca a neutralizzare le caratteristiche migliori dell'altro team.

Passando ai direttori sportivi ed alla dirigenza, una volta analizzate le carenze della squadra, potrebbero intervenire in maniera mirata sulla rosa. Mediante il **dream team senza vincoli**, una volta selezionato il modulo e le caratteristiche, una per il portiere e una per i giocatori di movimento, vengono mostrati i migliori giocatori per quel fondamentale scelto, divisi per ruolo. Nella visualizzazione vengono riportati anche i corrispettivi valori per *FIFA value* e *FIFA wage*, valutando così l'onerosità di ciascuna operazione.
Grazie al **dream team con vincoli** è infine possibile anche inserire dei vincoli sul budget, sul numero di minuti giocati dai calciatori che dovranno essere presi in considerazione, sull'età ed infine sul numero massimo di calciatori appartenenti alla stessa squadra, in modo da comprendere quale sia il livello massimo raggiungibile, in quella caratteristica, con quegli specifici investimenti.

Le statistiche dei singoli giocatori potrebbero essere studiate anche dai giocatori stessi o dagli allenatori, per cercare di migliorarsi nelle statistiche peggiori, intensificando il lavoro nelle loro lacune, o trovando uno schieramento tattico (da parte dell'allenatore) che esalti al meglio le qualità della rosa a disposizione.
Le statistiche riguardanti i calciatori potrebbero essere usate anche dai direttori sportivi, per monitorare l'andamento dei calciatori da loro seguiti, oppure per fare una selezione dei profili migliori e più adeguati a ogni determinata posizione.

Invece, per quanto riguarda le criticità dell'applicazione, esse possono essere riscontrate all'interno dell'algoritmo ricorsivo, per la creazione del **dream team con vincoli**: questo perché la ricorsione è computazionalmente molto onerosa e per cercare di ottenere dei risultati in tempi ragionevoli è stato necessario imporre vincoli molto stringenti per la creazione del dream team.

3 - Descrizione del data-set utilizzato per l'analisi

Il dataset utilizzato per la realizzazione del progetto è stato creato incrociando tra di loro i dati estrapolati da 8 file CSV, tutti provenienti dalla piattaforma online Kaggle, ed in alcuni casi cercando e inserendo a mano i dati necessari e/o mancanti.

Per ogni file CSV sono stati estrapolati solo i valori strettamente necessari alla realizzazione del programma, andando così a snellire, in maniere significativa, il database. Alcuni record, inoltre, sono stati modificati e/o ampliati per garantire una corretta ed immediata visualizzazione ed interpretazione del dato.

Mediante questo processo è stato possibile creare un unico database denominato: "**statisticheucl_2021-2022**", che contenesse al suo interno tutte le tabelle, in questo caso 3, necessarie per la raccolta dati.

Le tabelle così originate sono le seguenti:

3.1 Giocatori

In questa tabella sono raccolte tutte le informazioni principali riguardanti i calciatori che hanno partecipato, scendendo in campo per almeno 1 minuto di gioco, all'edizione 2021-2022 della UEFA Champions League.
Tutti i valori inseriti all'interno della tabella derivano interamente dai fogli CSV (reperiti su Kaggle) e si contano, in totale, 747 record.

#	Nome	Tipo di dati
1	**IDGiocatore**	**INT**
2	name	VARCHAR
3	position	VARCHAR
4	age	INT
5	nationality	VARCHAR
6	FIFA_wage_€	DOUBLE
7	FIFA_value_€	DOUBLE

Struttura tabella "giocatori"

Entrando nello specifico della struttura vera e propria della tabella, la **chiave primaria** è rappresentata da un campo numerico denominato: "IDGiocatore", una colonna con valori auto incrementali, in modo da evitare casi di omonimia tra i giocatori.
Le altre colonne che la compongono sono:

- name → riporta il cognome/nome del giocatore.
- position → rappresenta la posizione in campo occupata dallo specifico giocatore. Quelle disponibili sono: "Portiere", "Difensore", "Centrocampista", "Attaccante".
- age → mostra l'età del giocatore all'inizio della competizione.
- nationality → rappresenta la nazionalità del giocatore.
- FIFA_wage_€ → rappresenta il valore del salario del rispettivo giocatore nel videogioco di calcio FIFA 22.
- FIFA_value_€ → rappresenta il valore economico del rispettivo giocatore nel videogioco di calcio FIFA 23, perciò la quotazione raggiunta dopo la fine della competizione UEFA Champions League 2021-2022.

3.2 Squadre

In questa tabella sono raccolte tutte le informazioni principali riguardanti le squadre che hanno partecipato, dalla fase a gironi in poi, all'edizione 2021-2022 della UEFA Champions League.
Tutti i valori inseriti all'interno della tabella sono stati reperiti ed inseriti manualmente, andando a contare quante partite in totale avesse giocato ciascuna squadra. Si contano, in totale, 32 record.

#	Nome	Tipo di dati
1	**club**	**VARCHAR**
2	total_match_played	INT

Struttura tabella "squadre"

Entrando nello specifico della struttura vera e propria della tabella, la **chiave primaria** è rappresentata da un campo letterale denominato: "club", al cui interno sono contenuti tutti i nomi delle squadre. In questo caso non c'è rischio di omonimia tra squadre in quanto il nome di ogni team deve differire dagli altri.

L'altra colonna che la compone è: *total_match_played* → riporta il numero totale di match, disputati da ogni squadra, nell'edizione 2021-2022 della UEFA Champions League.

3.3 Statistiche

In questa tabella sono raccolte tutte le statistiche riguardanti i calciatori che hanno partecipato, scendendo in campo per almeno 1 minuto di gioco, all'edizione 2021-2022 della UEFA Champions League.
Tutti i valori inseriti all'interno della tabella derivano interamente dai fogli CSV (reperiti su Kaggle) e si contano, in totale, 747 record.

#	Nome	Tipo di dati
1	IDGiocatore	INT
2	club	VARCHAR
3	saved	INT
4	conceded	INT
5	cleansheets	INT
6	balls_recoverd	INT
7	tackles	INT
8	tackles_won	INT
9	pass_attempted	INT
10	pass_completed	INT
11	assist	INT
12	shot_attempts	INT
13	shot_on_target	INT
14	goals	INT
15	dribbles	INT
16	minutes_played	INT
17	match_played	INT

Struttura tabella "statistiche"

Dal momento che questa tabella raccoglie le statistiche sia dei portieri, che dei giocatori di movimento, è stato impostato a "NULL" il valore di completamento dei campi vuoti, in quanto un giocatore di movimento non potrà mai avere statistiche relative ad un portiere e viceversa.
Inoltre il valore di riempimento con NULL è stato utile anche nei casi in cui non fosse presente, all'interno dei file CSV, il valore per quel determinato campo o nei casi in cui la statistica fosse stata volutamente tralasciata, poiché uguale a 0.

Entrando nello specifico della struttura vera e propria della tabella, la **chiave primaria** è composta da due campi: "IDGiocatore" e "club": Questa scelta della chiave composta è stata fatta dal momento che un player potrebbe aver giocato in almeno due squadre differenti nell'arco della stessa competizione (UEFA Champions League 2021-2022).
Inoltre la colonna "IDGiocatore" è una chiave esterna che fa riferimento alla tabella: "giocatori"; mentre la colonna "club" è una chiave esterna che fa riferimento alla tabella: "squadre".

Le altre colonne che la compongono sono divise in tre gruppi:

➢ Campi esclusivi per i PORTIERI:
- saved → numero totale di tiri parati.
- conceded → numero totale di goal subiti.
- cleansheets → numero totale di partite senza subire goal.

➢ Campi esclusivi per i GIOCATORI DI MOVIMENTO:
- balls recoverd → numero totale di palloni recuperati.
- tackles → numero totale di contrasti tentati.
- tackles won → numero totale di contrasti vinti.
- pass attempted → numero totale di passaggi tentati.
- pass completed → numero totale di passaggi riusciti.
- shot attempts → numero totale di tiri tentati.
- shot on target → numero totale di tiri in porta tentati.
- dribbles → numero totale di dribbling riusciti.

➢ Campi comuni per PORTIERI e GIOCATORI DI MOVIMENTO:
- assist → numero totale di assist fatti.
- goals → numero totale di goal fatti.
- minutes played → numero totale di minuti giocati.
- match played → numero totale di partite giocate.

Le statistiche evidenziate si riferiscono a tutte le partite in cui il giocatore ha preso parte.

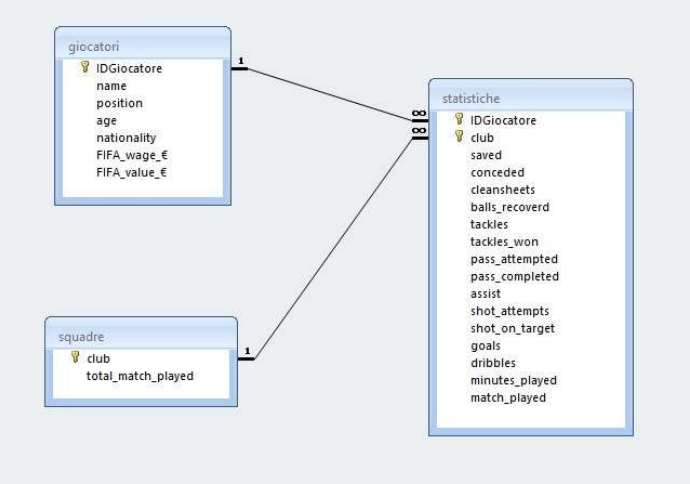


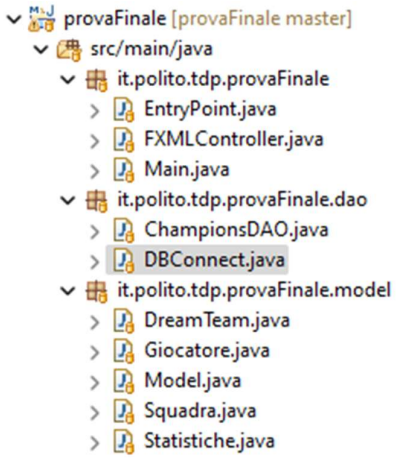
Diagramma E-R delle tabelle presenti nel database "statisticheucl_2021-2022"

Per quanto riguarda il riutilizzo e/o la ridistribuzione del database, dal momento che i file CSV di Kaggle sono accessibili gratuitamente presso la loro piattaforma, può essere ritenuta libera.

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

4.1 Strutture dati utilizzate

Il progetto è stato sviluppato in linguaggio Java, seguendo il pattern **MVC** (Model-View-Controller) ed il pattern **DAO** (Data-Access-Object), ottenendo così un codice modulare, più facile da gestire.
A tale scopo sono stati creati tre packages distinti, qui sotto elencati, che differiscono tra loro a seconda della loro funzione



4.1.1 it.polito.tdp.provaFinale

Questo package si occupa della gestione dell'interfaccia con l'utente, presentando la scena allo user agent e raccogliendo le sue interazioni.
È composto da 3 classi distinte:

1. EntryPoint: si occupa di caricare la scena, di inizializzare il controller per la gestione della scena, di inizializzare il model e di collegare il controller al model.
2. FXMLController: raccoglie tutte le interazioni dell'utente, le quali vengono inviate al model per ricevere una risposta specifica per il caso in questione. È responsabile dei cambiamenti di stato della scena.
3. Main: permette di avviare l'applicazione.

4.1.2 it.polito.tdp.provaFinale.dao

Questo package si occupa di stabilire una connessione con il database, permette perciò di passare i risultati ottenuti, mediante le query, al model. È stato implementato proprio per rispettare il pattern DAO, così da avere un acceso in lettura e in modifica dei dati limitato e controllato.
Si compone di 2 classi distinte:

1. ChampionsDAO: permette di estrarre dal database, mediante specifici algoritmi implementati, tutti i dati desiderati dall'utente per poi passarli al model.
2. DBConnect: permette di stabilire un canale di comunicazione (connessione) con il database.

4.1.3 it.polito.tdp.provaFinale.model

Questo package è il cuore del progetto in quanto tutte le elaborazioni dei dati vengono svolte qui; si occupa perciò della logica applicativa.
Si compone di 5 classi distinte:

1. Dream Team: una classe di appoggio ideata per facilitare la creazione del drem team, in quanto al suo interno ha tutti gli attributi inerenti e ricercati durante la creazione del dream team.
   Definisce così l'oggetto Dream Team ed ha il proprio costruttore: *Dream Team (name, position, club, age, fifaWage, fifaValue, minutesPlayed, statsName, statsValue).*
2. Giocatore: classe implementata per rispettare il pattern **ORM** (Object-Relational Mapping), corrisponde perciò alla tabella giocatori.
   Definisce l'oggetto Giocatore ed ha il proprio costruttore*: Giocatore (iDGiocatore, name, position, age, nationality, fifaWage, fifaValue)*.
   Ha i relativi getters per ogni attributo della classe, inoltre sono stati implementati i metodi: *hashCode ()*, *equals ()* e *toString ().*
3. Model: è la parte fondamentale del software, è in grado di gestire tutti i processi applicativi, dispone, infatti, di tutte le strutture di immagazzinamento dei dati e di tutti i metodi necessari per "rispondere" ad un input dell'utente. Si occupa perciò di fare da tramite tra l'applicazione e il database sottostante.
4. Squadra: classe implementata per rispettare il pattern **ORM** (Object-Relational Mapping), corrisponde perciò alla tabella squadre.

Definisce l'oggetto Squadra ed ha il proprio costruttore*: Squadra (club, totalMatchPlayed)*.
Ha i relativi getters per ogni attributo della classe, inoltre sono stati implementati i metodi: *hashCode ()*, *equals ()* e *toString ().*

5. Statistiche: classe implementata per rispettare il pattern **ORM** (Object-Relational Mapping), corrisponde perciò alla tabella statistiche.
   Definisce l'oggetto Squadra ed ha tre costruttori distinti tra loro:
   - Costruttore statistiche Giocatore: PORTIERE
   - Costruttore statistiche Giocatore: GIOCATORE DI MOVIMENTO
   - Costruttore statistiche Squadra

   Oltre ai campi presenti nella tabella, sono stati aggiunti anche tutti gli attributi relativi alle statistiche normalizzate e ai ranking di ogni singola statistica.

4.2 Algoritmi utilizzati

Gli algoritmi principali utilizzati per lo sviluppo dell'applicazione sono due: un *algoritmo di ricerca* per le statistiche dei giocatori, delle squadre (con relativa elaborazione del ranking finale) e per la creazione del dream team senza vincoli; un *algoritmo ricorsivo* per la creazione del dream team con vincoli.

4.2.1 Algoritmo di ricerca ed elaborazione del ranking finale

Per prima cosa sono stati popolati i menù a tendina relativi ai giocatori ed alle squadre, per permettere all'utente la scelta nella visualizzazione delle statistiche, relative ad un singolo giocatore o ad una singola squadra.

- Menù a tendina giocatori:
  ```
  String sql= "SELECT * FROM giocatori ORDER BY name";
  ```
  Questa istruzione mi permette di estrarre dal database tutti i giocatori presenti, ordinati in maniera crescente. Il valore risultante è stato poi passato, dal dao al model, che a sua volta lo ha reso disponibile al controller, il quale è stato così in grado di popolare il menù a tendina mediante questa istruzione:
  ```
  cmbPlayer.getItems().setAll(this.model.getAllPlayers());
  ```

- Menù a tendina squadre:
  ```
  String sql= "SELECT * FROM squadre ORDER BY club";
  ```
  Questa istruzione mi permette di estrarre dal database tutti le squadre presenti, ordinate in maniera crescente. Il valore risultante è stato poi passato, dal dao al model, che a sua volta lo ha reso disponibile al controller, il quale è stato così in grado di popolare il menù a tendina mediante questa istruzione:
  ```
  cmbClub.getItems().setAll(this.model.getAllTeams());
  ```

Una volta che l'utente avrà selezionato un giocatore (o una squadra) e schiacciato il relativo bottone delle statistiche, gli algoritmi forniranno al controller, sotto forma di stringa, l'output finale da mostrare all'utente. Nel caso in cui si trattasse di un giocatore viene effettuata una distinzione tra portiere e giocatore di movimento.

- Algoritmo di ricerca ed elaborazione del ranking finale statistiche PORTIERE:
```
public Statistiche getStatisticsGoalkeeper(int IDGiocatore) {
String sql= "SELECT tab.* FROM (SELECT IDGiocatore, club, saved, RANK() OVER (ORDER BY saved
desc) AS s, saved/(conceded+saved), RANK() OVER (ORDER BY saved/(conceded+saved) DESC) AS scs,
conceded, RANK() OVER (ORDER BY CASE WHEN conceded IS NULL THEN 1 ELSE 0 END, conceded) AS c,
conceded/minutes_played,RANK() OVER (ORDER BY CASE WHEN conceded/minutes_played IS NULL THEN 1
ELSE 0 END, conceded/minutes_played) AS cm, cleansheets, RANK() OVER (order BY cleansheets DESC)
AS cl, minutes_played, match_played
FROM statistiche) AS tab
WHERE tab.IDGiocatore=?";
try {
Connection conn= DBConnect.getConnection();
PreparedStatement st= conn.prepareStatement(sql);
st.setInt(1, IDGiocatore);
ResultSet rs= st.executeQuery();
rs.first();
Statistiche stats= new Statistiche(IDGiocatore, rs.getString("club"), rs.getInt("saved"),
rs.getInt("s"),
rs.getDouble("saved/(conceded+saved)"), rs.getInt("scs"), rs.getInt("conceded"), rs.getInt("c"),
rs.getDouble("conceded/minutes_played"), rs.getInt("cm"), rs.getInt("cleansheets"),
rs.getInt("cl"),rs.getInt("minutes_played"), rs.getInt("match_played"));
```

```
conn.close();
return stats;
} catch (Exception e) {
System.out.println("Errore in getStatisticsPlayer");
e.printStackTrace();
return null;
   }
}
```

- Algoritmo di ricerca ed elaborazione del ranking finale statistiche GIOCATORE DI MOVIMENTO:

```
public Statistiche getStatisticsPlayer(int IDGiocatore) {
String sql= "SELECT tab.* FROM (SELECT IDGiocatore, club, balls_recoverd, RANK() OVER (ORDER BY
balls_recoverd desc) AS b, balls_recoverd/minutes_played, RANK() OVER (ORDER BY
balls_recoverd/minutes_played DESC) AS bmp, tackles, RANK() OVER (ORDER BY tackles desc) AS t,
tackles_won, RANK() OVER (ORDER BY tackles_won DESC) AS tw, tackles_won/tackles, RANK() OVER
(ORDER BY tackles_won/tackles DESC) AS twp, pass_attempted,RANK() OVER (ORDER BY pass_attempted
desc) AS pa, pass_completed,RANK() OVER (ORDER BY pass_completed desc) AS pc,
pass_completed/pass_attempted,RANK() OVER (ORDER BY pass_completed desc) AS p, assist,RANK()
OVER (ORDER BY assist desc) AS a, assist/minutes_played,RANK() OVER (ORDER BY
assist/minutes_played desc) AS am, shot_attempts, RANK() OVER (order BY shot_attempts DESC) AS
sa, shot_on_target, RANK() OVER (order BY shot_on_target DESC) AS so,
shot_on_target/shot_attempts, RANK() OVER (order BY shot_on_target/shot_attempts DESC) AS s,
goals,RANK() OVER (ORDER BY goals desc) AS g, goals/shot_attempts,RANK() OVER (ORDER BY
goals/shot_attempts desc) AS gs, goals/minutes_played,RANK() OVER (ORDER BY goals/minutes_played
desc) AS gm, dribbles,RANK() OVER (ORDER BY dribbles desc) AS d, dribbles/minutes_played,RANK()
OVER (ORDER BY dribbles/minutes_played desc) AS dm, minutes_played, match_played
FROM statistiche) AS tab
WHERE tab.IDGiocatore=?";
try {
Connection conn= DBConnect.getConnection();
PreparedStatement st= conn.prepareStatement(sql);
st.setInt(1, IDGiocatore);
ResultSet rs= st.executeQuery();
rs.first();
Statistiche stats= new Statistiche(IDGiocatore, rs.getString("club"),
rs.getInt("balls_recoverd"), rs.getInt("b"), rs.getDouble("balls_recoverd/minutes_played"),
rs.getInt("bmp"), rs.getInt("tackles"), rs.getInt("t"), rs.getInt("tackles_won"),
rs.getInt("tw"), rs.getDouble("tackles_won/tackles"), rs.getInt("twp"),
rs.getInt("pass_attempted"), rs.getInt("pa"), rs.getInt("pass_completed"), rs.getInt("pc"),
rs.getDouble("pass_completed/pass_attempted"), rs.getInt("p"),
rs.getInt("assist"), rs.getInt("a"), rs.getDouble("assist/minutes_played"),
rs.getInt("am"),rs.getInt("shot_attempts"), rs.getInt("sa"), rs.getInt("shot_on_target"),
rs.getInt("so"), rs.getDouble("shot_on_target/shot_attempts"), rs.getInt("s"),
rs.getInt("goals"), rs.getInt("g"), rs.getDouble("goals/shot_attempts"), rs.getInt("gs"),
rs.getDouble("goals/minutes_played"), rs.getInt("gm"),rs.getInt("dribbles"), rs.getInt("d"),
rs.getDouble("dribbles/minutes_played"), rs.getInt("dm"), rs.getInt("minutes_played"),
rs.getInt("match_played"));
conn.close();
return stats;
} catch (Exception e) {
System.out.println("Errore in getStatisticsPlayer");
e.printStackTrace();
return null;
  }
}
```

- Algoritmo di ricerca ed elaborazione del ranking finale statistiche SQUADRA:

```
public Statistiche getStatisticsClub(String team) {
String sql= "SELECT * from(SELECT c.club, SUM(s.saved) AS \"saved\", RANK() OVER (ORDER BY
SUM(s.saved) desc) AS s, SUM(s.saved)/SUM(s.conceded+s.saved) AS \"saved/(conceded+saved)\",
RANK() OVER (ORDER BY (SUM(s.saved)/SUM(s.conceded+s.saved)) DESC) AS scs, sum(s.conceded) AS
\"conceded\", RANK() OVER (ORDER BY sum(s.conceded) ASC) AS c,
sum(s.conceded)/c.total_match_played AS \"conceded/total_match_played\", RANK() OVER (ORDER BY
sum(s.conceded)/c.total_match_played ASC) AS cm, SUM(s.cleansheets) AS \"cleansheets\", RANK()
OVER (order BY SUM(s.cleansheets) DESC) AS cl, SUM(s.balls_recoverd) AS \"balls_recoverd\",
RANK() OVER (ORDER BY SUM(s.balls_recoverd) desc) AS b,
SUM(s.balls_recoverd)/c.total_match_played AS \"balls_recoverd/total_match_played\", RANK() OVER
(ORDER BY SUM(s.balls_recoverd)/c.total_match_played DESC) AS bmp, SUM(s.tackles) AS
\"tackles\", RANK() OVER (ORDER BY SUM(s.tackles) desc) AS t, SUM(s.tackles_won) AS
\"tackles_won\", RANK() OVER (ORDER BY SUM(s.tackles_won) DESC) AS tw,
SUM(s.tackles_won)/SUM(s.tackles) AS \"tackles_won/tackles\", RANK() OVER (ORDER BY
SUM(s.tackles_won)/SUM(s.tackles) DESC) AS twp, SUM(s.pass_attempted) AS \"pass_attempted\",
RANK() OVER (ORDER BY SUM(s.pass_attempted) desc) AS pa, SUM(s.pass_completed) AS
\"pass_completed\",RANK() OVER (ORDER BY SUM(s.pass_completed) desc) AS pc,
SUM(s.pass_completed)/SUM(s.pass_attempted) AS \"pass_completed/pass_attempted\", RANK() OVER
(ORDER BY SUM(s.pass_completed)/SUM(s.pass_attempted) desc) AS p, SUM(s.assist) AS
\"assist\",RANK() OVER (ORDER BY SUM(s.assist) desc) AS a, SUM(s.assist)/c.total_match_played AS
\"assist/total_match_played\", RANK() OVER (ORDER BY SUM(s.assist)/c.total_match_played desc) AS
am, SUM(s.shot_attempts) AS \"shot_attempts\", RANK() OVER (order BY SUM(s.shot_attempts) DESC)
AS sa, SUM(s.shot_on_target) AS \"shot_on_target\", RANK() OVER (order BY SUM(s.shot_on_target)
```

```
DESC) AS so, SUM(s.shot_on_target)/SUM(s.shot_attempts) AS \"shot_on_target/shot_attempts\",
RANK() OVER (order BY SUM(s.shot_on_target)/SUM(s.shot_attempts) DESC) AS soa, SUM(s.goals) AS
\"goals\", RANK() OVER (ORDER BY SUM(s.goals) desc) AS g, SUM(s.goals)/SUM(s.shot_attempts) AS
\"goals/shot_attempts\", RANK() OVER (ORDER BY SUM(s.goals)/SUM(s.shot_attempts) desc) AS gs,
SUM(s.goals)/c.total_match_played AS \"goals/total_match_played\",RANK() OVER (ORDER BY
SUM(s.goals)/c.total_match_played desc) AS gm, SUM(s.dribbles) AS \"dribbles\",RANK() OVER
(ORDER BY SUM(s.dribbles) desc) AS d, SUM(s.dribbles)/c.total_match_played AS
\"dribbles/total_match_played\",RANK() OVER (ORDER BY SUM(s.dribbles)/c.total_match_played desc)
AS dm, c.total_match_played
FROM statistiche s, squadre c
WHERE s.club= c.club
GROUP BY s.club) AS tab
WHERE tab.club= ?";
try {
Connection conn= DBConnect.getConnection();
PreparedStatement st= conn.prepareStatement(sql);
st.setString(1, team);
ResultSet rs= st.executeQuery();
rs.first();
Statistiche stats= new Statistiche(rs.getString("club"),rs.getInt("saved"), rs.getInt("s"),
rs.getDouble("saved/(conceded+saved)"), rs.getInt("scs"), rs.getInt("conceded"), rs.getInt("c"),
rs.getDouble("conceded/total_match_played"), rs.getInt("cm"), rs.getInt("cleansheets"),
rs.getInt("cl"), rs.getInt("balls_recoverd"), rs.getInt("b"),
rs.getDouble("balls_recoverd/total_match_played"), rs.getInt("bmp"), rs.getInt("tackles"),
rs.getInt("t"), rs.getInt("tackles_won"), rs.getInt("tw"), rs.getDouble("tackles_won/tackles"),
rs.getInt("twp"), rs.getInt("pass_attempted"), rs.getInt("pa"), rs.getInt("pass_completed"),
rs.getInt("pc"), rs.getDouble("pass_completed/pass_attempted"), rs.getInt("p"),
rs.getInt("assist"), rs.getInt("a"), rs.getDouble("assist/total_match_played"), rs.getInt("am"),
rs.getInt("shot_attempts"), rs.getInt("sa"), rs.getInt("shot_on_target"), rs.getInt("so"),
rs.getDouble("shot_on_target/shot_attempts"), rs.getInt("soa"), rs.getInt("goals"),
rs.getInt("g"), rs.getDouble("goals/shot_attempts"), rs.getInt("gs"),
rs.getDouble("goals/total_match_played"), rs.getInt("gm"),rs.getInt("dribbles"), rs.getInt("d"),
rs.getDouble("dribbles/total_match_played"), rs.getInt("dm"), rs.getInt("total_match_played"));
conn.close();
return stats;
} catch (Exception e) {
System.out.println("Errore in getStatisticsClub");
e.printStackTrace();
return null;
  }
}
```

Questi metodi, contenuti all'interno della classe ChampionsDAO, danno come risultato un oggetto di tipo *Statistiche*. Una volta richiamato il metodo nel Model, l'oggetto *Statistiche* verrà elaborato per renderlo comprensibile e facilmente interpretabile per l'utente, trasformandolo così in una stringa che verrà poi richiamata dal Controller.

4.2.2 Algoritmo di ricerca e creazione del dream team senza vincoli

Per prima cosa sono stati popolati i menù a tendini relativi a modulo, statistica portiere da ottimizzare e statistica giocatori di movimento da ottimizzare, con i valori prestabiliti a inizio progetto.
Una volta che l'utente schiaccia il relativo bottone per la creazione del **dream team senza vincoli** viene sfruttato il seguente algoritmo di ricerca, per la scelta del PORTIERE:

```
public DreamTeam goalkeeperForStatisticaDaOttimizzare() {
String sql="SELECT g.name, g.`position`, s.club, g.age, g.`FIFA_wage_€`, g.`FIFA_value_€`,
s.minutes_played, statisticaDaOttimizzare AS stats FROM giocatori g, statistiche s
WHERE g.IDGiocatore= s.IDGiocatore AND g.`position`=\"Portiere\" AND s.minutes_played>= 540
ORDER BY stats ASC";
try {
Connection conn= DBConnect.getConnection();
PreparedStatement st= conn.prepareStatement(sql);
ResultSet rs= st.executeQuery();
rs.first();
DreamTeam risultato= new DreamTeam(rs.getString("name"),rs.getString("position"),
rs.getString("club"),
rs.getInt("age"),rs.getDouble("FIFA_wage_€"),rs.getDouble("FIFA_value_€"),rs.getInt("minutes_played"
), "statistica da ottimizzare", rs.getDouble("stats"));
conn.close();
return risultato;
} catch (Exception e) {
System.out.println("Errore in goalkeeperForStatisticaDaOttimizzare");
e.printStackTrace();
return null;
```

```
  }
}
```

Mentre l'algoritmo di ricerca per la scelta dei GIOCATORI DI MOVIMENTO è il seguente:

```
public List<DreamTeam> playersForStatisticaDaOttimizzare(){
String sql= "SELECT g.name, g.`position`, s.club, g.age, g.`FIFA_wage_€`, g.`FIFA_value_€`,
statisticaDaOttimizzare AS stats
FROM giocatori g, statistiche s
WHERE g.IDGiocatore= s.IDGiocatore AND g.`position`!=\"Portiere\" AND s.minutes_played>= 540
ORDER BY stats DESC";
List<DreamTeam> risultato= new ArrayList<>();
try {
Connection conn= DBConnect.getConnection();
PreparedStatement st= conn.prepareStatement(sql);
ResultSet rs= st.executeQuery();
while(rs.next())
risultato.add(new DreamTeam(rs.getString("name"), rs.getString("position"), rs.getString("club"),
rs.getInt("age"),rs.getDouble("FIFA_wage_€"), rs.getDouble("FIFA_value_€"),
rs.getInt("minutes_played"), " C ", rs.getDouble("stats")));
conn.close();
return risultato;
} catch (Exception e) {
System.out.println("Errore in playersForStatistiche DaOttimizzare");
e.printStackTrace();
return null;
  }
}
```

NOTA: Gli algoritmi sopra riportati, per la ricerca del portiere e dei giocatori di movimento, non sono quelli realmente implementati, in quanto la voce "Statistica Da Ottimizzare" è sostituita con la scelta effettuata dall'utente nei menù a tendina: statistica portiere da ottimizzare e statistica giocatori di movimento da ottimizzare.
L'ordinamento dei record (crescente o decrescente) dipenderà dal tipo di statistica, cioè se debba essere minimizzata (ordinamento crescente) o massimizzata (ordinamento decrescente).

Il metodo per la ricerca del portiere darà come risultato un unico oggetto *DreamTeam*, mentre dall'algoritmo per la ricerca dei giocatori di movimento sarà ottenuta una lista contenente oggetti *DreamTeam*.
Una volta richiamato il metodo per la ricerca dei giocatori di movimento nel model, la sua lista risultante verrà snellita in modo tale da ottenere solo i 10 giocatori di movimento richiesti, rispettando i vincoli imposti dal modulo (andando quindi a selezionare un numero di difensori, di centrocampisti e di attaccanti stabilito dal modulo scelto dall'utente).
Per ultimo, sempre nel model, dopo aver ottenuto la lista finale dei 10 giocatori di movimento, sarà curata la rappresentazione dei dati che sarà fatta sotto forma di stringa, dividendo i giocatori per ruoli.
A questo punto il risultato verrà richiamato nel controller e verrà presentato all'utente.

4.2.3 Algoritmo ricorsivo per la creazione del dream team con vincoli

Per prima cosa sono stati popolati i menù a tendini relativi a modulo, statistica portiere da ottimizzare, statistica giocatori di movimento da ottimizzare, numero massimo giocatori di movimento per squadra con i valori prestabiliti a inizio progetto ed infine l'età con tutti i valori compresi tra la minima e la massima età presenta nella tabella giocatori.
Una volta che l'utente ha compilato tutti i campi e schiacciato il relativo bottone per la creazione del **dream team con vincoli** si avvia l'algoritmo. Esso fa una serie di controlli dell'input dello user agent, verificando se i vincoli imposti dall'utente siano troppo stringenti, non permettendo perciò di ottenere 11 giocatori, suddivisi nelle posizioni richieste dall'utente.
Dopodiché, per fini di ottimizzazione, viene verificato se la soluzione ottenuta senza ricorsione ("dream team senza vincoli") rispetti i vincoli di *value*, *wage*, numero massimo giocatori di movimento per squadra imposti dall'utente; in caso negativo viene controllato se i vincoli inseriti dall'utente siano troppo ampi, rendendo così l'algoritmo ricorsivo computazionalmente troppo oneroso.
Arrivati perciò a questo punto può esistere una soluzione ottima, ma va ricercata mediante un algoritmo ricorsivo.
Dapprima viene selezionato il portiere, che viene scelto mediante un algoritmo di ricerca, essendo un unico record; successivamente è necessario fornire la lista con tutti i record riguardanti i giocatori di movimento,

filtrati in base ai vincoli imposti dall'utente, e, mediante l'algoritmo ricorsivo viene trovata, se presente, la soluzione ottima.

```
// Parametri RICORSIONE
private List<DreamTeam> bestPlayers= new ArrayList<>();
private List<DreamTeam> playersForDreamTeam= new ArrayList<>(lista);
// Ottimizzazione legata ai ruoli dei giocatori
private int spyNumDif=0;
private int spyNumCen=0;
private int spyNumAtt=0;
// Ottimizzazione legata a value e wage
private double maxTeamValue= maxValue;
private double maxTeamWage= maxWage;
// Ottimizzazione legata a giocatori appartenenti alla stessa squadra
private int playersFromSameTeam= maxSameTeam;
// Valore
private double bestStatsValue= 0.0;
List<DreamTeam> parziale= new ArrayList<>();
// Popolo la mappa con tutte le squadre e numero di giocatori uguale a 0
Map<String, Integer> clubs= new HashMap<>();

for(Squadra s: getAllTeams())
    clubs.put(s.getClub(), 0);

ricorsione(parziale, 0, 0.0, 0.0, 0.0, clubs, numDifensori, numCentrocampisti, numAttaccanti);
```

NOTA: Si tratta di un frammento di codice per far comprendere da dove provengano i parametri della ricorsione

4.2.3.1 Ricorsione implementata all'interno del codice

```
private void ricorsione(List<DreamTeam> parziale, int livello, double statistica, double value,
double wage, Map<String, Integer> clubs, int Ndifensori, int Ncentrocampisti, int Nattaccanti) {
if(livello== 10) {
    if(statistica> this.bestStatsValue) {
        this.bestPlayers= new ArrayList<>(parziale);
        this.bestStatsValue= statistica;
    }
    return;
}
for(DreamTeam player: this.playersForDreamTeam) {
    String position= player.getPosition();
    if(checkPosition(position, Ndifensori, Ncentrocampisti, Nattaccanti) && (livello==0 ||
            parziale.get(parziale.size()-1).getName().compareTo(player.getName())< 0) &&
            ((value+player.getFifaValue())<= this.maxTeamValue) && ((wage+player.getFifaWage())<=
            this.maxTeamWage) && clubs.get(player.getClub())< this.playersFromSameTeam) {
        parziale.add(player);
        clubs.put(player.getClub(), clubs.get(player.getClub())+1);
        // Aggiornare il numero di giocatori per quel ruolo
        editFormation(position, 1);
        ricorsione(parziale, livello+1, statistica+player.getStatsValue(),
                value+player.getFifaValue(), wage+player.getFifaWage(), clubs, Ndifensori,
                Ncentrocampisti, Nattaccanti);
        clubs.put(player.getClub(), clubs.get(player.getClub())-1);
        editFormation(position, -1);
        parziale.remove(parziale.size()-1);
     }
   }
return;
}
```

Ai fini di ottimizzare l'algoritmo ricorsivo, riducendo così il numero di chiamate, sono stati introdotti i seguenti vincoli, prima di inserire un giocatore nella lista di appoggio:

- Ricercare i record in ordine alfabetico crescente;
- Verificare che la somma tra *value* e *wage* attuali e quelli del player da inserire non superi i tetti massimi imposti dall'utente;
- Verificare che la numerosità imposta per ogni reparto non venga superata;
- Verificare che non venga superato il numero massimo di calciatori appartenenti alla stessa squadra.

5 - Diagramma delle classi delle parti principali dell'applicazione

Model (package: it.polito.tdp.provaFinale)

- dao : ChampionsDAO
- bestPlayers : List<DreamTeam>
- playersForDreamTeam : List<DreamTeam>
- spyNumDif : int
- spyNumCen : int
- spyNumAtt : int
- maxTeamValue : double
- maxTeamWage : double
- playersFromSameTeam : int
- bestStatsValue : double
- Model()
- getAllPlayers() : List<Giocatore>
- getAllTeams() : List<Squadra>
- createDreamTeam(String, String, String) : String
- getGoalkeeperDreamTeam(String) : DreamTeam
- getPlayersDreamTeam(String, int, int, int) : List<DreamTeam>
- createDreamTeam(String, String, String, double, double, double, double, int, int, int, int) : String
- getGoalkeeperDreamTeam(String, int, int, int, double, double) : DreamTeam
- getPlayersDreamTeam(String, int, int, int, int, int, int, double, double, int) : List<DreamTeam>
- ricorsione(List<DreamTeam>, int, double, double, double, Map<String, Integer>, int, int, int) : void
- checkPosition(String, int, int, int) : boolean
- editFormation(String, int) : void
- getStatisticsPlayer(Giocatore) : String
- getStatisticsClub(String) : String

DreamTeam (package: it.polito.tdp.provaFinale)

- name : String
- position : String
- club : String
- age : int
- fifaWage : double
- fifaValue : double
- minutesPlayed : int
- statsName : String
- statsValue : double
- DreamTeam(String, String, String, int, double, double, int, String, double)
- getName() : String
- getPosition() : String
- getClub() : String
- getAge() : int
- getFifaWage() : double
- getFifaValue() : double
- getMinutesPlayed() : int
- getStatsName() : String
- getStatsValue() : double
- compareTo(DreamTeam) : int
- toString() : String

Statistiche (package: it.polito.tdp.provaFinale)

- IDGiocatore : int
- club : String
- saved : int
- savedRank : int
- savedDividedConcededPlusSaved : double
- ranksavedDividedConcededPlusSaved : int
- conceded : int
- concededRank : int
- concededDividedMinutesPlayed : double
- rankConcededDividedMinutesPlayed : int
- concededDividedMatchPlayed : double
- rankConcededDividedMatchPlayed : int
- cleansheets : int
- cleansheetsRank : int
- ballsRecoverd : int
- ballsRecoverdRank : int
- ballsRecoverdDividedMinutesPlayed : double
- rankBallsRecoverdDividedMinutesPlayed : int
- ballsRecoverdDividedMatchPlayed : double
- rankBallsRecoverdDividedMatchPlayed : int
- tackles : int
- tacklesRank : int
- tacklesWon : int
- tacklesWonRank : int
- tacklesWonDividedTackles : double
- rankTacklesWonDividedTackles : int
- passAttempted : int
- passAttemptedRank : int
- passCompleted : int
- passCompletedRank : int
- passCompletedDividedPassAttempted : double
- passCompletedDividedPassAttemptedRank : int
- assist : int
- rankAssist : int
- assistDividedMinutesPlayed : double
- rankAssistDividedMinutesPlayed : int
- assistDividedMatchPlayed : double
- rankAssistDividedMatchPlayed : int
- shotAttempts : int
- rankShotAttempts : int
- shotOnTarget : int
- rankShotOnTarget : int
- shotOnTargetDividedShotAttempts : double
- shotOnTargetDividedShotAttemptsRank : int
- goals : int
- goalsRank : int
- goalsDividedShotAttempts : double
- goalsDividedShotAttemptsRank : int
- goalsDividedMinutesPlayed : double
- goalsDividedMinutesPlayedRank : int
- goalsDividedMatchPlayed : double
- goalsDividedMatchPlayedRank : int
- dribbles : int
- dribblesRanK : int
- dribblesDividedMinutesPlayed : double
- dribblesDividedMinutesPlayedRank : int
- dribblesDividedMatchPlayed : double
- dribblesDividedMatchPlayedRank : int
- minutesPlayed : int
- matchPlayed : int

- getIDGiocatore() : int
- getClub() : String
- getSaved() : int
- getSavedDividedConcededPlusSaved() : double
- getConceded() : int
- getConcededDividedMinutesPlayed() : double
- getConcededDividedMatchPlayed() : double
- getCleansheets() : int
- getBallsRecoverd() : int
- getBallsRecoverdDividedMinutesPlayed() : double
- getBallsRecoverdDividedMatchPlayed() : double
- getTackles() : int
- getTacklesWon() : int
- getTacklesWonDividedTackles() : double
- getPassAttempted() : int
- getPassCompleted() : int
- getPassCompletedDividedPassAttempted() : double
- getAssist() : int
- getAssistDividedMinutesPlayed() : double
- getAssistDividedMatchPlayed() : double
- getShotAttempts() : int
- getShotOnTarget() : int
- getShotOnTargetDividedShotAttempts() : double
- getGoals() : int
- getGoalsDividedShotAttempts() : double
- getGoalsDividedMinutesPlayed() : double
- getGoalsDividedMatchPlayed() : double
- getDribbles() : int
- getDribblesDividedMinutesPlayed() : double
- getDribblesDividedMatchPlayed() : double
- getMinutesPlayed() : int
- getMatchPlayed() : int
- getSavedRank() : int
- getRanksavedDividedConcededPlusSaved() : int
- getConcededRank() : int
- getRankConcededDividedMinutesPlayed() : int
- getRankConcededDividedMatchPlayed() : int
- getCleansheetsRank() : int

- getBallsRecoverdRank() : int
- getRankBallsRecoverdDividedMinutesPlayed() : int
- getRankBallsRecoverdDividedMatchPlayed() : int
- getTacklesRank() : int
- getTacklesWonRank() : int
- getRankTacklesWonDividedTackles() : int
- getPassAttemptedRank() : int
- getPassCompletedRank() : int
- getPassCompletedDividedPassAttemptedRank() : int
- getRankAssist() : int
- getRankAssistDividedMinutesPlayed() : int
- getRankAssistDividedMatchPlayed() : int
- getRankShotAttempts() : int
- getRankShotOnTarget() : int
- getShotOnTargetDividedShotAttemptsRank() : int
- getGoalsRank() : int
- getGoalsDividedShotAttemptsRank() : int
- getGoalsDividedMinutesPlayedRank() : int
- getGoalsDividedMatchPlayedRank() : int
- getDribblesRanK() : int
- getDribblesDividedMinutesPlayedRank() : int
- getDribblesDividedMatchPlayedRank() : int
- hashCode() : int
- equals(Object) : boolean
- toString() : String

- Statistiche(int, String, int, int, double, int, int, int, double, int, int, int, int, int)
- Statistiche(int, String, int, int, double, int, int, int, int, int, double, int, int, int, int, int, double, int, int, int, int, double, int, int, int, int, double, int, int, int, double, int, double, int, int, int, double, int, int, int)
- Statistiche(String, int, int, double, int, int, int, double, int, int, int, int, int, double, int, int, int, int, int, double, int, int, int, int, int, double, int, int, int, int, int, double, int, int, int, double, int, double, int, in

ChampionsDAO (package: it.polito.tdp.provaFinale.dao)

- getAllPlayers() : List<Giocatore>
- getAllTeams() : List<Squadra>
- goalkeeperForSavedDividedConcededPlusSaved() : DreamTeam
- goalkeeperForConcededDividedMinutesPlayed() : DreamTeam
- goalkeeperForCleansheetsDividedMatchPlayed() : DreamTeam
- playersForBallsRecoverdDividedMinutesPlayed() : List<DreamTeam>
- playersForTacklesWonDividedTackles() : List<DreamTeam>
- playersForPassCompletedDividedPassAttempted() : List<DreamTeam>
- playersForAssistDividedMinutesPlayed() : List<DreamTeam>
- playersForGoalsDividedMinutesPlayed() : List<DreamTeam>
- playersForDribblesDividedMinutesPlayed() : List<DreamTeam>
- playersForShotOnTargetDividedShotAttempts() : List<DreamTeam>
- playersForGoalsDividedShotAttempts() : List<DreamTeam>
- goalkeeperForSavedDividedConcededPlusSaved(int, int, int, double, double) : DreamTeam
- goalkeeperForConcededDividedMinutesPlayed(int, int, int, double, double) : DreamTeam
- goalkeeperForCleansheetsDividedMatchPlayed(int, int, int, double, double) : DreamTeam
- playersForBallsRecoverdDividedMinutesPlayed(int, int, int, double, double) : List<DreamTeam>
- playersForTacklesWonDividedTackles(int, int, int, double, double) : List<DreamTeam>
- playersForPassCompletedDividedPassAttempted(int, int, int, double, double) : List<DreamTeam>
- playersForAssistDividedMinutesPlayed(int, int, int, double, double) : List<DreamTeam>
- playersForGoalsDividedMinutesPlayed(int, int, int, double, double) : List<DreamTeam>
- playersForDribblesDividedMinutesPlayed(int, int, int, double, double) : List<DreamTeam>
- playersForShotOnTargetDividedShotAttempts(int, int, int, double, double) : List<DreamTeam>
- playersForGoalsDividedShotAttempts(int, int, int, double, double) : List<DreamTeam>
- getStatisticsGoalkeeper(int) : Statistiche
- getStatisticsPlayer(int) : Statistiche
- getStatisticsClub(String) : Statistiche

6 - Alcune videate dell'applicazione e link al video

L'interfaccia grafica è stata realizzata grazie all'ausilio del software *SceneBuilder*.
Tutti i campi in cui è necessaria l'interazione dell'utente sono caratterizzati da placeholder, facilitandone così l'utilizzo e rendendo, in questo modo, l'applicazione user friendly.
La presenza dei placeholder, inoltre, dà informazioni aggiuntive riguardando la formattazione del dato da inserire. L'area di testo conterrà i risultati della scelta effettuata dall'utente.
- Nel caso l'utente scelga di visualizzare le **statistiche di un giocatore** per prima cosa verrà riportato il numero totale di record presenti nel database, il giocatore selezionato, al quale verrà aggiunto il ruolo, la squadra di provenienza, l'età, il FIFA value ed il FIFA wage. Dopodiché a seconda che si tratti di un portiere o di un giocatore di movimento verranno rappresentate le statistiche sopra elencate, divise in sezioni (PORTIERE, DIFENSIVE, PASSAGGI, OFFENSIVE) ed infine i minuti e le partite totali in cui i calciatori hanno preso parte.
- Nel caso in cui la scelta ricadesse su **una squadra**, per prima cosa verrà riportato il numero totale di record presenti nel database, le partite disputate dalla squadra selezionata e tutte le statistiche, sempre divise in sezioni (PORTIERE, DIFENSIVE, PASSAGGI, OFFENSIVE).
- Se invece l'utente volesse visualizzare il **dream team**, i dati estrapolati saranno divisi in quattro sezioni: PORTIERE, DIFENSORI, CENTROCAMPISTI e ATTACCANTI ed avranno numerosità pari al modulo selezionato dall'utente. Infine per ogni calciatore facente parte del dream team verranno visualizzati anche: nome, squadra, età, FIFA value, FIFA wage, caratteristica da ottimizzare e valore della caratteristica da ottimizzare.


Prova finale
Statistiche UEFA Champions League 2021-2022
CHAMPIONS LEAGUE
Statistiche singolo giocatore:
Alexis Sánchez
Ricerca statistiche giocatore
Statistiche squadra:
Selezionare una squadra
Ricerca statistiche squadra
Dream team:
Modulo
Statistica portiere da ottimizzare
Statistica giocatori di movimento da ottimizzare
Selezionare un modulo
Selezionare caratteristica portiere
Selezionare caratteristica giocatori di movimento
Crea Dream Team senza vincoli
Budget totale value
Percentuale value per il portiere
Budget totale wage
Percentuale wage per il portiere
Esempio: 484000000.00 €
Esempio: 9.50 %
Esempio: 1067000.00 €
Esempio: 11.85 %
Numero minimo minuti giocati
Età
Numero massimo giocatori di movimento per squadra
inserire un valore ≥ 540 min
Minima
Massima
Selezionare un valore
Crea Dream Team con vincoli
In totale nel database ci sono: 693 giocatori di movimento.
GIOCATORE SELEZIONATO:
Alexis Sánchez, Attaccante, Inter, 32 anni, FIFA value: 11.000.000 €, FIFA wage: 105.000 €
STATISTICHE DIFENSIVE:
- balls recoverd: 4 → 478°
- balls recoverd / minutes played: 0.0258 → 473°
- tackles: 6 → 162°
- tackles won: 2 → 184°


Prova finale
Statistiche UEFA Champions League 2021-2022
CHAMPIONS LEAGUE
Statistiche singolo giocatore:
Selezionare un giocatore
Ricerca statistiche giocatore
Statistiche squadra:
Atlético de Madrid
Ricerca statistiche squadra
Dream team:
Modulo
Statistica portiere da ottimizzare
Statistica giocatori di movimento da ottimizzare
Selezionare un modulo
Selezionare caratteristica portiere
Selezionare caratteristica giocatori di movimento
Crea Dream Team senza vincoli
Budget totale value
Percentuale value per il portiere
Budget totale wage
Percentuale wage per il portiere
Esempio: 484000000.00 €
Esempio: 9.50 %
Esempio: 1067000.00 €
Esempio: 11.85 %
Numero minimo minuti giocati
Età
Numero massimo giocatori di movimento per squadra
inserire un valore ≥ 540 min
Minima
Massima
Selezionare un valore
Crea Dream Team con vincoli
STATISTICHE PASSAGGI:
- pass attempted: 3226 → 15°
- pass completed: 2597 → 16°
- pass completed / pass attempted: 0.805 → 27°
- assist: 7 → 15°
- assist / match played: 0.7 → 22°
STATISTICHE OFFENSIVE:
- shot attempts: 96 → 10°
- shot on target: 27 → 18°

Link al video dimostrativo: https://youtu.be/4sGSEhHNXyY

7 - Tabelle con risultati sperimentali ottenuti

7.1 Risultato statistiche singolo giocatore

INPUT: Lukaku

OUTPUT:

In totale nel database ci sono: 693 giocatori di movimento.

GIOCATORE SELEZIONATO:
Lukaku, Attaccante, Chelsea, 28 anni, FIFA value: 68.000.000 €, FIFA wage: 260.000 €
STATISTICHE DIFENSIVE:
- balls recoverd: 2 → 542º
- balls recoverd / minutes played: 0.0063 → 617º
- tackles: 1 → 433º
- tackles won: 1 → 279º
- tackles won / tackles: 1.0 → 1º

STATISTICHE PASSAGGI:
- pass attempted: 61 → 480º
- pass completed: 49 → 478º
- pass completed / pass attempted: 0.8033 → 478º
- assist: 0 → 177º
- assist / minutes played: 0.0 → 177º

STATISTICHE OFFENSIVE:
- shot attempts: 11 → 67º
- shot on target: 3 → 87º
- shot on target / shot attempts: 0.2727 → 268º
- goals: 2 → 42º
- goals / shot attempts: 0.1818 → 114º
- goals / minutes played: 0.0063 → 46º
- dribbles: 0 → 176º
- dribbles / minutes played: 0.0 → 176º

Ha giocato: 320 minuti in 6 partite.

TEMPO ESECUZIONE: 0.018 secondi

7.2 Risultato statistiche squadra

INPUT: Real Madrid CF

OUTPUT:

In totale nel database ci sono: 32 squadre.

SQUADRA SELEZIONATA:
Real Madrid CF, 13 partite giocate.

STATISTICHE PORTIERI:
- saved: 61 → 1º
- saved / (conceded + saved): 0.8133 → 1º
- conceded: 14 → 22º
- conceded / match played: 1.0769 → 8º
- cleansheets: 5 → 1º

STATISTICHE DIFENSIVE:
- balls recoverd: 514 → 2º
- balls recoverd / match played: 39.5385 → 20º
- tackles: 166 → 1º
- tackles won: 76 → 1º
- tackles won / tackles: 0.4578 → 13º

STATISTICHE PASSAGGI:
- pass attempted: 6932 → 3º
- pass completed: 6148 → 3º
- pass completed / pass attempted: 0.8869 → 5º
- assist: 21 → 4º
- assist / match played: 1.6154 → 7º

STATISTICHE OFFENSIVE:
- shot attempts: 179 → 4º
- shot on target: 69 → 3º

- shot on target / shot attempts: 0.3855 → 9º
- goals: 28 → 2º
- goals / shot attempts: 0.1564 → 6º
- goals / match played: 2.1538 → 5º
- dribbles: 165 → 2º
- dribbles / match played: 12.6923 → 3º

TEMPO ESECUZIONE: 0.011 secondi

7.3 Dream Team

Di seguito vengono proposti 3 esempi applicativi per la generazione di un Dream Team

7.3.1 Esempio 1

INPUT:

- Modulo: 4-3-3
- Statistica portiere da ottimizzare: saved / (conceded + saved)
- Statistica giocatori di movimento da ottimizzare: goals / minutes played
- Budget totale value: 484.000.000,00 €
- Percentuale value per il portiere: 9 %
- Budget totale wage: 10.670.000,00 €
- Percentuale wage per il portiere: 11%
- Numero minimo minuti giocati: 540 min
- Età → Minima: 16 Massima: 25
- Numero massimo giocatori di movimento per squadra: 1

OUTPUT:
DREAM TEAM con vincoli:

PORTIERE
Grbic, LOSC Lille, 25 anni, FIFA value: 4.500.000 €, FIFA wage: 25.000 €, saved / (conceded + saved): 0.7647

DIFENSORI
Christensen, Chelsea, 25 anni, FIFA value: 37.500.000 €, FIFA wage: 95.000 €, goals / minutes played: 0.0017
Konaté, Liverpool, 22 anni, FIFA value: 48.000.000 €, FIFA wage: 73.000 €, goals / minutes played: 0.0028
Wöber, FC Red Bull Salzburg, 23 anni, FIFA value: 7.500.000 €, FIFA wage: 19.000 €, goals / minutes played: 0.0017
Zabarnyi, Dynamo Kyiv, 18 anni, FIFA value: 6.000.000 €, FIFA wage: 500 €, goals / minutes played: 0.0

CENTROCAMPISTI
Danjuma, Villarreal CF, 24 anni, FIFA value: 36.500.000 €, FIFA wage: 28.000 €, goals / minutes played: 0.0066
Foden, Manchester City, 21 anni, FIFA value: 109.500.000 €, FIFA wage: 125.000 €, goals / minutes played: 0.0046
Sané, FC Bayern München, 25 anni, FIFA value: 49.500.000 €, FIFA wage: 105.000 €, goals / minutes played: 0.0075

ATTACCANTI
Antony, Ajax, 21 anni, FIFA value: 49.000.000 €, FIFA wage: 17.000 €, goals / minutes played: 0.0035
David, LOSC Lille, 21 anni, FIFA value: 28.500.000 €, FIFA wage: 31.000 €, goals / minutes played: 0.0044
Núñez, SL Benfica, 22 anni, FIFA value: 61.500.000 €, FIFA wage: 13.000 €, goals / minutes played: 0.0098

GIOCATORI DI MOVIMENTO ANALIZZATI: 51
TEMPO ESECUZIONE: 51.778 secondi

7.3.2 Esempio 2

INPUT:

- Modulo: 4-4-2
- Statistica portiere da ottimizzare: clean sheets / match played
- Statistica giocatori di movimento da ottimizzare: pass completed / pass attempted
- Budget totale value: 375.000.000,00 €
- Percentuale value per il portiere: 5 %
- Budget totale wage: 950.000,00 €
- Percentuale wage per il portiere: 10%
- Numero minimo minuti giocati: 540 min
- Età → Minima: 24 Massima: 27
- Numero massimo giocatori di movimento per squadra: 2

OUTPUT:
DREAM TEAM con vincoli:

PORTIERE
Grbic, LOSC Lille, 25 anni, FIFA value: 4.500.000 €, FIFA wage: 25.000 €, clean sheets / match played: 0.5

DIFENSORI
Christensen, Chelsea, 25 anni, FIFA value: 37.500.000 €, FIFA wage: 95.000 €, pass completed / pass attempted: 0.9366
Pau Torres, Villarreal CF, 24 anni, FIFA value: 50.500.000 €, FIFA wage: 43.000 €, pass completed / pass attempted: 0.9032
Pavard, FC Bayern München, 25 anni, FIFA value: 20.000.000 €, FIFA wage: 64.000 €, pass completed / pass attempted: 0.9068
Stones, Manchester City, 27 anni, FIFA value: 34.500.000 €, FIFA wage: 140.000 €, pass completed / pass attempted: 0.9596

CENTROCAMPISTI
Barrios, Zenit St. Petersburg, 27 anni, FIFA value: 18.000.000 €, FIFA wage: 66.000 €, pass completed / pass attempted: 0.9216
Fabinho, Liverpool, 27 anni, FIFA value: 77.000.000 €, FIFA wage: 165.000 €, pass completed / pass attempted: 0.9227
S. Thill, FC Sheriff Tiraspol, 27 anni, FIFA value: 1.600.000 €, FIFA wage: 850 €, pass completed / pass attempted: 0.8107
Weigl, SL Benfica, 25 anni, FIFA value: 23.000.000 €, FIFA wage: 16.000 €, pass completed / pass attempted: 0.9003

ATTACCANTI
Haller, Ajax, 27 anni, FIFA value: 31.000.000 €, FIFA wage: 23.000 €, pass completed / pass attempted: 0.8272
Sterling, Manchester City, 26 anni, FIFA value: 72.500.000 €, FIFA wage: 290.000 €, pass completed / pass attempted: 0.8981

GIOCATORI DI MOVIMENTO ANALIZZATI: 38
TEMPO ESECUZIONE: 11.097 secondi

7.3.3 Esempio 3

INPUT:

- Modulo: 3-5-2
- Statistica portiere da ottimizzare: conceded / minutes played
- Statistica giocatori di movimento da ottimizzare: tackles won / tackles
- Budget totale value: 325.000.000,00 €
- Percentuale value per il portiere: 10 %
- Budget totale wage: 2.100.000,00 €
- Percentuale wage per il portiere: 7 %
- Numero minimo minuti giocati: 600 min
- Età → Minima: 28 Massima: 36
- Numero massimo giocatori di movimento per squadra: 3

OUTPUT:
DREAM TEAM con vincoli:

PORTIERE
Neuer, FC Bayern München, 35 anni, FIFA value: 13.500.000 €, FIFA wage: 86.000 €, conceded / minutes played: 0.0074

DIFENSORI
Albiol, Villarreal CF, 35 anni, FIFA value: 5.500.000 €, FIFA wage: 30.000 €, tackles won / tackles: 0.6667
Otamendi, SL Benfica, 33 anni, FIFA value: 8.500.000 €, FIFA wage: 18.000 €, tackles won / tackles: 0.6071
Vertonghen, SL Benfica, 34 anni, FIFA value: 7.000.000 €, FIFA wage: 18.000 €, tackles won / tackles: 0.6667

CENTROCAMPISTI
André, LOSC Lille, 30 anni, FIFA value: 12.500.000 €, FIFA wage: 43.000 €, tackles won / tackles: 0.5
Koke, Atlético de Madrid, 29 anni, FIFA value: 34.500.000 €, FIFA wage: 90.000 €, tackles won / tackles: 0.7273
Modric, Real Madrid CF, 35 anni, FIFA value: 29.000.000 €, FIFA wage: 190.000 €, tackles won / tackles: 0.3333
Parejo, Villarreal CF, 32 anni, FIFA value: 42.000.000 €, FIFA wage: 64.000 €, tackles won / tackles: 0.4706
Thiago Alcántara, Liverpool, 30 anni, FIFA value: 55.500.000 €, FIFA wage: 180.000 €, tackles won / tackles: 0.6875

ATTACCANTI
Benzema, Real Madrid CF, 33 anni, FIFA value: 64.000.000 €, FIFA wage: 350.000 €, tackles won / tackles: 1.0
Rafa Silva, SL Benfica, 28 anni, FIFA value: 30.000.000 €, FIFA wage: 22.000 €, tackles won / tackles: 0.5

GIOCATORI DI MOVIMENTO ANALIZZATI: 34
TEMPO ESECUZIONE: 2.714 secondi

8 - Valutazioni sui risultati ottenuti e conclusioni

L'applicazione sviluppata si basa principalmente su due algoritmi: un algoritmo di ricerca e un algoritmo ricorsivo.
In merito all'algoritmo di ricerca la visualizzazione dei dati statistici elaborati è pressoché immediata, con tempi di risposta all'incirca inferiori al secondo. Su questo schema si basa la visualizzazione delle statistiche relative ai singoli giocatori, alle singole squadre ed alla creazione del dream team senza vincoli.

L'algoritmo ricorsivo rappresenta, invece, la parte più onerosa del programma, dal momento che i tempi di risposta variano a seconda dell'input inserito dall'utente. Ci sono tre aspetti principali che influenzano la velocità di esecuzione dell'algoritmo: 1) **la numerosità del campione** contenente i giocatori di movimento, 2) il **numero massimo di giocatori di movimento per squadra**; 3) la scelta del **modulo di gioco**.

Analizzando la **numerosità del campione** contenente i giocatori di movimento, questo parametro può essere facilmente controllato dall'utente quando inserisce i vincoli relativi a: numero minimo di minuti giocati ed età minima e massima dei giocatori da prendere in considerazione. Perciò inserendo un range di età molto ampio e un numero di minuti minimi giocati pari al limite inferiore (540 minuti) si otterranno molti record tra cui andare a ricercare i migliori 10 giocatori di movimento, rendendo, così, la ricerca ricorsiva praticamente insostenibile, dal punto di vista dei tempi di elaborazione. Infatti, più record ho, più chiamate ricorsive dovrò effettuare e non sarò così in grado di ottenere una soluzione in un tempo accettabile.
A tale scopo è stato introdotto, all'interno del codice, un controllo riguardante il numero di record trattabili per la ricorsione (min. 10 - max. 51), in modo che l'utente ottenga un output in un tempo massimo dell'ordine del minuto. Se viene, invece, superato il numero massimo di record trattabili, l'elaborazione non viene effettuata e viene mostrato all'utente un messaggio di errore, che lo invita a restringere il campo di ricerca, inserendo vincoli più restrittivi.

A parità di record da analizzare un elemento che influenza in larga misura la velocità di esecuzione è rappresentato dal **numero massimo giocatori di movimento per squadra**. Infatti, nel caso si scelga 1 giocatore per squadra, la velocità di esecuzione è molto elevata, dal momento che le squadre all'interno del database hanno come minimo un'ampiezza di rosa pari a 20. In questo modo si è in grado di "tagliare" molte chiamate ricorsive, risparmiando molto tempo, dal momento che posso inserire un unico giocatore per ogni squadra.
L'ultimo fattore che influenza la velocità di esecuzione, a parità di record da analizzare, è rappresentato dalla scelta del **modulo di gioco**. Moduli equilibrati, ad esempio: 4-3-3, 3-4-3, 4-4-2, garantiscono una velocità di esecuzione media e costante, mentre gli altri moduli che hanno al loro interno reparti con 1 o 5 giocatori non garantiscono velocità costanti e questo dipende dalla quantità di record, per quel reparto, presenti all'interno della lista. Ad esempio, se il reparto è molto numeroso ed il suo valore corrispondente è 1, la ricerca ricorsiva sarà molto veloce; viceversa, se il suo valore corrisponde a 5 la soluzione sarebbe ottenuta in tempi molto più lunghi. Infine gli ultimi vincoli, che però hanno meno peso sul tempo di risposta, sono rappresentati dal budget per *value* e *wage* e dalla selezione delle statistiche ricercate.

In conclusione, con i vincoli impostati, l'applicazione riesce a creare in tempi abbastanza ragionevoli il dream team. L'unico aspetto negativo viene rappresentato dal fatto che, in alcuni casi, il software richieda vincoli molti restrittivi, non permettendo perciò di avere una visione globale del problema ricercato.