Trabajo Práctico Nº3

Programación III

**Integrantes**

* Luraschi, Roberto Gino

Contenido

[Contenido 2](#_Toc516593819)

[Decisiones tomadas 3](#_Toc516593820)

[Decisiones tomadas a nivel de código de controladores 3](#_Toc516593821)

[Decisiones en algoritmo de busqueda 3](#_Toc516593822)

[Código de las clases 4](#_Toc516593823)

# Decisiones tomadas

A lo largo del desarrollo del trabajo práctico, se han tomado distintas decisiones, tanto a nivel de código funcional como del código visual.

A continuación se detallarán las decisiones tomadas en cada caso.

## Decisiones tomadas a nivel de código de controladores

En cuanto a nivel general de la aplicación se decidió tomar un diseño de MVC.

En el modelo nos encontramos con cada jugador con cada uno de sus detalles como posición, puntaje, tarjetas, goles y nacionalidad además del nombre con el cual se identifica al jugador.

La vista utilizara 2 formularios, uno llamado MainForm que al ser el formulario principal se encargará de mostrar listado de jugadores junto con sus puntajes. Estos jugadores pueden ser seleccionados desde la lista y pueden ser editados (Por cuestiones de tiempo no se alcanzó a hacer que se pueda editar las posiciones). El otro formulario es un formulario encargado de mostrar el equipo ideal con los jugadores en las posiciones en donde son los mejores. Al no poder elegir un jugador para una posición por algún impedimento, se pasa a mostrar la posición en blanco. Por último, este formulario muestra también el promedio de este equipo ideal y las restricciones alcanzadas, por las cuales se suponen no se eligió un equipo diferente.

En cuanto al controlador, este no posee una gran complejidad, debido a que solo tiene la lista de jugadores a la hora de ejecutarse y solamente decide en qué momento se guardaran para poder manipularse posteriormente. Además este controlador tiene un método para buscar jugadores en su lista y un método que llamara a calcular el equipo ideal. También nos podemos encontrar con el controlador de Jugadores, el cual se encarga de persistir estos datos en un archivo Json.

## Decisiones en algoritmo de búsqueda

A la hora de buscar el equipo ideal se decidió por utilizar un algoritmo heurístico que no busque un resultado exacto, sino que logre encontrar un mejor equipo en un tiempo polinomial, en cual es determinado por la cantidad de posiciones. Esto se logró haciendo que se ordenaran los jugadores por su promedio, después se filtren posición por posición y se elija el mejor en la misma. Este algoritmo tiene una complejidad de ***O(m.n)*** donde m son las posibles posiciones y n son la cantidad de jugadores ingresados. Esté método devuelve un conjunto y recibe una instancia además de un Generador de números, que en el caso de la implementación principal de éste método se utiliza un generador aleatorio.

# Código de las clases

**Algoritmo de Controlador.java**

package controlador;

import java.util.ArrayList;

import java.util.List;

import algoritmos.Conjunto;

import algoritmos.Instancia;

import algoritmos.SolverGoloso;

import generadores.GeneradorAleatorio;

import modelo.Jugador;

public class Controlador {

//----------------Variables------------------

private List<Jugador> jugadores;

//----------------Constantes-----------------

static String PATH = "./src/datos.txt";

//----------------Contructores---------------

public Controlador(){

Jugadores archivo = new Jugadores();

this.jugadores = archivo.levantarDatos();

}

//----------------Getters--------------------

public Jugador[] obtenerJugadores(){

Jugador[] ret = new Jugador[jugadores.size()];

int x = 0;

for(Jugador j : jugadores){

ret[x] = j;

x++;

}

return ret;

}

private void guardarJugadores() {

Jugadores j = new Jugadores();

j.persistirDatos(jugadores);

}

//----------------Setters------------------

public void agregarJugador(Jugador j){

if(jugadores.contains(j))jugadores.remove(j);

jugadores.add(j);

guardarJugadores();

}

public Conjunto seleccionarEquipo() {

SolverGoloso solver =

new SolverGoloso(new Instancia((ArrayList<Jugador>) jugadores),

new GeneradorAleatorio());

Conjunto sol = solver.resolver();

return sol;

}

public Jugador obtenerJugador(String nombre) {

for(Jugador j : jugadores)

if(j.getNombre().equals(nombre))

return j;

return null;

}

}

**Algoritmo de SolverGoloso.java**

package algoritmos;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import enums.Posicion;

import generadores.Generador;

import modelo.Jugador;

public class SolverGoloso {

private Instancia instancia;

private Generador \_random;

public SolverGoloso(Instancia instancia, Generador generador) {

this.instancia = instancia;

\_random = generador;

}

private int guess(int t) {

int i = \_random.nextInt(t < 1 ? 1 : t);

return i;

}

public Conjunto resolver() {

Conjunto ret = new Conjunto();

List<Jugador> ordenados = ordenarJugadores();

List<Posicion> posicionesClone = new ArrayList<Posicion>();

for (Posicion posicion : Posicion.values())

posicionesClone.add(posicion);

while (!posicionesClone.isEmpty()) {

Posicion posicionActual = posicionesClone.get(guess(posicionesClone.size()));

List<Jugador> jugPorPosicion = new ArrayList<Jugador>();

ordenados.stream()

.filter(j -> j.juegaDe(posicionActual))

.forEach(j -> jugPorPosicion.add(j));

for (Jugador jugador : jugPorPosicion) {

if (puedeSerIncluido(jugador, ret)) {

ret.agregar(jugador, posicionActual);

break;

}

posicionesClone.remove(posicionActual);

}

}

return ret;

}

private boolean puedeSerIncluido(Jugador jugador, Conjunto conjunto) {

if (conjunto.contiene(jugador)) {

return false;

}

if (jugador.getCantTarjetas() > 0 && conjunto.esLimiteAmarilla()) {

return false;

}

if (jugador.getGoles() == 0 && conjunto.esLimiteSinGoles()) {

return false;

}

if (conjunto.esLimitePorPais(jugador.getNacionalidad().getId())) {

return false;

}

return true;

}

private List<Jugador> ordenarJugadores() {

ArrayList<Jugador> ret = instancia.getJugadores();

Collections.sort(ret);

return ret;

}

}

**Algoritmo de Jugadores.java**

package controlador;

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import com.google.gson.Gson;

import com.google.gson.GsonBuilder;

import modelo.Jugador;

public class Jugadores {

private List<Jugador> jugadores;

public List<Jugador> levantarDatos() {

Gson json = new Gson();

Jugadores ret = new Jugadores();

try {

BufferedReader br = new BufferedReader(

new FileReader(Controlador.PATH));

ret = (Jugadores) json.fromJson(br, this.getClass());

} catch (FileNotFoundException e) {

return new ArrayList<Jugador>();

}

return ret.jugadores;

}

public void persistirDatos(List<Jugador> j) {

this.jugadores = j;

Gson gson = new GsonBuilder().setPrettyPrinting().create();

String json = gson.toJson(this);

try {

FileWriter file = new FileWriter(Controlador.PATH);

file.write(json);

file.close();

} catch (IOException e) {

System.out.println(e.getMessage());

}

}

public int size() {

return jugadores.size();

}

}