Politecnico di Torino

Politecnico di Torino

Corso di Laurea in Ingegneria Gestionale
Classe L8: percorso Informatico
A.A. 2023/2024
Sessione di Laurea Marzo 2024

Simulatore della Ryder Cup

Simulatore del torneo di golf più seguito al mondo

Relatore:
Professor Corno Fulvio

Candidato:
Verrazzani Carlo

Sommario

CAPITOLO 1 - PROPOSTA DEL PROGETTO

1.1 Studente proponente

S269249 Verrazzani Carlo

1.2 Titolo della proposta

Simulatore della Ryder Cup di golf: il torneo di golf a squadre più seguito al mondo.

CAPITOLO 2 - COPIA INTEGRALE DELLA PROPOSTA DI PROGETTO

2.1 Descrizione del problema proposto

Lo scopo del programma è di simulare in modo realistico l'andamento della gara di golf più importante a livello mondiale: la Ryder Cup, una competizione che vede schierati da una parte i migliori 12 giocatori europei e dall'altra i migliori 12 americani. La gara dura 3 giorni e prevede 8 incontri di doppi il primo giorno, 8 incontri di doppi il secondo giorno e 12 incontri singoli il terzo giorno. Ogni vittoria porta 1 punto alla squadra vincitrice, zero alla squadra perdente e 0,5 ad entrambe in caso di pareggio. L'idea è di creare un simulatore che sia in grado di selezionare i giocatori per le 2 squadre e decretare attraverso la simulazione l'andamento della gara.

2.2 Descrizione della rilevanza gestionale del problema

Dal punto di vista gestionale, la rilevanza sta nella scelta dei giocatori in base a determinati parametri: la scelta dei giocatori è spesso oggetto di critiche in quanto è per buona parte di competenza del capitano della squadra. Tramite questo programma è possibile scegliere i giocatori secondo un criterio oggettivo che garantisce la scelta dei giocatori più in forma sulla base di dati relativi alle performance individuali dei giocatori, quali la media score e i guadagni incassati durante l'anno. Inoltre tramite questo programma è possibile simulare l'andamento generale del torneo, formulando una previsione del possibile risultato al variare del numero di apparizioni di ciascun giocatore.

2.3 Presentazione dei data-set per la valutazione

I data-set per la valutazione sono 5.
-Il primo è il database di ranking mondiale
-Il secondo include le medie dei risultati di ogni giocatore nell'anno corrente all'interno del tour americano.
-il terzo rappresenta una classifica globale sui guadagni dei giocatori americani. Il dataset contiene la cifra in $ dei guadagni di ogni giocatore e il numero di eventi a cui esso ha partecipato durante la stagione.

-il quarto e il quinto datasets sono i corrispettivi del secondo e del terzo ma provenienti da giocatori e tour europei.
Identifico le fonti dalle quali verranno tratti i dati utilizzati:
1 - https://www.owgr.com/current-world-ranking
2 - https://www.pgatour.com/stats/detail/120
3 - https://www.cbssports.com/golf/rankings/money-list/
4 - https://www.europeantour.com/dpworld-tour/stats/career-money-list/
5 - https://www.europeantour.com/dpworld-tour/stats/2023/leaderboard/?stats=SCORING&type=STROKE_AVERAGE

2.4 Descrizione preliminare degli algoritmi coinvolti

L'applicazione è principalmente basata su tre algoritmi: uno per selezionare le squadre, uno per creare il calendario e uno per simulare le partite. Il primo algoritmo delega a tre sotto-algoritmi il compito di "convocare" le due squadre, tra questi tre uno è basato sulla ricorsione ed è responsabile della scelta di una parte dei giocatori massimizzando i loro guadagni durante l'anno e rispettando il vincolo in input relativo al numero minimo di apparizioni che ciascun giocatore deve aver totalizzato durante l'anno. Per la creazione del calendario è stato utilizzato un algoritmo che crea le coppie per i primi due giorni di gara separatamente e con approcci diversi ed infine crea gli incontri singoli. La simulazione avviene attraverso la classe Simulator, nella quale viene attribuito un punteggio per ogni incontro, calcolato attraverso la simulazione strutturata sulle medie dei risultati di ciascun giocatore durante l'anno.

2.5 Descrizione preliminare delle funzionalità previste per l'applicazione software

L'applicazione sarà in grado attraverso una interfaccia semplice e dotata di campi testuali e bottoni di:
-selezionare i 24 giocatori componenti le due squadre tramite due pulsanti (selectTeamEUR e selectTeamUSA) e due campi testuali in cui l'utente inserisce un numero riferito a quante gare deve aver disputato ciascun giocatore e un altro numero che rappresenta la posizione massima nel ranking entro cui deve essere posizionato ciascun giocatore.
-creare un calendario che garantisca un approccio passivo nel primo giorno(consistente nella creazione di squadre omogenee) e più aggressivo negli ultimi due(consistente nella creazione di coppie randomiche) al fine di rendere la simulazione il più possibile combattuta ed aperta fino alla fine.
-simulare l'andamento dei tre giorni di gara attraverso tre bottoni (SimulateMatches) stampando i singoli risultati dei vari incontri in un'area di testo e il risultato finale della gara in una apposita griglia sulla destra giorno per giorno. (Nella stampa dei risultati ho messo la classica rappresentazione dei punteggi in UP e DOWN riferita ai giocatori europei: pertanto se il risultato è 3DOWN significa che il giocatore/coppia europea ha perso di 3 punti, se il risultato è 2UP significa che il giocatore/coppia europea ha vinto

di 2 punti, se il risultato è EVEN significa che il giocatore/coppia europea ha pareggiato con quella americana.)
-attraverso la selezione di un giocatore tramite una tendina e la pressione di un pulsante sarà possibile visualizzare tutti i risultati relativi agli incontri a cui esso ha partecipato.

CAPITOLO 3 - DESCRIZIONE DETTAGLIATA DEL PROBLEMA

3.1 L'importanza della scelta dei giocatori nelle squadre

L'attuale funzionamento della Ryder Cup prevede che nella selezione delle due squadre sei giocatori si qualifichino automaticamente in base a un ranking calcolato sui risultati annuali dei giocatori e al capitano spetti la selezione dei restanti sei giocatori. Talvolta le scelte di alcuni capitani si sono rivelate disastrose, per esempio Padraig Harrington, capitano della squadra europea nella Ryder Cup 2021, ha tenuto fuori Justin Rose (numero uno al mondo tra 2018 e 2019). Altri problemi possono verificarsi nelle scelte delle coppie che vengono schierate nei doppi; ne sono esempi Hal Sutton nel 2004 e Tom Watson nel 2014: entrambi i capitani hanno tenuto fuori dai doppi i giocatori più in forma che avevano con l'idea di schierarli nei singoli. Tali strategie furono causa di due sconfitte storiche per la squadra americana.
Attraverso gli algoritmi impiegati in questo software è possibile selezionare due squadre che comprendano in automatico i migliori sei giocatori nel world ranking e altri 6 giocatori ottenuti ottimizzando profitti e medie dei punteggi durante l'anno; inoltre tutti e 12 i giocatori devono aver partecipato ad un certo numero di gare scelto dall'utente. Il numero di gare scelto dall'utente serve per garantire che tra i giocatori selezionati in base alla media non ci siano giocatori che hanno partecipato a meno gare di quelle scelte: è infatti da tenere in considerazione che la media di un giocatore che ha partecipato a poche gare non è una media veritiera in quanto può cambiare radicalmente con un singolo risultato in più, mentre la media di un giocatore che ha partecipato a tante gare è decisamente più consolidata e rappresenta in maniera più veritiera una possibile performance di tale giocatore. Tutte queste considerazioni, unite con l'inserimento del ranking minimo per la selezione portano alla certezza sullo stato di forma dei giocatori schierati.
Inoltre attraverso un ulteriore algoritmo è possibile realizzare un calendario in cui le formazioni sono schierate in base a un criterio efficiente. Le coppie vengono schierate in modo tale da ottenere squadre il più possibile omogenee il primo giorno, mischiando i giocatori "più forti" con quelli "meno forti"(simulando un approccio conservativo per non correre rischi nei primi 8 doppi) e squadre più aggressive al secondo e al terzo giorno, mantenendo invariate per il giorno 2 soltanto le coppie che hanno portato una vittoria al giorno 1.

3.2 Input, output, criticità, potenzialità, e rilevanza del caso scelto

Per il funzionamento dell'applicazione è necessario inserire tre **input**:
(1)-> Un numero intero in un campo testuale rappresentante il numero di apparizioni minime che ciascun giocatore deve aver effettuato durante l'anno in tornei ufficiali. La selezione avviene solo tra i giocatori che hanno superato il numero di apparizioni minime inserite.
(2)-> Un numero intero in un campo testuale che rappresenta il numero massimo di posizione nel ranking mondiale oltre il quale non considero più i giocatori per selezionare le squadre.
(3)-> Un giocatore attraverso una combo box che permette di selezionare un giocatore tra quelli convocati alla gara.
Gli **output** sono divisi su cinque aree testuali:
(1)->Due aree contengono gli output relativi agli algoritmi di selezione delle squadre, nonché le liste dei giocatori selezionati stampati a video.
(2)->Un'altra area è stata creata per contenere il calendario e i risultati dei singoli incontri
(3)->Un campo testuale contenente il risultato finale della partita
(4)->Un'ultima area testuale nella quale verranno stampati tutti i risultati relativi agli incontri del giocatore selezionato in precedenza
Generalmente le criticità maggiori si creano nel momento in cui i giocatori sono scelti o schierati in base a criteri approssimativi o non oggettivi. Siccome metà dei giocatori sono scelti esclusivamente dal capitano è possibile che per errori di valutazione o per qualsiasi motivo uno o più giocatori meritevoli di partecipare alla squadra vengano tagliati fuori, oppure che i giocatori selezionati vengano schierati in maniera errata. Attraverso questa applicazione è possibile per il capitano avere un'idea di quale formazione selezionare e schierare ottenendo anche una possibile previsione del risultato attraverso la simulazione.

CAPITOLO 4 - DESCRIZIONE DEL DATA-SET UTILIZZATO PER L'ANALISI

4.1 Descrizione dei data-sets

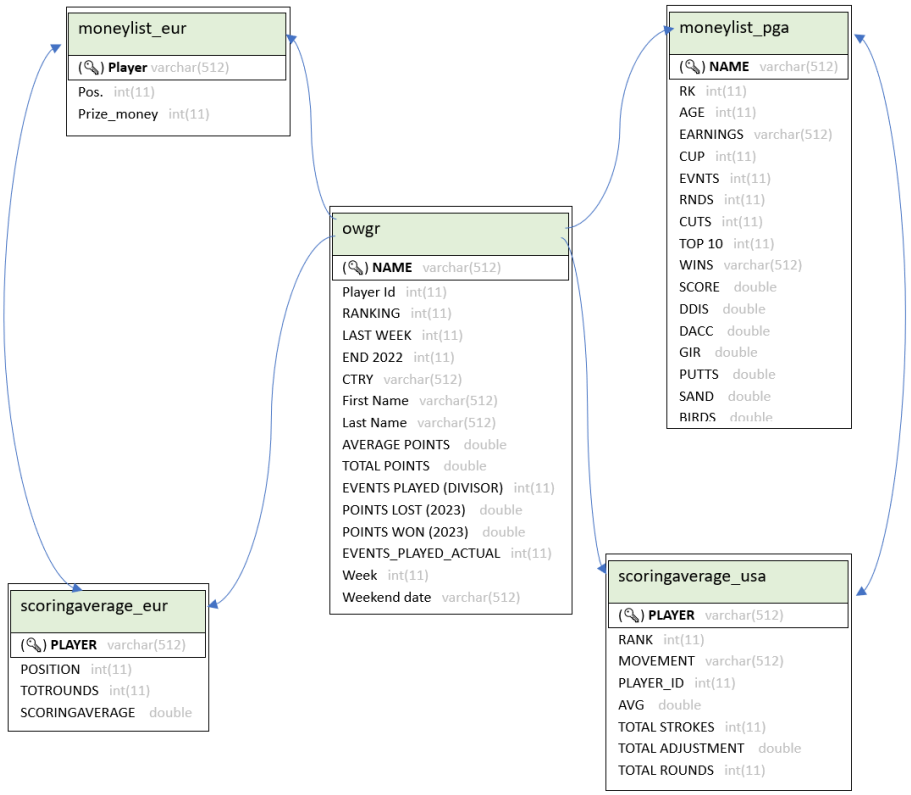
I data-set per la valutazione sono 5.
(1)-**owgr**: E' il dataset di ranking mondiale: viene usato per selezionare 6 dei 12 giocatori componenti ciascuna squadra.
(2)-**scoringaverage_usa:** Il dataset include le medie dei risultati di ogni giocatore nell'anno corrente all'interno del tour americano e serve per decretare i risultati della partita simulata.
(3)- **moneylist_pga** il dataset rappresenta una classifica globale sui guadagni dei giocatori americani. Il dataset contiene la cifra in $ dei guadagni di ogni giocatore e il

numero di eventi a cui esso ha partecipato durante la stagione. Tale valore è stato utilizzato per decretare gli altri 3 giocatori su 12 attraverso un algoritmo ricorsivo.
(4)- **scoringaverage_eur: :** il dataset contiene la classifica globale relativa alle medie dei punteggi dei giocatori europei
(5)- **moneylist_eur:** il dataset contiene la classifica globale relativa ai guadagni dei giocatori europei
Questi 5 dataset vedono al centro la tabella del ranking mondiale, la quale è in relazione alle altre tabelle in base al nome del giocatore (chiave primaria). Nel database di ranking mondiale sono presenti sia giocatori europei che americani, mentre le altre coppie di tabelle (scoringaverage e moneylist) sono relative ai singoli campionati (europei e americani).
Identifico le fonti dalle quali verranno tratti i dati utilizzati:
https://www.owgr.com/current-world-ranking
https://www.pgatour.com/stats/detail/120
https://www.cbssports.com/golf/rankings/money-list/
https://www.europeantour.com/dpworld-tour/stats/career-money-list/
https://www.europeantour.com/dpworld-tour/stats/2023/leaderboard/?stats=SCORING&type=STROKE_AVERAGE

4.2 Modifiche apportate ai data-sets

Alcuni datasets hanno richiesto la rimozione di colonne che rappresentavano la nazionalità dei giocatori attraverso immagini delle bandiere, causando errori nelle importazioni e nella gestione delle tabelle, altri invece hanno richiesto la modifica di alcuni caratteri nei nomi delle righe (ad esempio alcune contenevano spazi) al fine di poter strutturare in maniera efficiente le query.

4.3 Diagramma ER



CAPITOLO 5 - DESCRIZIONE DI STRUTTURE DATI E ALGORITMI UTILIZZATI

5.1 Descrizione strutture dati

L'applicazione è stata realizzata in linguaggio Java e segue i pattern ModelViewController(MVC) e il pattern DataAccessObject(DAO).
Per la creazione dell'interfaccia grafica è stata utilizzata l'applicazione SceneBuilder.
Il progetto è strutturato in tre packages:
it.polito.tdp.RyderCupSimulator: in questo package sono presenti tre classi : una classe Main la quale permette di far girare l'applicazione attraverso un comando run e le classi EntryPoint e FXMLController, le quali contengono i metodi per creare gli output da stampare e collegarli con l'interfaccia utente.
it.polito.tdp.RyderCupSimulator.dao: in questo package sono presenti due classi: la classe DBConnect, che permette attraverso una connessione alla HikariDataSource di collegare il progetto al database e la classe RyderCupDAO, nella quale sono presenti tutti i metodi in grado di strutturare le istruzioni di query da compiere al fine di importare i valori presenti nel database nelle strutture dati adeguate.
it.polito.tdp.RyderCupSimulator.model: in questo package sono presenti otto classi: la classe Model, nella quale è relegata la parte algoritmica, le classi relative agli oggetti [giocatori(classe Player) e partite(classi MatchSingolo e MatchDoppio)]e le classi relative alla simulazione(Simulator, Evento, SimResult).

5.2 Descrizione algoritmi

Gli algoritmi coinvolti saranno principalmente 7:
(1) -ALGORITMO PER SELEZIONARE LE SQUADRE

```
public List<Player> calcolaTeamUSA(Integer nMinA) {//alleggerisco la ricorsione inserendo per ciascun team di default
    this.salarioMaggiore = 0.0;
    List<Player> rimanenti = new ArrayList<>(this.loadPlayersUSA(nMinA));
    List<Player> parziale = new ArrayList<>();
    parziale.add(rimanenti.get(0));
    parziale.add(rimanenti.get(1));
    parziale.add(rimanenti.get(2));
    parziale.add(rimanenti.get(3));
    parziale.add(rimanenti.get(4));
    parziale.add(rimanenti.get(5));
    //parziale.add(rimanenti.get(6));
    rimanenti.remove(parziale.get(0));
    rimanenti.remove(parziale.get(1));
    rimanenti.remove(parziale.get(2));
    rimanenti.remove(parziale.get(3));
    rimanenti.remove(parziale.get(4));
    rimanenti.remove(parziale.get(5));
    //rimanenti.remove(parziale.get(6));

    ricorsioneUSASalario(parziale, rimanenti, nMinA);
    List<Player>last3Players = new ArrayList<>(calcolaTrePlayerPerMediaUSA(parziale, rimanenti, nMinA));
    this.teamUSA.addAll(last3Players);
    return this.teamUSA;
}
```

Questa prima parte di algoritmo seleziona i primi 6 giocatori per ranking, chiama il metodo ricorsivo per effettuare la scelta di altri 3 giocatori sulla base degli incassi degli

stessi e delega ad un altro metodo la selezione degli ultimi tre giocatori in base alla media dei punteggi durante l'anno. Il metodo ritorna una lista di giocatori di lunghezza 12 rappresentante la formazione americana. Un metodo identico è stato utilizzato anche per la selezione dei 12 componenti della formazione europea.

```
private void ricorsioneUSASalario(List<Player> parziale, List<Player> rimanenti, Integer nMinA){
    // Condizione Terminale
    if (parziale.size() == 9) {
        //calcolo totIncassi
        double salario = getSalarioTeam(parziale);
        if (salario > this.salarioMaggiore) {
            this.salarioMaggiore = salario;
            this.teamUSA = new ArrayList<Player>(parziale);
        }
            return;
    }
    for (Player p : rimanenti) {
        List<Player> currentRimanenti = new ArrayList<>(rimanenti);
            parziale.add(p);
            currentRimanenti.remove(p);
            ricorsioneUSASalario(parziale, currentRimanenti, nMinA);
            parziale.remove(parziale.size()-1);
    }
}

private List<Player> calcolaTrePlayerPerMediaUSA(List<Player>parziale, List<Player> rimanenti, Integer nMinA){
    List<Player>ultimiPlayers = new ArrayList<>();
    rimanenti.removeAll(teamUSA);
    List<Player>rimanentiOrdinati = new ArrayList<>(rimanenti);
    Collections.sort(rimanentiOrdinati);
    Integer i = 0;
    while(ultimiPlayers.size()<3){
        if(!teamUSA.contains(rimanentiOrdinati.get(i))) {
            ultimiPlayers.add(rimanentiOrdinati.get(i));
            i++;
        }
    }
    return ultimiPlayers;
}
```

Il metodo ricorsioneUSASalario(…)compie la scelta dei tre giocatori attraverso la ricorsione: esso ha una prima parte racchiusa in un if() all'interno della quale è contenuta la condizione terminale, ovvero che la dimensione della squadra selezionata sia complessivamente lunga 9(ossia che ai 6 giocatori presenti in essa siano stati aggiunti tre giocatori massimizzando il salario).
-Il metodo calcolaTrePlayersPerMedia(…) invece cerca nella lista di player ancora disponibili per la selezione(chiamata rimanenti) i tre giocatori che hanno la media di colpi a giro più bassa attraverso l'utilizzo del metodo Collections.sort. In parallelo a questo ho implementato il metodo compareTo nella classe player che ordina i giocatori in ordine crescente di media score(dal più basso al più alto).

(2) - ALGORITMO PER CREAZIONE CALENDARIO DEL DAY1:

```
public void generaCalendarioDay1() {//metto: 1+12, 2+11, ... e infine i migliori 2 per squadra giocano 2 ma
    this.matchesDay1 = new ArrayList<>();
    Player Player0EUR = this.teamEUR.get(0);
    Player Player1EUR = this.teamEUR.get(1);
    Player Player2EUR = this.teamEUR.get(2);
    Player Player3EUR = this.teamEUR.get(3);
    Player Player4EUR = this.teamEUR.get(4);
    Player Player5EUR = this.teamEUR.get(5);
    Player Player6EUR = this.teamEUR.get(6);
    Player Player7EUR = this.teamEUR.get(7);
    Player Player8EUR = this.teamEUR.get(8);
    Player Player9EUR = this.teamEUR.get(9);
    Player Player10EUR = this.teamEUR.get(10);
    Player Player11EUR = this.teamEUR.get(11);

    Player Player0USA = this.teamUSA.get(0);
    Player Player1USA = this.teamUSA.get(1);
    Player Player2USA = this.teamUSA.get(2);
    Player Player3USA = this.teamUSA.get(3);
    Player Player4USA = this.teamUSA.get(4);
    Player Player5USA = this.teamUSA.get(5);
    Player Player6USA = this.teamUSA.get(6);
    Player Player7USA = this.teamUSA.get(7);
    Player Player8USA = this.teamUSA.get(8);
    Player Player9USA = this.teamUSA.get(9);
    Player Player10USA = this.teamUSA.get(10);
    Player Player11USA = this.teamUSA.get(11);

    MatchDoppio m0 = new MatchDoppio(Player0EUR, Player11EUR, Player0USA, Player11USA, 0.0, 0.0, 0);
    MatchDoppio m1 = new MatchDoppio(Player1EUR, Player10EUR, Player1USA, Player10USA, 0.0, 0.0, 0);
    MatchDoppio m2 = new MatchDoppio(Player2EUR, Player9EUR, Player2USA, Player9USA, 0.0, 0.0, 0);
    MatchDoppio m3 = new MatchDoppio(Player3EUR, Player8EUR, Player3USA, Player8USA, 0.0, 0.0, 0);
    MatchDoppio m4 = new MatchDoppio(Player4EUR, Player7EUR, Player4USA, Player7USA, 0.0, 0.0, 0);
    MatchDoppio m5 = new MatchDoppio(Player5EUR, Player6EUR, Player5USA, Player6USA, 0.0, 0.0, 0);
    MatchDoppio m6 = new MatchDoppio(Player0EUR, Player3EUR, Player0USA, Player3USA, 0.0, 0.0, 0);
    MatchDoppio m7 = new MatchDoppio(Player1EUR, Player2EUR, Player1USA, Player2USA, 0.0, 0.0, 0);
    matchesDay1.add(m0);
    matchesDay1.add(m1);
    matchesDay1.add(m2);
    matchesDay1.add(m3);
    matchesDay1.add(m4);
    matchesDay1.add(m5);
    matchesDay1.add(m6);
    matchesDay1.add(m7);
}
```

il giorno 1 si giocano 8 incontri a coppie (essendo 12 i giocatori per squadra è necessario che quattro di essi giochino 2 partite in un giorno). Ordino la squadra in base alla posizione nel ranking mondiale e accoppio i giocatori in modo tale da ottenere le squadre più omogenee possibili [(1°+12°), (2°+11°), (3°+10°), (4°+9°) , (5°+8°) , (6°+7°) (i giocatori 1°, 2°, 3° e 4° inoltre giocheranno un secondo doppio in una giornata accoppiati nel seguente modo: (1°+4°) e (2°+3°)].

(3) - ALGORITMO PER CREAZIONE CALENDARIO DEL DAY2:

```
public void generaCalendarioDay2() {//qui creo accoppiamenti in base ai risultati del
    this.matchesDay2 = new ArrayList<>();
    List<CoppiaPlayers>vincentiEUR = new ArrayList<>();
    List<CoppiaPlayers>vincentiUSA = new ArrayList<>();
    List<Player>disponibiliEUR = new ArrayList<>();
    List<Player>disponibiliUSA = new ArrayList<>();
    List<Player>disponibiliEUR2 = new ArrayList<>();
    List<Player>disponibiliUSA2 = new ArrayList<>();

    List<CoppiaPlayers>coppieEUR = new ArrayList<>();
    List<CoppiaPlayers>coppieUSA = new ArrayList<>();

    for(Player p : this.teamEUR) {
        disponibiliEUR.add(p);
        disponibiliEUR2.add(p);
    }
    for(Player p : this.teamUSA) {
        disponibiliUSA.add(p);
        disponibiliUSA2.add(p);
    }
    //conto solo i primi sei match del day 1
    List<MatchDoppio>matchesSelezionati = new ArrayList<>();
    for(int i = 0; i<6; i++) {
        matchesSelezionati.add(this.risultatiDay1.get(i));
    }
    for(MatchDoppio x : matchesSelezionati) {
        //mantengo traccia delle coppie vincenti
        //le coppie che hanno vinto il 1 giorno le mantengo
        if(x.getRisultatoMatch() < 0) {
            Player p1 = x.getPlayer1EUR();
            Player p2 = x.getPlayer2EUR();
            CoppiaPlayers c = new CoppiaPlayers(p1, p2, "eur");
            vincentiEUR.add(c);
            coppieEUR.add(c);
            disponibiliEUR.remove(p1);
            disponibiliEUR.remove(p2);
        }
        if(x.getRisultatoMatch()>0) {
            Player p1 = x.getPlayer1USA();
            Player p2 = x.getPlayer2USA();
            CoppiaPlayers c = new CoppiaPlayers(p1, p2, "usa");
            vincentiUSA.add(c);
            coppieUSA.add(c);
            disponibiliUSA.remove(p1);
            disponibiliUSA.remove(p2);
        }
    }
        //per gli altri giocatori: cambio coppie in modo casuale
        while(!disponibiliEUR.isEmpty()) {//cosi ho creato 6 coppie per l'Europa
            Integer n0 = (int) (Math.random()*disponibiliEUR.size());
            Player p0EU = disponibiliEUR.get(n0);
            disponibiliEUR.remove(p0EU);
            Integer n1 = (int) (Math.random()*disponibiliEUR.size());
            Player p1EU = disponibiliEUR.get(n1);
            disponibiliEUR.remove(p1EU);
            CoppiaPlayers c = new CoppiaPlayers(p0EU, p1EU, "eur");
            coppieEUR.add(c);
        }
        while(!disponibiliUSA.isEmpty()) {//cosi ho creato 6 coppie per l'America
            Integer nu0 = (int) (Math.random()*disponibiliUSA.size());
            Player p0US = disponibiliUSA.get(nu0);
            disponibiliUSA.remove(p0US);
            Integer nu1 = (int) (Math.random()*disponibiliUSA.size());
            Player p1US = disponibiliUSA.get(nu1);
            disponibiliUSA.remove(p1US);
            CoppiaPlayers c = new CoppiaPlayers(p0US, p1US, "usa");
            coppieUSA.add(c);
        }


for(int i = 0; i<6; i++) {//creo i primi 6 matches
    Player p1EUR = coppieEUR.get(i).getPlayer1();
    Player p2EUR = coppieEUR.get(i).getPlayer2();
    Player p1USA = coppieUSA.get(i).getPlayer1();
    Player p2USA = coppieUSA.get(i).getPlayer2();
    MatchDoppio m = new MatchDoppio(p1EUR, p2EUR, p1USA, p2USA, 0.0, 0.0, 0);
    matchesDay2.add(m);

}
//ora devo creare ancora 2 coppie per ciascuna squadra(infatti 4 giocatori per s
disponibiliEUR2.remove(0);
disponibiliEUR2.remove(1);
disponibiliEUR2.remove(2);
disponibiliEUR2.remove(3);
disponibiliUSA2.remove(0);
disponibiliUSA2.remove(1);
disponibiliUSA2.remove(2);
disponibiliUSA2.remove(3);
for(int i = 0; i<2;  i++) {
    Integer n0 = (int) (Math.random()*disponibiliEUR2.size());
    Player p0EU = disponibiliEUR2.get(n0);
    disponibiliEUR2.remove(p0EU);
    Integer n1 = (int) (Math.random()*disponibiliEUR2.size());
    Player p1EU = disponibiliEUR2.get(n1);
    disponibiliEUR2.remove(p1EU);

    Integer nu0 = (int) (Math.random()*disponibiliUSA2.size());
    Player p0US = disponibiliUSA2.get(nu0);
    disponibiliUSA2.remove(p0US);
    Integer nu1 = (int) (Math.random()*disponibiliUSA2.size());
    Player p1US = disponibiliUSA2.get(nu1);
    disponibiliUSA2.remove(p1US);
    MatchDoppio m = new MatchDoppio(p0EU, p1EU, p0US, p1US, 0.0, 0.0, 0);
    matchesDay2.add(m);
}
```

Nel secondo giorno devo creare nuovamente 8 coppie di players. Per fare in modo che le squadre fossero schierate nella maniera più efficace possibile ho mantenuto invariate le sole coppie che hanno vinto nei primi 6 incontri al giorno 1 e ho modificato quelle che hanno perso o pareggiato: In tal modo ho formato le coppie per i primi 6 incontri. In seguito ho creato le restanti 4 coppie(2 per squadra) con l'idea di escludere da tali coppie i giocatori che hanno già disputato due incontri al giorno 1, al fine di non stancarli troppo per la fase finale.

(4) - ALGORITMO PER CREAZIONE CALENDARIO DEL DAY3:

```
public void generaCalendarioDay3() {
    this.matchesDay3 = new ArrayList<>();
    List<Player>squadraEUR = new ArrayList<>(this.teamEUR);
    List<Player>squadraUSA = new ArrayList<>(this.teamUSA);
    while(!squadraEUR.isEmpty()) {

        Integer n0 = (int) (Math.random()*squadraEUR.size());
        Player p0EU = squadraEUR.get(n0);
        squadraEUR.remove(p0EU);

        Integer nu0 = (int) (Math.random()*squadraUSA.size());
        Player p0US = squadraUSA.get(nu0);
        squadraUSA.remove(p0US);

        MatchSingolo m = new MatchSingolo(p0EU, p0US, 0.0, 0.0, 0);
        matchesDay3.add(m);
    }

}
```

il terzo giorno è riservato a 12 incontri singoli: creo accoppiamenti in modo casuale (attraverso la funzione Math.random()), al fine di alzare la variabilità della simulazione.

(5) - ALGORITMO PER CALCOLO RISULTATI INCONTRI SINGOLI (Day3):

```
public void initialize() {
    this.queue = new PriorityQueue<Evento>();
    this.risultatiDay3 = new ArrayList<>();

    //eventi matchDay1
    for (MatchSingolo x : this.calendarioDay3) {
        Double scoreEUR = (x.getPlayerEUR().getMediaScore()+Math.random()*3-Math.random()*5);
        Double scoreUSA = (x.getPlayerUSA().getMediaScore()+Math.random()*3-Math.random()*5);
        Integer punteggioMatch = (int) (scoreEUR-scoreUSA);
        x.setScorePlayerEUR(scoreEUR);
        x.setScorePlayerUSA(scoreUSA);
        x.setPunteggio(punteggioMatch);
        this.queue.add(new Evento(EventType.MATCHSINGOLO, 3, x.getPlayerEUR(), x.getPlayerUSA(), null, null, scoreEUR, scoreUSA, punteggioMatch));
        this.risultatiDay3.add(x);
    }

}
```

```
public void run() {
    this.puntiEUR = 0.0;
    this.puntiUSA = 0.0;
    while(!queue.isEmpty()) {
        Evento e = queue.poll();
        switch(e.getType()) {
        case MATCHDOPPIO:
            System.out.print("(Day:"+e.getDay()+"): ["+e.getPlayer1().getNome()+ e.getPlayer1().getCognome()+"+"+ e.getPlayer2().getNome()+e.getPlayer2().getCognome()+"] vs"+" ["+
        e.getPlayer3().getNome()+e.getPlayer3().getCognome()+"+"+ e.getPlayer4().getNome()+e.getPlayer4().getCognome()+"] result: "+e.getPunteggio()+"\n");
            if(e.getPunteggio()>0) {
                this.puntiUSA += 1.0;
            }
            if(e.getPunteggio()<0) {
                this.puntiEUR += 1.0;
            }
            if(e.getPunteggio()==0) {
                this.puntiUSA += 0.5;
                this.puntiEUR += 0.5;
            }
            break;
        case MATCHSINGOLO:
            System.out.print("(Day:"+e.getDay()+"): "+e.getPlayer1().getNome()+ e.getPlayer1().getCognome()+" vs "+ e.getPlayer2().getNome()+e.getPlayer2().getCognome()+" result: "+e.getPunteggio()+"\n")
            if(e.getPunteggio()>0) {
                this.puntiUSA += 1.0;
            }
            if(e.getPunteggio()<0) {
                this.puntiEUR += 1.0;
            }
            if(e.getPunteggio()==0) {
                this.puntiUSA += 0.5;
                this.puntiEUR += 0.5;
            }
            break;
        default:
            break;
        }
    }
    System.out.println("punti EUR: "+this.puntiEUR+" puntiUSA: "+this.puntiUSA);
}
```

Gli incontri singoli vedono schierato un giocatore per parte: per decretare il punteggio dell'incontro simulato attribuisco a ciascun giocatore un punteggio simulato calcolato come media score del giocatore durante l'anno(colonna SCORINGAVERAGE della tabella scoringaverage_eur e colonna AVG della tabella scoringaverage_usa) a cui viene aggiunta una quantità casuale da 0 a 3 e sottratta una stessa quantità casuale da 0 a 5 in modo da avere una componente casuale che rappresenta il numero di colpi persi e guadagnati nel corso dell'incontro. Dalla differenza dei valori così calcolati attribuisco un risultato all'incontro.

(5b) -ALGORITMO PER CALCOLO RISULTATI INCONTRI DOPPI (Day1 e Day2):

```
public void initialize() {
    this.queue = new PriorityQueue<Evento>();
    this.risultatiDay1 = new ArrayList<>();

    //eventi matchDay1
    for (MatchDoppio x : this.calendarioDay1) {
        Double scoreEUR = (x.getPlayer1EUR().getMediaScore()+x.getPlayer2EUR().getMediaScore())/2+Math.random()*5-Math.random()*7;
        Double scoreUSA = (x.getPlayer1USA().getMediaScore()+x.getPlayer2USA().getMediaScore())/2+Math.random()*5-Math.random()*7;
        Integer punteggioMatch = (int) (scoreEUR-scoreUSA);
        x.setScoreEUR(scoreEUR);
        x.setScoreUSA(scoreUSA);
        x.setRisultatoMatch(punteggioMatch);
        this.queue.add(new Evento(EventType.MATCHDOPPIO, 1, x.getPlayer1EUR(), x.getPlayer2EUR(), x.getPlayer1USA(), x.getPlayer2USA(), scoreEUR, scoreUSA, punteggioMatch));
        this.risultatiDay1.add(x);
    }

}
```

```
  public void run() {
      this.puntiEUR = 0.0;
      this.puntiUSA = 0.0;
      while(!queue.isEmpty()) {
          Evento e = queue.poll();
          switch(e.getType()) {
          case MATCHDOPPIO:
              System.out.print("(Day:"+e.getDay()+"): ["+e.getPlayer1().getNome()+ e.getPlayer1().getCognome()+"+"+ e.getPlayer2().getNome()+e.getPlayer2().getCognome()+"] vs"+" ["+
          e.getPlayer3().getNome()+e.getPlayer3().getCognome()+"+"+ e.getPlayer4().getNome()+e.getPlayer4().getCognome()+"] result: "+e.getPunteggio()+"\n");
              if(e.getPunteggio()>0) {
                  this.puntiUSA += 1.0;
              }
              if(e.getPunteggio()<0) {
                  this.puntiEUR += 1.0;
              }
              if(e.getPunteggio()==0) {
                  this.puntiUSA += 0.5;
                  this.puntiEUR += 0.5;
              }
              break;
          case MATCHSINGOLO:
              System.out.print("(Day:"+e.getDay()+"): "+e.getPlayer1().getNome()+ e.getPlayer1().getCognome()+" vs "+ e.getPlayer2().getNome()+e.getPlayer2().getCognome()+" result: "+e.getPunteggio()+"\n")
              if(e.getPunteggio()>0) {
                  this.puntiUSA += 1.0;
              }
              if(e.getPunteggio()<0) {
                  this.puntiEUR += 1.0;
              }
              if(e.getPunteggio()==0) {
                  this.puntiUSA += 0.5;
                  this.puntiEUR += 0.5;
              }
              break;
          default:
              break;
          }
      }
      System.out.println("punti EUR: "+this.puntiEUR+" puntiUSA: "+this.puntiUSA);
  }
```

Per gli incontri doppi attribuisco lo stesso punteggio calcolato nel day 3 a ciascun giocatore (con una varianza maggiore) e calcolo la media dei punteggi dei giocatori appartenenti alla stessa squadra: dal confronto di tali medie calcolo il risultato dell'incontro.
Inoltre per ogni incontro (sia singolo che doppio) è assegnato un punto, dalla cui somma si ha il risultato finale della giornata: in caso di vittoria assegno un punto alla squadra, in caso di sconfitta ne assegno zero e se l'incontro termina in pareggio vengono assegnati 0.5 punti a testa.

(6) - ALGORITMO PER SELEZIONARE A VIDEO I RISULTATI DEL GIOCATORE SCELTO:

```
public String satsPlayer(String player) {
    Player p = this.idMapPlayers.get(player);
    String risultati = player + "'s results:\n";
    List<MatchSingolo>singoli = new ArrayList<>(risultatiDay3);
    List<MatchDoppio>doppi1 = new ArrayList<>(risultatiDay1);
    List<MatchDoppio>doppi2 = new ArrayList<>(risultatiDay2);
    List<MatchDoppio>doppi = new ArrayList<>();
    doppi.addAll(doppi1);
    doppi.addAll(doppi2);

    for(MatchSingolo x : singoli) {
        if(x.toString().contains(player) /*x.getPlayerEUR().equals(p) || x.getPlayerUSA().equals(p)*/) {
            String risMatch = "";
            if(x.getPunteggio()<0) {
                Integer delta = -x.getPunteggio();
                risMatch += delta+" UP (EU)";
            }
            if(x.getPunteggio()==0) {
                risMatch += " EVEN";
            }
            if(x.getPunteggio()>0) {
                risMatch += x.getPunteggio()+" DOWN (EU)";
            }
            risultati += x.toStringSenzaACapo()+" "+risMatch+"\n";
        }
    }
    for(MatchDoppio x : doppi) {
        if(x.toString().contains(player)) {
            String risMatch = "";
            if(x.getRisultatoMatch()<0) {
                Integer delta = -x.getRisultatoMatch();
                risMatch += delta+" UP (EU)";
            }
            if(x.getRisultatoMatch()==0) {
                risMatch += "EVEN";
            }
            if(x.getRisultatoMatch()>0) {
                risMatch += x.getRisultatoMatch()+" DOWN (EU)";
            }
            risultati +=  x.toStringSenzaACapo()+" "+risMatch+"\n";
        }
    }
    return risultati;
}
```

Dato l'input relativo alla scelta un giocatore da parte dell'utente: si stampa a video l'elenco di tutti gli incontri disputati dal giocatore in modo da avere una rapida idea dell'apporto al punteggio finale dato dal giocatore scelto. Il metodo utilizza una ricerca testuale tra i risultati, stampando la lista di tutti quelli in cui il giocatore selezionato ha contribuito.

(7) -ALGORITMI PER LEGGERE DAL DATABASE LE INFORMAZIONI NECESSARIE:

```
public List<Player> getAllPlayersEUR(Integer n, Integer rank){
    String query = "SELECT * "
            + "FROM owgr o "
            + "WHERE o.CTRY IN ( "
            + "    'Albania', 'Andorra', 'England', 'Northern Ireland', 'Austria', 'Belarus', 'Belgium', 'Bosnia and Herzegovina', 'Bulgaria', 'Croatia', 'Cyprus',
            + ")"
            + "AND o.EVENTS_PLAYED_ACTUAL > ? "
            + "AND o.RANKING < ? "
            + "ORDER BY o.RANKING ";

    List<Player> result = new ArrayList<Player>();
    try {
        Connection conn = DBConnect.getConnection();
        PreparedStatement st = conn.prepareStatement(query);
        st.setInt(1, n);
        st.setInt(2, rank);
        ResultSet rs = st.executeQuery();

        while (rs.next()) {
            String nome = rs.getString("First Name");
            String cognome = rs.getString("Last Name");
            String nazione = rs.getString("CTRY");
            Integer nApparizioni = rs.getInt("EVENTS_PLAYED_ACTUAL");
            Integer posizioneRanking = rs.getInt("RANKING");
            Integer totaleIncassiAnno = this.getTotaleIncassiEUR(nome, cognome);
            if(totaleIncassiAnno == 0) {
                totaleIncassiAnno = this.getTotaleIncassiUSA(nome, cognome);
            }
            Double mediaScore = this.getMediaScoreEUR(nome, cognome);
            if(mediaScore == 0.0) {
                mediaScore = this.getMediaScoreUSA(nome, cognome);
            }
            Player p = new Player(nome, cognome, nazione, nApparizioni, posizioneRanking, totaleIncassiAnno, mediaScore);
            result.add(p);
        }
        conn.close();
        return result;
    } catch (SQLException e) {
        e.printStackTrace();
        System.out.println("Errore connessione al database");
        throw new RuntimeException("Error Connection Database");
    }
}
```

Per selezionare dal database di ranking mondiale i soli giocatori europei ho strutturato una query che seleziona tutti e soli i giocatori la cui nazione (colonna CTRY) è

compresa tra gli Stati membri dell'Unione Europea (giocatori ordinati per ranking). Inoltre bisogna tener conto che ci sono giocatori europei che giocano su tour americani e viceversa: per questo motivo ho messo un if() sul totale incassi e sulla media dei giocatori nel tour europeo. Questo if() controlla che tali valori non siano nulli e, nel caso lo siano cerca di trovare il valore di tali attributi nelle apposite tabelle relative al tour americano.

```
public List<Player> getAllPlayersUSA(Integer n, Integer rank) {
    String query = "SELECT * "
            + "FROM owgr o "
            + "WHERE o.CTRY = 'United States' "
            + "AND o.EVENTS_PLAYED_ACTUAL > ? "
            + "AND o.RANKING < ? "
            + "ORDER BY o.RANKING ";
    List<Player> result = new ArrayList<Player>();
    try {
        Connection conn = DBConnect.getConnection();
        PreparedStatement st = conn.prepareStatement(query);
        st.setInt(1, n);
        st.setInt(2, rank);
        ResultSet rs = st.executeQuery();

        while (rs.next()) {
            String nome = rs.getString("First Name");
            String cognome = rs.getString("Last Name");
            String nazione = rs.getString("CTRY");
            Integer nApparizioni = rs.getInt("EVENTS_PLAYED_ACTUAL");
            Integer posizioneRanking = rs.getInt("RANKING");
            Integer totaleIncassiAnno = this.getTotaleIncassiUSA(nome, cognome);
            if(totaleIncassiAnno == 0) {
                totaleIncassiAnno = this.getTotaleIncassiEUR(nome, cognome);
            }
            Double mediaScore = this.getMediaScoreUSA(nome, cognome);
            if(mediaScore == 0.0) {
                mediaScore = this.getMediaScoreEUR(nome, cognome);
            }
            Player p = new Player(nome, cognome, nazione, nApparizioni, posizioneRanking, totaleIncassiAnno, mediaScore);
            result.add(p);
        }
        conn.close();
        return result;
    } catch (SQLException e) {
        e.printStackTrace();
        System.out.println("Errore connessione al database");
        throw new RuntimeException("Error Connection Database");
    }
}
```

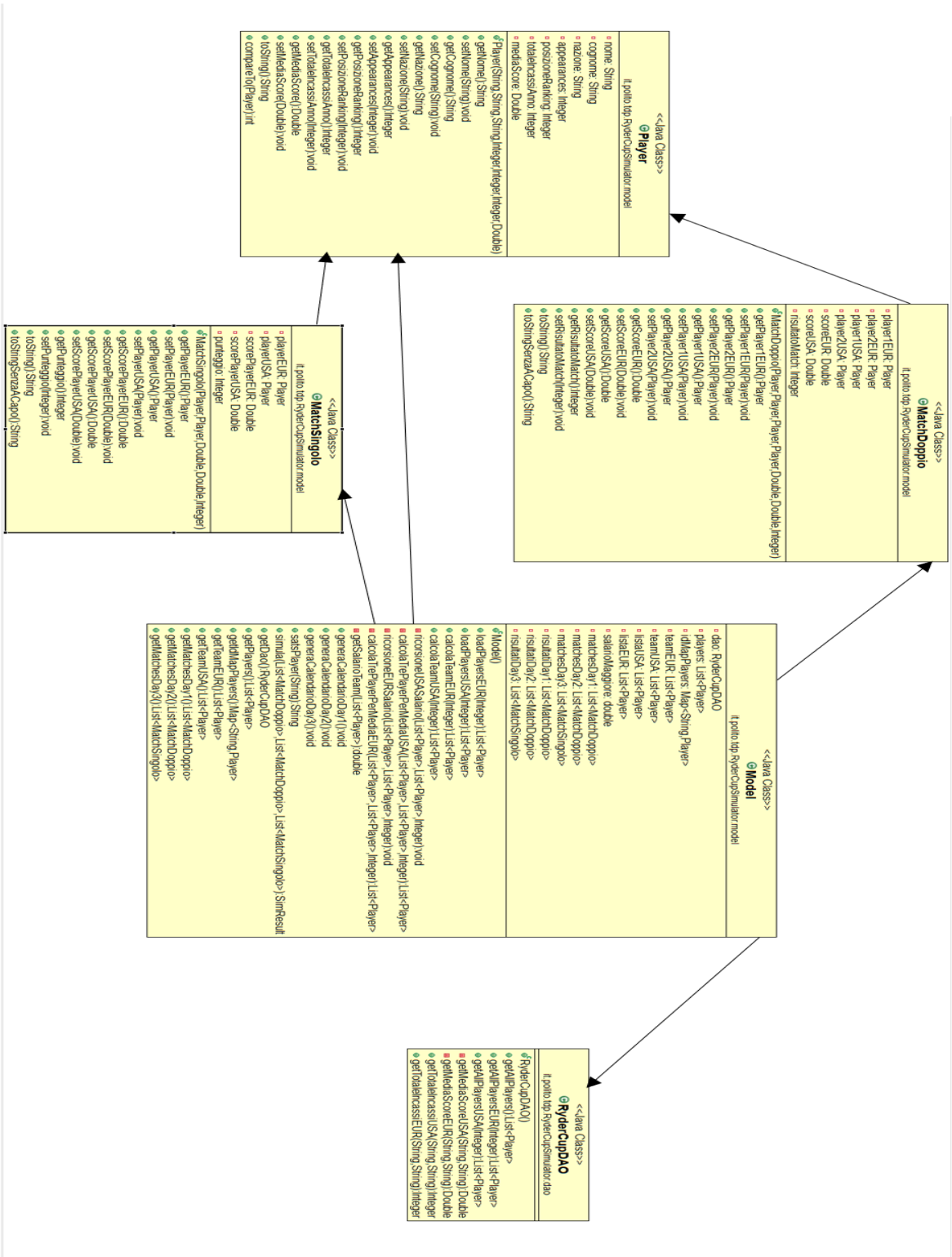
In modo analogo ho strutturato la query per selezionare i giocatori americani.

```
private Double getMediaScoreEUR(String nome, String cognome) {
    String query = "SELECT s.ScoringAverage "
            + "FROM scoringaverage_eur s "
            + "WHERE s.player = ? ";
    String fullName = cognome.toUpperCase()+nome;
    Double mediaScore = 0.0;
    try {
        Connection conn = DBConnect.getConnection();
        PreparedStatement st = conn.prepareStatement(query);
        st.setString(1, fullName);
        ResultSet rs = st.executeQuery();

        while (rs.next()) {
            mediaScore = rs.getDouble("ScoringAverage");
        }
        conn.close();
        return mediaScore;
    } catch (SQLException e) {
        e.printStackTrace();
        System.out.println("Errore connessione al database");
        throw new RuntimeException("Error Connection Database");
    }
}
```

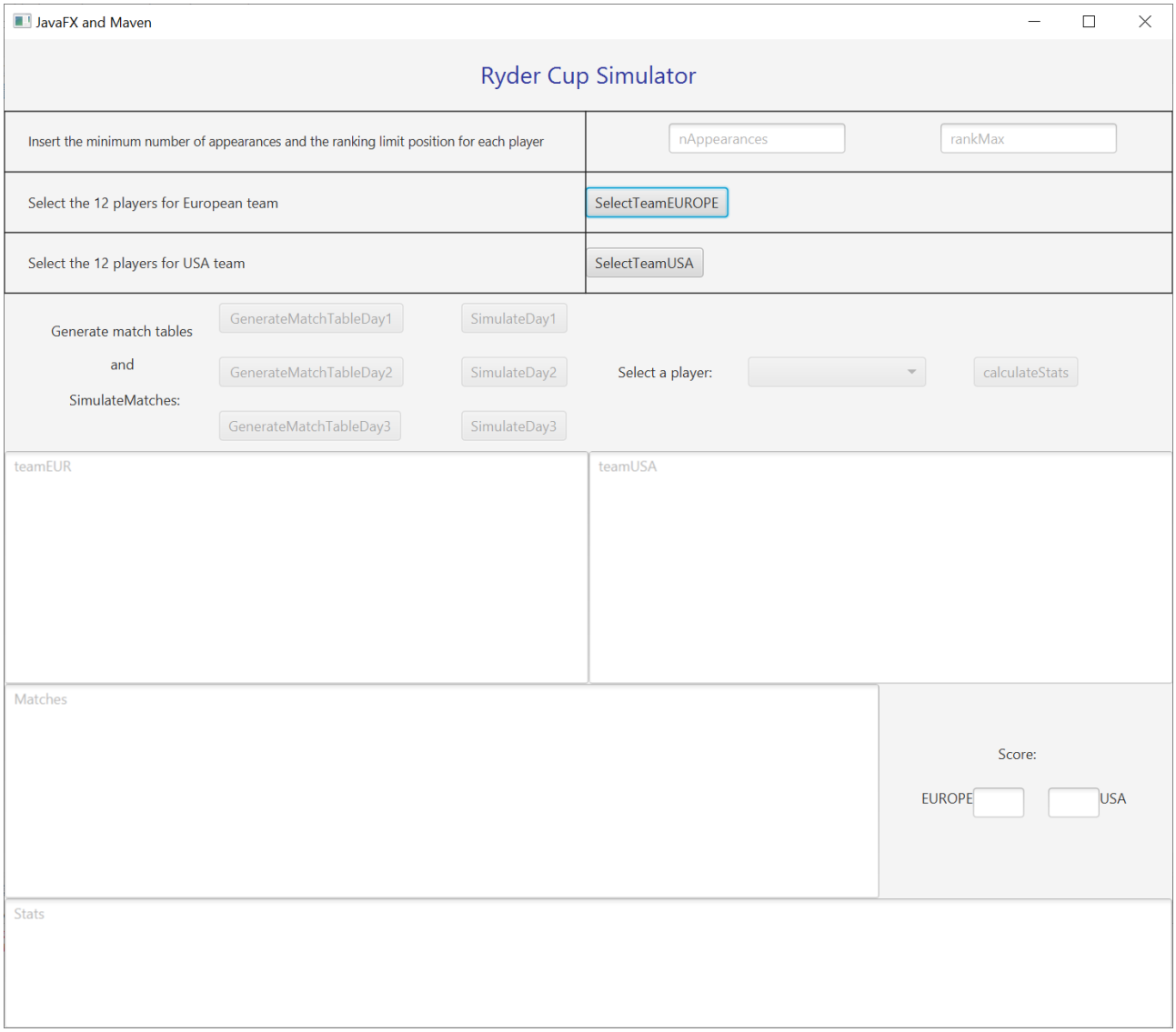
```
public Integer getTotaleIncassiUSA(String nome, String cognome){
    String query = "SELECT m.EARNINGS "
            + "FROM moneylist_pga m "
            + "WHERE m.NAME = ? ";

    String fullNameX = nome+cognome;
    String fullName = fullNameX.strip();
    Integer totIncassi = 0;
    try {
        Connection conn = DBConnect.getConnection();
        PreparedStatement st = conn.prepareStatement(query);
        st.setString(1, fullName);
        ResultSet rs = st.executeQuery();

        while (rs.next()) {
            String totaleIncassiAnno = rs.getString("EARNINGS");
            String totIncassiB = totaleIncassiAnno.substring(1, totaleIncassiAnno.length());
            totIncassi = Integer.parseInt(totIncassiB);

        }
        conn.close();
        return totIncassi;
    } catch (SQLException e) {
        e.printStackTrace();
        System.out.println("Errore connessione al database");
        throw new RuntimeException("Error Connection Database");
    }
}
```

Questi due metodi invece leggono media score e totale incassi dalle apposite tabelle per ciascun giocatore al fine di poter inserire tali valori nel metodo precedente, salvandoli negli attributi relativi alla classe Player.

CAPITOLO 6 - DIAGRAMMA DELLE CLASSI DELLE PARTI PRINCIPALI DELL'APPLICAZIONE

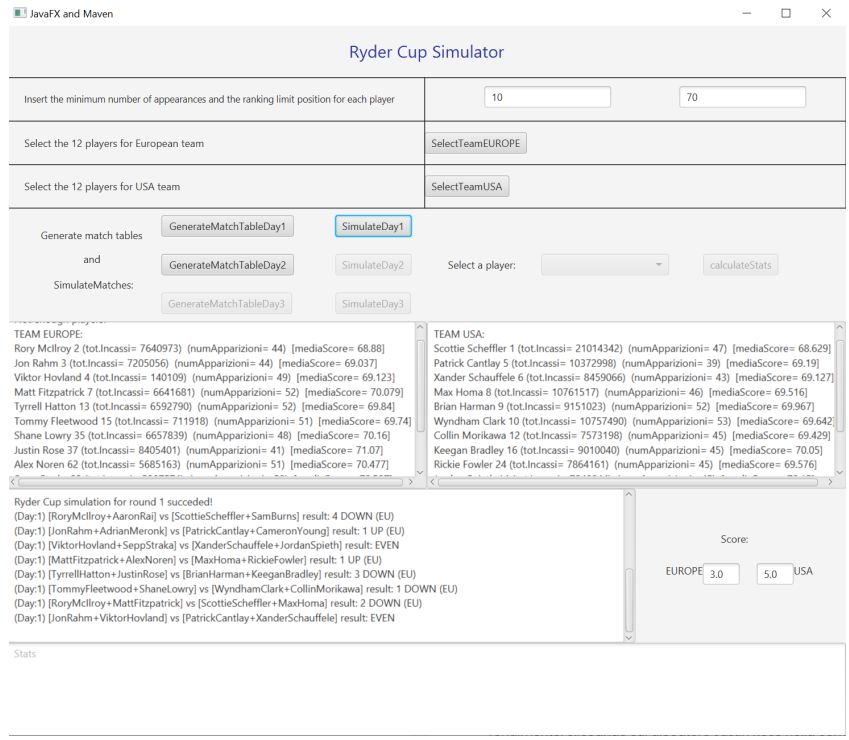


CAPITOLO 7 - VIDEATE DELL'APPLICAZIONE E LINK DEL VIDEO

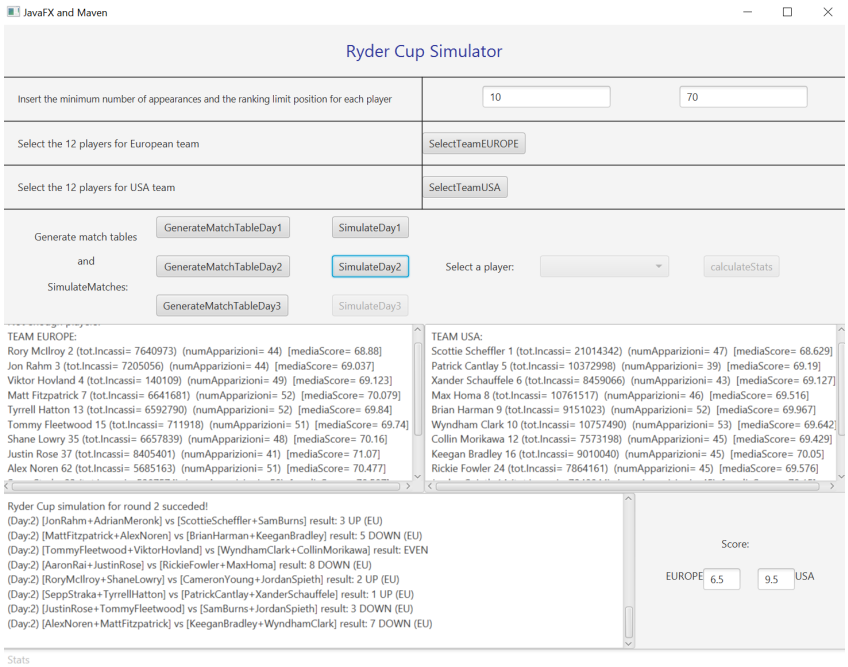


Per un corretto funzionamento dell'applicazione occorre compiere i seguenti passaggi:
1-Inserire un numero intero (numero minimo di gare a cui ogni giocatore deve aver partecipato) e un numero intero (posizione del ranking entro la quale selezionare i giocatori)
2-selezionare le due squadre(tramite i due appositi bottoni)
3-creare calendario(bottone GenerateMatchTableDay1) e simulare le partite(bottone Simulate Day1) ed osservare il punteggio del giorno 1
4- creare calendario(bottone GenerateMatchTable Day2) e simulare le partite(bottone SimulateDay2) ed osservare il punteggio del giorno 2
5- creare calendario(bottone GenerateMatchTable Day3) e simulare le partite(bottone Simulate Day3) ed osservare il punteggio del giorno 3
6-selezionare un giocatore dalla tedina e stampare i risultati relativi ad esso tramite il bottone calculateStats.
-LINK DEL VIDEO DIMOSTRATIVO:
https://youtu.be/-ePZJKeEW9o

CAPITOLO 8 - TABELLE CON RISULTATI OTTENUTI

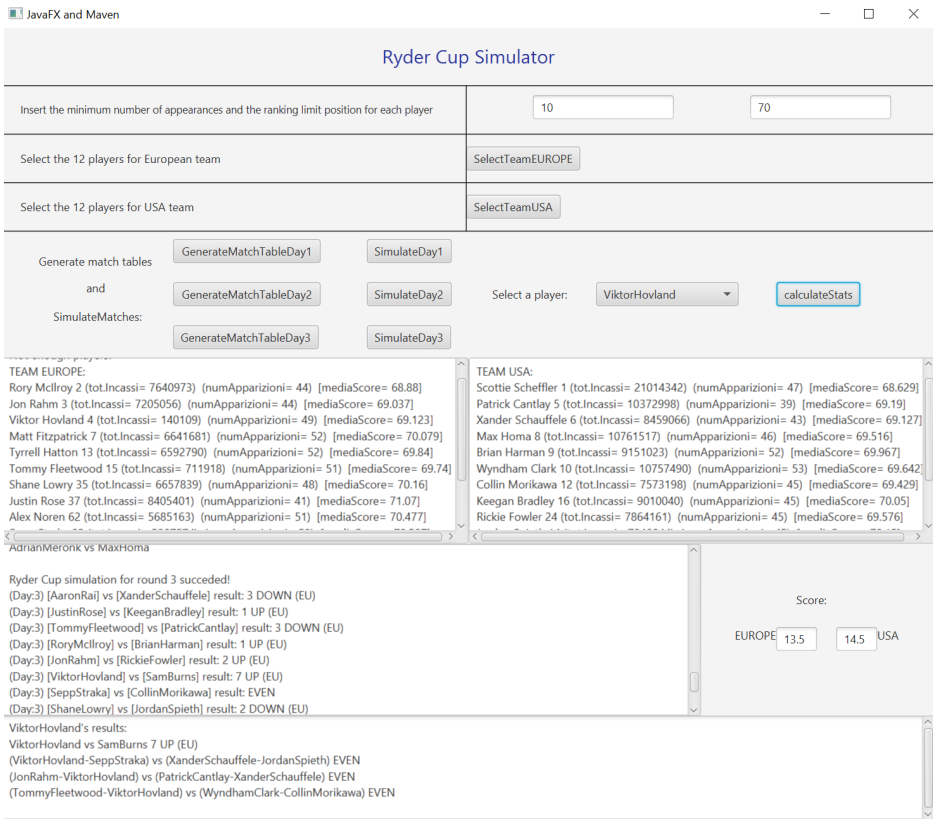


(2)



Ecco alcuni esempi di output: ho selezionato giocatori che hanno totalizzato almeno 10 apparizioni durante l'anno corrente e che sono nelle prime 70 posizioni del ranking mondiale: **(1)** L'Europa è in svantaggio 3 a 5 punti alla fine del giorno 1. Nel giorno **(2)** invece il parziale è diventato 6,5 a 9,5: la squadra americana ha aumentato il suo vantaggio, staccando di 3 punti la squadra europea.

(3)



Nel giorno **(3)** (giorno conclusivo dell'evento) il risultato finale è di 13,5 a 14,5 per la squadra americana. A questo punto posso selezionare un giocatore per analizzarne il rendimento: cliccando sul giocatore Viktor Hovland nella comboBox ho osservato che esso ha partecipato a 4 incontri vincendo lo scontro singolo con Sam Burns per 7 UP e pareggiando i tre doppi a cui ha partecipato(EVEN).

CAPITOLO 9 - VALUTAZIONI SUI RISULTATI OTTENUTI E CONCLUSIONI

9.1 Valutazioni sui risultati ottenuti

I risultati ottenuti sono da intendere come lo svolgimento di uno dei possibili scenari di gara e godono inoltre di una buona verosimiglianza. La simulazione delle partite genera risultati diversi anche per input uguali, infatti, nonostante l'attribuzione dei punteggi sia basata sulle medie dei risultati di ciascun giocatore durante l'anno, vi è una forte componente casuale data dalla stima di un certo numero di bogeys e birdies (colpi persi e guadagnati) che fanno discostare il risultato dalla media di ciascun giocatore. Tale accorgimento fa sì che la simulazione sia una stima verosimile di un possibile andamento della Ryder Cup. Inoltre simulando più volte risulta evidente che la squadra americana tende a vincere più spesso: infatti, malgrado la vittoria di quest'anno (2023) della squadra europea, al momento la formazione americana resta la favorita.

9.2 Conclusioni

Gli obiettivi del programma sono soddisfatti: il software è funzionante ed è in grado di simulare in modo verosimile l'andamento dei tre giorni di gara.
Sarebbe possibile migliorare la logica applicativa attraverso l'impiego di algoritmi più sofisticati: ad esempio tenendo in conto più parametri per il calcolo dei risultati degli incontri. Infatti se nella determinazione di tale calcolo tenessi traccia dei risultati colpo per colpo otterrei una simulazione ancora più verosimile e dettagliata nello svolgimento delle partite e potrei inserire anche un fattore pressione, basandomi su statistiche di rendimento dei giocatori in momenti di pressione (la pressione potrebbe entrare in gioco nelle fasi finali dell'incontro).
Implementando una simulazione colpo per colpo sarebbe inoltre possibile differenziare gli incontri doppi in foursomes e four-balls, calcolando i risultati degli incontri in modo diverso.
Infatti nei foursomes ogni squadra di due giocatori gioca una palla per buca, alternando un colpo a testa, finché ogni buca non è stata completata. I giocatori si alternano al colpo dal tee, uno andando per primo al tee delle buche pari, e l'altro per primo al tee delle buche dispari. La squadra con il punteggio più basso vince la buca e, se il punteggio è pari, la buca finisce in pareggio.
Nei four-balls invece ogni membro delle squadre di due giocatori gioca la sua palla, cosicché verranno giocate quattro palle a ogni buca, Ogni squadra considera il più basso tra i punteggi su ogni buca e la squadra il cui giocatore ha il punteggio più basso vince la buca e se i due punteggi più bassi sono pari, la buca finisce in pareggio.
Calcolando i risultati degli incontri in tale modo otterrei un risultato ancora più attendibile e potrei analizzare più nel dettaglio l'andamento della partita simulata, rendendo possibile attraverso più simulazioni consecutive la formulazione di una previsione molto verosimile.