Sesión 4 - Simulación de un campeonato de palas de playa mediante un Sistema Multiagente

SISTEMAS DE AYUDA A LA TOMA DE DECISIONES

CURSO 2023/2024

Óscar Brizuela García (820773@unizar.es)

David Arruga Escuer (816058@unizar.es)

Entrega: 17/12/2023

Índice

1. Estructura de datos para el campeonato

La estructura de datos para el campeonato se detalla en el Anexo 6.1 . Esta estructura consta de la clase *Campeonato,* que incluye el número de jugadores, pasado como argumento en su constructor, y una matriz del mismo tamaño que el número de jugadores para almacenar los resultados de cada partido. Cada celda de esta matriz es de tipo *Resultado*. Este tipo de dato, representado por la clase *Resultado*, permite guardar el número de toques del jugador ganador, el del perdedor, así como quién ha sido el ganador y el perdedor de la partida. Por tanto, la celda en la fila *i* y columna *j* representa el partido entre el jugador *i* y el jugador *j*, sabiendo que cada jugador está identificado por un número único.

Además, la clase *Campeonato* proporciona todos los métodos necesarios para llevar a cabo un campeonato de manera eficiente. Esto incluye la capacidad de registrar nuevos resultados mediante la información de toques, nombres de los jugadores, así como la fila y columna correspondientes en la matriz. Asimismo, facilita la generación de emparejamientos según el número de jornada, devolviendo una lista de listas de tipo *string* con dos componentes. La clase también posibilita la impresión de la clasificación en tiempo real, ordenada por puntos y, en caso de empate, alfabéticamente. Además, ofrece la visualización del podio final y la matriz completa del campeonato.

Es importante señalar que en el caso de un campeonato con un número impar de jugadores, un jugador diferente descansará en cada jornada. Por lo tanto, el número total de jornadas será uno más que en un campeonato con un número par de jugadores.

2. Estructura de agentes

1.1. Agente organizador

Para implementar el agente organizador del campeonato, se ha adaptado la clase *Agente_GUI* suministrada. El método *setup()* de esta clase inicia generando un número aleatorio entre 10 y 20, que determinará la cantidad de jugadores en el campeonato. Luego, se instancia la clase *Campeonato* con dicho número y se crean los jugadores correspondientes utilizando la función *createNewAgent()* de la librería *ContainerController*, siguiendo la secuencia de instrucciones proporcionada. Es importante señalar que cada jugador se nombra con la letra 'J' seguida de un número único desde 1 hasta el máximo número de jugadores.

El comportamiento cíclico de esta clase también se ha modificado. Ahora recibe, además, como argumentos la clase *Campeonato* y el número de jugadores. En el método *action()*, durante la primera iteración, se generan los emparejamientos para la primera ronda, se visualizan en la interfaz mediante la instrucción "*vChat.taReceived.append*" y se envía un mensaje de tipo *REQUEST* a cada jugador según los emparejamientos obtenidos. Así, cada jugador recibe la información sobre si debe sacar o recibir, y quién es su adversario. Posteriormente, el agente espera la recepción de mensajes por parte de los jugadores.

Dado que solo los jugadores ganadores se comunican con el organizador del torneo, este ya puede inferir que el remitente del mensaje es el ganador, y el jugador mencionado en el mensaje es el perdedor, junto con la información sobre el número de toques. Con estos datos, se puede registrar el resultado en la clase *Campeonato*. Cabe destacar que, dado que este formato de campeonato no contempla partidos de ida y vuelta, el mismo resultado se guarda en la posición opuesta de la matriz, conformando una matriz simétrica, con la diagonal principal siempre con valores nulos (un jugador no puede jugar contra sí mismo).

Una vez recibidos todos los mensajes de una ronda, se muestra el botón "Mostrar Clasificación" en la interfaz. Se ha ajustado el código en la clase *VentanaChat* para que, al hacer clic en dicho botón, se envíe un mensaje al organizador, solicitando la visualización de la clasificación. Del mismo modo, se ha aplicado esta lógica al botón "Siguiente ronda". Cuando el organizador recibe el mensaje "Siguiente ronda", genera nuevos emparejamientos y notifica a los jugadores correspondientes, como se mencionó anteriormente. Este proceso se repite iterativamente hasta que se detecta la última ronda. En este punto, se muestra el botón "Mostrar Podio". Al pulsarlo y recibir el organizador el mensaje "*Mostrar podio*", se presenta una pantalla final que muestra al ganador (medalla de oro), al subcampeón (medalla de plata) y al tercer clasificado (medalla de bronce) con sus respectivas puntuaciones.

1.2. Agente jugador

El código del agente jugador, derivado del ejercicio 7, ha experimentado modificaciones sustanciales.

La principal alteración radica en que la clase ya no recibe parámetros directos, sino que estos son enviados desde el agente organizador. En cuanto al comportamiento de la clase, la primera acción consiste en esperar un mensaje del organizador que contenga toda la información necesaria para iniciar una partida. Una vez recibido este mensaje, el comportamiento es idéntico al desarrollado en el ejercicio 7. En caso de fallar, el agente no muere; en su lugar, permanece a la espera de un mensaje del organizador para iniciar otra partida. Por otro lado, en caso de victoria, el agente envía un mensaje al controlador informando del estado final de la partida.

3. Simulación del campeonato mediante paso de mensajes

Como se evidencia en secciones previas, se ha decidido no seguir estrictamente un protocolo FIPA, optando en su lugar por un enfoque más flexible. Aunque se contempló la posibilidad de ajustar el protocolo para alinearlo con algún estándar FIPA existente, la decisión final se basó en la eficacia del sistema actual. La ausencia de pérdida de mensajes, la uniformidad de todos los mensajes como tipo *REQUEST* y la capacidad de la clase destinataria para ejecutar acciones específicas en función del contenido del mensaje respaldan la elección actual. Además, el enfoque actual se adapta de manera precisa a los requisitos particulares del torneo, permitiendo una implementación más sencilla y eficiente.

Por tanto, el protocolo para el intercambio de mensajes sigue una estructura constante: el organizador crea los jugadores, quienes aguardan un mensaje para iniciar la partida. Una vez recibido, cada jugador se enfrenta a su oponente y, al concluir, ambos esperan un nuevo mensaje del organizador. Antes de ello, el ganador envía la información de la partida al organizador. Por otra parte, el organizador, después de avisar a cada jugador, espera las respuestas de los jugadores ganadores. Una vez recibidas todas estas respuestas, permanece a la espera de un mensaje que indique que el usuario ha interactuado con la interfaz, momento en el que ejecuta la acción correspondiente.

4. Interfaz

En cuanto a la interfaz que presenta tanto la información del campeonato como la lista de partidos en juego en una ronda específica, la clasificación parcial de los jugadores o la pantalla final con el podio, se ha adoptado la decisión de utilizar una única ventana que se actualiza conforme evoluciona el campeonato. En consecuencia, los botones se muestran o se ocultan según las necesidades específicas del momento.

Para implementar esta funcionalidad, se ha ajustado el código de la clase *VentanaChat*, donde ahora existen tres botones: "Mostrar clasificación actual", "Jugar siguiente ronda" y "Mostrar podio final". La visibilidad de estos botones en la interfaz es controlada por el organizador mediante los métodos públicos "*setBoton1Visible()*", "*setBoton2Visible()*" y "*setBoton3Visible()*", respectivamente.

Además, al pulsar cualquiera de estos botones, se envía un mensaje al agente cuyo nombre está especificado en el campo "*taSent*" y cuyo contenido se encuentra en el campo "*tfResponder*". Para modificar estos campos, se han creado los métodos "*setTaSent(string destinatario)*" y "*setTfResponder(string mensaje)*", los cuales son empleados por el agente organizador, y se pueden observar en el Anexo 6.6.

A continuación, se presentan las diversas opciones que ofrece la interfaz en función del momento específico del campeonato.

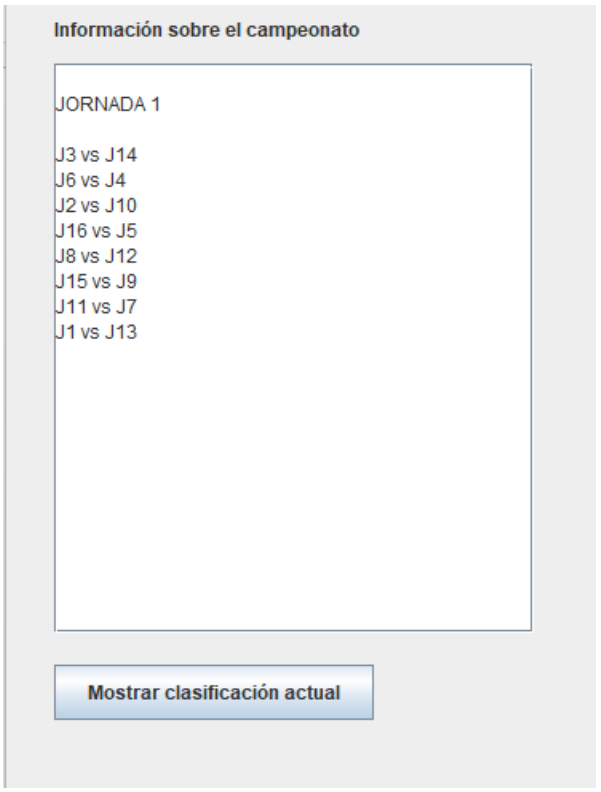

Figura 1: Interfaz donde se muestran los enfrentamientos para una jornada

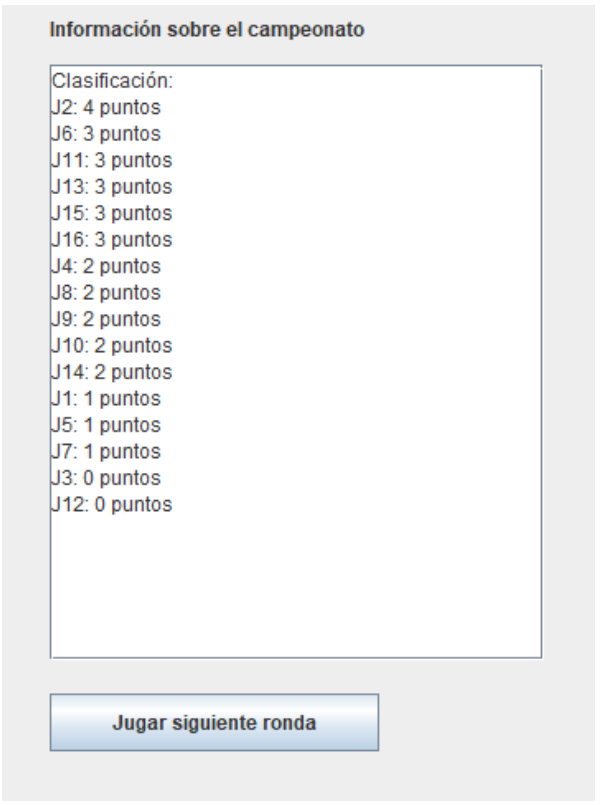

Figura 2: Interfaz donde se muestra la clasificación tras jugarse una jornada

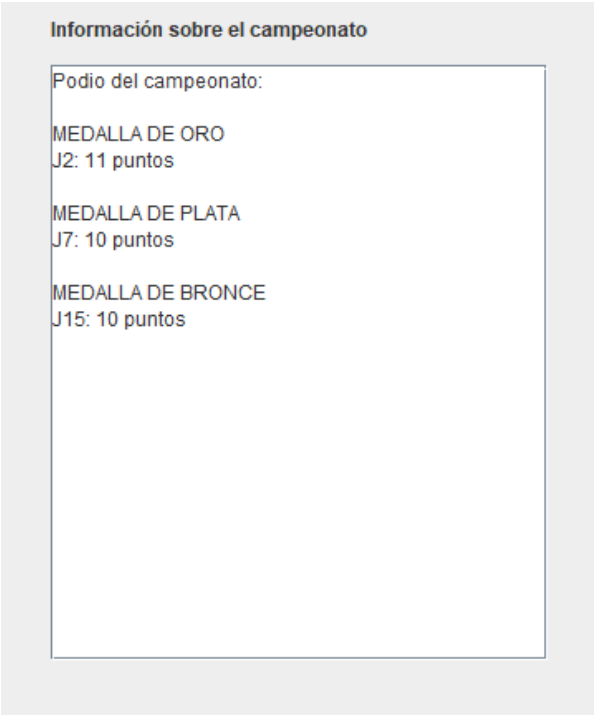

Figura 3: Interfaz donde se muestra el podio final del campeonato

5. Ampliación de la simulación

5.1. Función aditiva

Para dotar de mayor realismo al juego, se ha decidido incorporar una función adicional a la simulación previa. En este contexto, se ha formulado la siguiente función aditiva, la cual calcula la probabilidad de que un jugador le dé a la pelota, tomando en consideración su posición en el ranking, su estado físico y su estado anímico:

Si EF = 0, entonces probabilidad de toque = 0 (si el jugador está lesionado, no puede jugar)

Si EF ≠ 0, entonces probabilidad de toque = $1 * 1/R + 4 * EF + 2 * EA$

Donde:

- R: posición en el ranking
- EF: estado físico
- EA: estado anímico

Además, se han definido las siguientes codificación de estados:

- Estados físicos
  - Lesionado: 0
  - En recuperación: 1

    - Bien: 2
    - Excelente: 3
- Estados anímicos
    - Malo: 1
    - Bueno: 2
    - Excelente: 3

Además, para abordar la posibilidad de que la probabilidad anterior pueda exceder 1, se ha optado por normalizarla utilizando la siguiente función:

$$Probabilidad\ de\ toque\ normalizada\ = \frac{Probabilidad\ de\ toque * 0.9}{23}$$

Esta elección se fundamenta en asignar una probabilidad de 0.9 al jugador que ocupa la primera posición en el ranking y presenta un estado físico y anímico excelentes.

Por otro lado se ha decidido que, al final de cada ronda, los estados físicos y anímicos de un jugador pueden experimentar cambios. En este sentido, las probabilidades de transición de un jugador a otro estado son las siguientes:

- Cambio de estado físico:
    - Si el jugador está "lesionado", hay un 40% de probabilidad de pasar a "en recuperación".
    - Si el jugador está "en recuperación", existe un 30% de probabilidad de regresar a "lesionado"; no obstante, si la probabilidad supera el 70%, el jugador avanza a un estado "bien".
    - Si el jugador está "bien", se presenta un 30% de probabilidad de pasar a "en recuperación"; sin embargo, si la probabilidad excede el 70%, el jugador progresa a un estado "excelente".
    - Si el jugador está "excelente", existe un 30% de probabilidad de regresar a un estado "bien".
- Cambio de estado anímico:
    - Si el estado es "malo", hay un 60% de probabilidad de pasar a "bien".
    - Si el estado es "bueno", existe un 20% de probabilidad de regresar a "malo"; no obstante, si la probabilidad supera el 70%, el jugador avanza a un estado "excelente".
    - Si el estado es "excelente", se presenta un 40% de probabilidad de pasar a "bueno".

5.2. Modificación del código

Para incorporar la implementación de la nueva función aditiva a la simulación, fue necesario realizar ajustes en el código mencionado previamente. En primer lugar, se introdujeron tres nuevos atributos en las clases *Jugador* y *Jugador_Comportamiento* para que cada jugador pueda almacenar su posición en el ranking global, así como sus estados anímicos y físicos. Estos valores se proporcionan como argumentos al crear un jugador y, posteriormente, son pasados a la clase *Jugador_Comportamiento* para posibles modificaciones.

Por tanto, en el momento en que el agente crea a los jugadores se ha decidido que, inicialmente, el número de ranking coincida con el número de nombre de cada jugador, que el estado físico sea "bien", y que el estado anímico de los seis primeros jugadores sea "excelente", mientras que el resto tenga un estado anímico "bueno".

Se ha realizado otra modificación crucial que implica incorporar la probabilidad de toque en el mensaje que el organizador envía a cada jugador al inicio de una partida. La determinación de esta probabilidad se efectúa mediante el método *calcular_probabilidad_toque()* en la clase encargada del comportamiento del agente organizador. Este cálculo se realiza considerando la función aditiva previamente mencionada.

Además, teniendo en cuenta las probabilidades de cambio de estado, se ha implementado el método *actualizar_estados_jugadores()* en la misma clase. Este método facilita la actualización de los estados de los jugadores de acuerdo con las probabilidades establecidas. Además, notifica a cada jugador sobre el cambio de su estado mediante el envío de un mensaje correspondiente. La ejecución de este método tiene lugar al concluir cada jornada del campeonato.

Es relevante destacar que, para evitar la necesidad de consultar continuamente a cada jugador sobre su estado actual, se optó por la creación de un vector en esta clase. Esta medida facilita los cálculos y contribuye a reducir la cantidad de mensajes en la simulación.

Por otro lado, en lo que respecta al comportamiento del jugador, se ha incorporado la capacidad de procesar mensajes que comienzan con "cambio_estado". En tal caso, el jugador actualizará sus estados anímico y físico, sin realizar ninguna otra acción.

En el Anexo se encuentra el código de la clase Organizador en la sección 6.2, el código del comportamiento del Organizador en la sección 6.3, el código de la clase Jugador en la sección 6.4, y el código del comportamiento de la clase Jugador en la sección 6.5.

6. Anexo

6.1. Clase Campeonato

```java
public class Campeonato {

  private Resultado[][] enfrentamientos;

  private int num_jugadores;

  private class Resultado{
    int toquesGanador;
    int toquesPerdedor;
    String ganador;
    String perdedor;

    public Resultado( int toquesGanador, int toquesPerdedor, String ganador,
String perdedor){
      this.toquesGanador = toquesGanador;
      this.toquesPerdedor = toquesPerdedor;
      this.ganador = ganador;
      this.perdedor = perdedor;
    }
    public String getGanador(){
      return ganador;
    }

    @Override
    public String toString() {
      return "{" +toquesGanador +"," + toquesPerdedor +"," + ganador + "," +
perdedor +'}';
    }
  }

  public Campeonato(int num_jugadores){
    this.enfrentamientos = new Resultado[num_jugadores][num_jugadores];
    this.num_jugadores = num_jugadores;
  }

  public void anadirResultado(int toquesGanador, int toquesPerdedor, String
ganador, String perdedor, int fila, int columna) {
    Resultado resultado = new Resultado(toquesGanador, toquesPerdedor,
ganador,perdedor);
    this.enfrentamientos[fila][columna] = resultado;
  }
```

```
  private List<List<Integer>> filterUniqueTuples(List<List<Integer>>
inputList) {
    List<List<Integer>> result = new ArrayList<>();
    List<Integer> seen = new ArrayList<>();
    Collections.shuffle(inputList);
    for (List<Integer> tuple : inputList) {
      boolean contains = false;
      for (Integer element : tuple) {
        if (seen.contains(element)) {
          contains = true;
          break;
        }
      }
      if (!contains) {
        result.add(tuple);
        seen.addAll(tuple);
      }
    }
    return result;
  }

  private List<List<Integer>> forzardescanso(int num_jornada,
List<List<Integer>> lista_emparejamientos) {
    List<List<Integer>> listaActualizada = new ArrayList<>();
    Iterator<List<Integer>> iterator = lista_emparejamientos.iterator();

    while (iterator.hasNext()) {
      List<Integer> tupla = iterator.next();

      if (!tupla.contains(num_jornada)) {
        listaActualizada.add(tupla);
      }
    }
    return listaActualizada;
  }

  public List<List<String>> generarEmparejamientos(int num_jornada){
    List<List<Integer>> emparejamientos = new ArrayList<>();
    for (int i = 0; i < num_jugadores; i++) {
      for (int j = 0; j < num_jugadores; j++) {
        if (i != j && enfrentamientos[i][j] == null) {
          List<Integer> emparejamiento = new ArrayList<>();
          emparejamiento.add(i);
          emparejamiento.add(j);
          emparejamientos.add(emparejamiento);
        }

      }
```

```
    }
    if(num_jugadores % 2 != 0){
      emparejamientos = forzardescanso(num_jornada, emparejamientos);
    }
    List<List<Integer>> sin_repetidos = new ArrayList<>();
    sin_repetidos = filterUniqueTuples(emparejamientos);
    while (sin_repetidos.size() != (num_jugadores / 2)) {
      sin_repetidos = filterUniqueTuples(emparejamientos);
    }
    List<List<String>> emparejamientos_jug = new ArrayList<>();

    for (List<Integer> tuple : sin_repetidos) {
      List<String> transformedTuple = new ArrayList<>();
      for (Integer number : tuple) {
        transformedTuple.add("J" + (number + 1));
      }
      emparejamientos_jug.add(transformedTuple);
    }
    return emparejamientos_jug;
  }

  public String imprimirClasificacion(){
    Resultado resultado_jugador;
    HashMap<String, Integer> puntos_jugadores = new HashMap<>();
    for (int i = 1; i <= num_jugadores; i++) {
      puntos_jugadores.put("J" + i, 0);
    }
    for (int i = 0; i < num_jugadores; i++) {
      for (int j = i+1; j < num_jugadores; j++) {
        if (enfrentamientos[i][j] != null) {
          resultado_jugador = enfrentamientos[i][j];
          String ganador = resultado_jugador.getGanador();
          if (puntos_jugadores.containsKey(ganador)) {
            puntos_jugadores.put(ganador, puntos_jugadores.get(ganador) + 1);
          }
        }
      }
    }
    List<Entry<String, Integer>> listaPuntuaciones = new
ArrayList<>(puntos_jugadores.entrySet());

    Collections.sort(listaPuntuaciones, new Comparator<Entry<String,
Integer>>() {
      @Override
      public int compare(Entry<String, Integer> entry1, Entry<String, Integer>
entry2) {
        // Comparar por puntos
```

```
        int comparacionPorPuntos =
entry2.getValue().compareTo(entry1.getValue());
        if (comparacionPorPuntos != 0) {
          return comparacionPorPuntos;
        } else {
          // En caso de empate por puntos, comparar por longitud del nombre
          int comparacionPorLongitud =
Integer.compare(entry1.getKey().length(), entry2.getKey().length());
          if (comparacionPorLongitud != 0) {
            return comparacionPorLongitud;
          } else {
            // En caso de empate por longitud del nombre, ordenar
alfabéticamente por el nombre del jugador
            return entry1.getKey().compareTo(entry2.getKey());
          }
        }
      }
    });
    String clasificacion = "";
    String jugador_str = "";
    // Imprimir la clasificación ordenada
    clasificacion = clasificacion.concat("Clasificación:");
    clasificacion = clasificacion.concat("\n");
    for (Entry<String, Integer> entrada : listaPuntuaciones) {
      String jugador = entrada.getKey();
      int puntos = entrada.getValue();
      jugador_str = jugador + ": " + puntos + " puntos"+"\n";
      clasificacion = clasificacion.concat(jugador_str);
    }
    return clasificacion;
  }

  public String imprimirPodio(){
    String clasificacion = imprimirClasificacion();
    String[] lineas = clasificacion.split("\n");
    String podio = "Podio del campeonato:\n";
    podio = podio.concat("\n");
    podio = podio.concat("MEDALLA DE ORO\n");
    podio = podio.concat(lineas[1]);
    podio = podio.concat("\n");
    podio = podio.concat("\n");
    podio = podio.concat("MEDALLA DE PLATA\n");
    podio = podio.concat(lineas[2]);
    podio = podio.concat("\n");
    podio = podio.concat("\n");
    podio = podio.concat("MEDALLA DE BRONCE\n");
    podio = podio.concat(lineas[3]);
    return podio;
```

```
  }

  public void imprimirCampeonato() {
    // Imprimir encabezado de columnas
    System.out.print("      ");
    for (int col = 0; col < num_jugadores; col++) {
      System.out.print((col) + "      "); // Ajusta los espacios según sea
necesario
    }
    System.out.println();

    // Imprimir línea divisoria
    System.out.print("   +");
    for (int col = 0; col < num_jugadores; col++) {
      System.out.print("------");
    }
    System.out.println();

    // Imprimir matriz con números de fila y columna
    for (int i = 0; i < num_jugadores; i++) {
      // Número de fila
      System.out.print((i) + " |");

      for (int j = 0; j < num_jugadores; j++) {
        System.out.print(" " + enfrentamientos[i][j] + "     ");
      }
      System.out.println();
    }
  }
}
```

6.2. Agente Organizador

Java

```
public class Agente_GUI extends GuiAgent{

  VentanaChat vChat;

  boolean packFrame = false;

  int num_jugadores;

  List<String> jugadores = new ArrayList<>();
  private int generarnumJugadores(){
```

```
    Random rand = new Random();
    return rand.nextInt(20 -10 +1)+10;
  }

  public void setup() {
    this.num_jugadores = generarnumJugadores();
    Object[] caracteristicasJugadores = new Object[num_jugadores];
    Campeonato campeonato = new Campeonato(this.num_jugadores);
    String nombre_jugador = "";
    String estado_animico = "";
    System.out.println("Numero de jugadores: "+this.num_jugadores);
    for (int i = 1; i <= this.num_jugadores; i++) {
      nombre_jugador = "J" + i;
      this.jugadores.add(nombre_jugador);
      if(i < 6){
        estado_animico = "excelente";
      }
      else{
        estado_animico = "bueno";
      }
      Object[] parametros = new Object[] {i, "bien", estado_animico};
      caracteristicasJugadores[i-1] =parametros;
      ContainerController cc = getContainerController();
      try {
        AgentController jugadorController = cc.createNewAgent(nombre_jugador,
Jugador.class.getName(), parametros);
        jugadorController.start();

      } catch (Exception e) {
        e.printStackTrace();
      }
    }
     vChat = new VentanaChat(this);
    if (packFrame)
      vChat.pack();
    else
      vChat.validate();

    CyclicBehaviour cb = new Agente_TalkBehaviour(this, vChat, campeonato,
num_jugadores, caracteristicasJugadores);
    addBehaviour(cb);

    // Centrar la ventana
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = vChat.getSize();
    if (frameSize.height > screenSize.height)
      frameSize.height = screenSize.height;
    if (frameSize.width > screenSize.width)
```

```
      frameSize.width = screenSize.width;

    vChat.setLocation((screenSize.width - frameSize.width) / 2,
        (screenSize.height - frameSize.height)/2);
    vChat.setVisible(true);

  }

  public void onGuiEvent(GuiEvent ge){
    String receiverName = (String) ge.getParameter(0);
    String msgContent = (String) ge.getParameter(1);
    ACLMessage toSend = new ACLMessage(ACLMessage.REQUEST);
    toSend.setContent(msgContent);
    toSend.setPerformative(ACLMessage.REQUEST);
    toSend.addReceiver(new AID(receiverName, AID.ISLOCALNAME));
    send(toSend);
  }

}
```

6.3. Comportamiento del agente Organizador

Java

```
public class Agente_TalkBehaviour extends CyclicBehaviour{

  VentanaChat vChat;

  Campeonato campeonato;

  List<List<String>> emparejamientos;

  boolean primeraVez = true;

  int num_mensajes_recibidos = 0;

  int num_jugadores = 0;

  int num_rondas = 1;

  Object[] caracteristicasJugadores;
```

```
  public Agente_TalkBehaviour(GuiAgent ga, VentanaChat vc, Campeonato
campeonato_, int num_jugadores_, Object[] caracteristicasJugadores_){
    super(ga);
    vChat = vc;
    campeonato = campeonato_;
    num_jugadores = num_jugadores_;
    caracteristicasJugadores = caracteristicasJugadores_;
  }

  private int extraerNumero(String nombre_agente) {
    Pattern pattern = Pattern.compile("\\d+");
    Matcher matcher = pattern.matcher(nombre_agente);
    if (matcher.find()) {
      int numero = Integer.parseInt(matcher.group());
      return numero;
    }
    return -1;
  }

  private int conversor_estado_fisico(String estadoFisico) {
    switch (estadoFisico) {
      case "lesionado":
        return 0;
      case "en recuperacion":
        return 1;
      case "bien":
        return 2;
      case "excelente":
        return 3;
      default:
        return -1;
    }
  }

  private int conversor_estado_animico(String estadoAnimico) {
    switch (estadoAnimico) {
      case "malo":
        return 1;
      case "bueno":
        return 2;
      case "excelente":
        return 3;
      default:
        return -1;
    }
  }

  private double normalizar_probabilidad(double probabilidad){
```

```
    return probabilidad*0.9/23;
  }

  private double calcular_probabilidad_toque(String jugador){
    double p = 0;
    int num_jugador = extraerNumero(jugador);
    Object[] elementos = (Object[]) caracteristicasJugadores[num_jugador-1];
    if(elementos[1].toString().equals("lesionado")){
      return 0;
    }

    p = (1.0 / (int) elementos[0])
        + 4 * conversor_estado_fisico(elementos[1].toString())
        + 2 * conversor_estado_animico(elementos[2].toString());
    return p;
  }

  private void avisarJugadores(){
    String msg = "";
    double probabilidad_toque = 0;
    for (List<String> emparejamiento : emparejamientos) {
      if (!emparejamiento.get(1).equals("")) {
        ACLMessage msg_enviado = new ACLMessage(ACLMessage.REQUEST);
        probabilidad_toque =
normalizar_probabilidad(calcular_probabilidad_toque(emparejamiento.get(1)));
        AID p = new AID(emparejamiento.get(1), AID.ISLOCALNAME);
        // Añadimos como receptor del mensaje el AID
        msg_enviado.addReceiver(p);
        // Añadimos el contenido del mensaje
        msg = "recibe"+","+ emparejamiento.get(0)+","+probabilidad_toque;
        msg_enviado.setContent(msg);
        // Enviamos el mensaje
        myAgent.send(msg_enviado);

        p = new AID(emparejamiento.get(0), AID.ISLOCALNAME);
        // Añadimos como receptor del mensaje el AID
        msg_enviado.addReceiver(p);
        // Añadimos el contenido del mensaje
        probabilidad_toque =
normalizar_probabilidad(calcular_probabilidad_toque(emparejamiento.get(0)));
        msg = "saca_bola"+","+ emparejamiento.get(1)+","+probabilidad_toque;
        msg_enviado.setContent(msg);
        // Enviamos el mensaje
        myAgent.send(msg_enviado);
      }
    }
  }
```

```
  private void actualizar_estados_jugadores(){
    String[] estadosAnimicos = {"malo", "bueno", "excelente"};
    String[] estadosFisicos = {"lesionado", "en recuperacion", "bien",
"excelente"};
    Random random = new Random();
    int num_jugador = 1;
    String jugador = "";
    String msg = "";
    for (Object caracteristicasJugador : caracteristicasJugadores) {
      Object[] elementos = (Object[]) caracteristicasJugador;

      if (elementos[1].equals("lesionado")) {
        if (random.nextDouble() < 0.4) {
          elementos[1] = "en recuperacion";
        }
      } else if (elementos[1].equals("en recuperacion")) {
        if (random.nextDouble() < 0.3) {
          elementos[1] = "lesionado";
        } else if (random.nextDouble() > 0.7) {
          elementos[1] = "bien";
        }
      } else if (elementos[1].equals("bien")) {
        if (random.nextDouble() < 0.3) {
          elementos[1] = "en recuperacion";
        } else if (random.nextDouble() > 0.7) {
          elementos[1] = "excelente";
        }
      } else if (elementos[1].equals("excelente")) {
        if (random.nextDouble() < 0.3) {
          elementos[1] = "bien";
        }
      }

      if (elementos[2].equals("malo")) {
        if (random.nextDouble() < 0.6) {
          elementos[2] = "bien";
        }
      } else if (elementos[2].equals("bueno")) {
        if (random.nextDouble() < 0.2) {
          elementos[2] = "malo";
        } else if (random.nextDouble() > 0.7) {
          elementos[2] = "excelente";
        }
      } else if (elementos[2].equals("excelente")) {
        if (random.nextDouble() < 0.4) {
          elementos[2] = "bueno";
        }
      }
```

```
      //avisar al jugador correspondiente
      jugador = "J"+num_jugador;
      ACLMessage msg_enviado = new ACLMessage(ACLMessage.REQUEST);
      AID p = new AID(jugador, AID.ISLOCALNAME);
      // Añadimos como receptor del mensaje el AID
      msg_enviado.addReceiver(p);
      // Añadimos el contenido del mensaje
      msg = "cambio_estado"+","+ elementos[1] +","+elementos[2] ;
      msg_enviado.setContent(msg);
      // Enviamos el mensaje
      myAgent.send(msg_enviado);

      num_jugador++;

    }
  }

  public void action(){
    if(primeraVez){
      vChat.setBoton1Visible(false);
      vChat.setBoton2Visible(false);
      vChat.setBoton3Visible(false);
      emparejamientos = campeonato.generarEmparejamientos(num_rondas-1);
      String partido = "";
      vChat.taReceived.append("\n" + "JORNADA "+num_rondas+"\n");
      for (List<String> listaInterna : emparejamientos) {
        partido = String.join(" vs ", listaInterna);
        vChat.taReceived.append("\n" + partido);
      }
      avisarJugadores();
      primeraVez = false;
    }

    ACLMessage reply =
myAgent.receive(MessageTemplate.MatchPerformative(ACLMessage.REQUEST));
    if (reply!=null){
      String content = reply.getContent();
      String sender = reply.getSender().getName();
      if(content.equals("Siguiente ronda")){
        vChat.setBoton2Visible(false);
        emparejamientos = campeonato.generarEmparejamientos(num_rondas-1);
        String partido = "";
        vChat.taReceived.append("JORNADA "+num_rondas+"\n");
        for (List<String> listaInterna : emparejamientos) {
          partido = String.join(" vs ", listaInterna);
          vChat.taReceived.append("\n" + partido);
        }
        vChat.taReceived.append("\n");
```

```
        avisarJugadores();
      }
      else if(content.equals("Mostrar clasificacion")){
        vChat.setBoton1Visible(false);
        vChat.taReceived.append(campeonato.imprimirClasificacion());
        vChat.setTaSent("Siguiente ronda");
        vChat.setTfResponder(myAgent.getLocalName());
        vChat.setBoton2Visible(true);
      }
      else if(content.equals("Mostrar podio")){
        vChat.setBoton3Visible(false);
        vChat.taReceived.append(campeonato.imprimirPodio());
      }
      else{
        String[] parts = sender.split("@");
        String nombre_agente_ganador = parts[0];
        int numero_agente_ganador = extraerNumero(nombre_agente_ganador);
        String[] contenido_mensaje = content.split(",");
        int num_toques_ganador = Integer.parseInt(contenido_mensaje[1]);
        int num_toques_perdedor = num_toques_ganador -1;
        String nombre_agente_perdedor = contenido_mensaje[2];
        int numero_agente_perdedor =  extraerNumero(nombre_agente_perdedor);

        campeonato.anadirResultado(num_toques_ganador,num_toques_perdedor,

nombre_agente_ganador,nombre_agente_perdedor,numero_agente_ganador-1
            ,numero_agente_perdedor-1);
        campeonato.anadirResultado(num_toques_ganador,num_toques_perdedor,

nombre_agente_ganador,nombre_agente_perdedor,numero_agente_perdedor-1
            ,numero_agente_ganador-1);
        num_mensajes_recibidos++;
        if(num_mensajes_recibidos == num_jugadores/2){
          campeonato.imprimirClasificacion();
          num_mensajes_recibidos = 0;
          num_rondas ++;
          actualizar_estados_jugadores();

          if ((num_rondas == num_jugadores && num_jugadores% 2 == 0)||(num_rondas
== num_jugadores+1 && num_jugadores% 2 != 0)){
            campeonato.imprimirCampeonato();
            vChat.setBoton3Visible(true);
            vChat.setTaSent("Mostrar podio");
            vChat.setTfResponder(myAgent.getLocalName());
            vChat.setBoton1Visible(false);
            vChat.setBoton2Visible(false);
          }
          else{
```

```
            campeonato.imprimirCampeonato();
            vChat.setTaSent("Mostrar clasificacion");
            vChat.setTfResponder(myAgent.getLocalName());
            vChat.setBoton1Visible(true);
          }
        }
      }


    }
    else
      block();
  }
}
```

6.4. Agente Jugador

Java

```
public class Jugador extends Agent {

  int ranking = 0;

  String estado_fisico = "";

  String estado_animico = "";


  @Override
  public void setup(){
    System.out.println("Hola, soy el agente " + getLocalName());
    Object [] args = getArguments();
    if (args != null){
      ranking = (int) args[0];
      estado_fisico = args[1].toString();
      estado_animico = args[2].toString();
    }
    addBehaviour(new
Jugador_Comportamiento(ranking,estado_fisico,estado_animico));
  }
  @Override
  public void takeDown(){
    System.out.println("El agente " + getLocalName() + " muere");
  }
```

```
}
```

6.5. Comportamiento del agente Jugador

Java

```
public class Jugador_Comportamiento extends CyclicBehaviour{
  int numToques = 0;
  String contenido_mensaje_recibido = "";
  String contenido_mensaje_enviado = "";
  String contrincante = "";
  String rol = "";
  String controlador = "";
  boolean primeraVez = true;
  double messageSendProbability = 1;
  double probabilidad_toque = 0;
  int ranking = 0;
  String estado_fisico = "";
  String estado_animico = "";

  private void decreaseProbability() {
    messageSendProbability *= probabilidad_toque;
    System.out.println(myAgent.getLocalName()+" le voy a dar con "+
probabilidad_toque);
    messageSendProbability = Math.max(0.0, Math.min(1.0,
messageSendProbability));
  }

  private boolean shouldSendMessage() {
    decreaseProbability();
    return Math.random() < messageSendProbability;
  }
  public Jugador_Comportamiento(int ranking_, String estado_fisico_, String
estado_animico_){
    super();
    ranking = ranking_;
    estado_fisico = estado_fisico_;
    estado_animico = estado_animico_;
  }

  private void limpiarMensajesRonda(){
    numToques = 0;
```

```
    contenido_mensaje_recibido = "";
    contenido_mensaje_enviado = "";
    contrincante = "";
    rol = "";
    controlador = "";
  }

  //@Override
  public void action() {
    if (primeraVez){
      ACLMessage msg_recibido = null;
      msg_recibido = myAgent.blockingReceive();
      if (msg_recibido != null) {
        contenido_mensaje_recibido = msg_recibido.getContent();
        controlador = msg_recibido.getSender().getLocalName();
        String[] partes = contenido_mensaje_recibido.split(",");
        if (partes[0].equals("recibe")){
          rol = "recibidor";
          probabilidad_toque = Double.parseDouble(partes[2]);
          contrincante = partes[1];
          contenido_mensaje_recibido = "";
          primeraVez = false;
        }
        else if(partes[0].equals("saca_bola")){
          rol = "pasador";
          probabilidad_toque = Double.parseDouble(partes[2]);
          contrincante = partes[1];
          contenido_mensaje_recibido = "";
          primeraVez = false;
        }
        else if(partes[0].equals("cambio_estado")){
          estado_fisico = partes[1];
          estado_animico = partes[2];
          rol = "";
        }

      }
    }
    if(rol.equals("pasador")){
      if(probabilidad_toque != 0){
        if (numToques == 0 && contenido_mensaje_recibido.equals("")) {
          contenido_mensaje_enviado = "Saco";
          numToques++;
          System.out.println(myAgent.getLocalName() + "->" +
contenido_mensaje_enviado);
        } else {
          if (shouldSendMessage()) {
            contenido_mensaje_enviado = "Le doy";
```

```
            numToques++;
            System.out.println(myAgent.getLocalName() + "->" +
contenido_mensaje_enviado);
          } else {
            contenido_mensaje_enviado = "¡Fallé!";
            System.out.println(myAgent.getLocalName() + "->" +
contenido_mensaje_enviado + " Conseguí " + numToques + " toques");
          }
        }
      }
      else{
        contenido_mensaje_enviado = "¡Fallé!";
        System.out.println(myAgent.getLocalName() + "->" +
contenido_mensaje_enviado + " Conseguí " + numToques + " toques");
      }

      myAgent.doWait(500);

      // Se construye un mensaje de tipo PETICIÓN
      ACLMessage msg_enviado = new ACLMessage(ACLMessage.REQUEST);

      // Construimos un objeto de tipo Identificador de Agente (Agent
IDentifier)
      AID p = new AID(contrincante, AID.ISLOCALNAME);
      // Añadimos como receptor del mensaje el AID
      msg_enviado.addReceiver(p);
      // Añadimos el contenido del mensaje

      msg_enviado.setContent(contenido_mensaje_enviado);

      // Enviamos el mensaje
      myAgent.send(msg_enviado);
      if(contenido_mensaje_enviado.equals("¡Fallé!")){
        limpiarMensajesRonda();
        primeraVez = true;
      }
      else{
        rol = "recibidor";
      }

    }
    if(rol.equals("recibidor")){
      ACLMessage msg_recibido = myAgent.blockingReceive();
      if (msg_recibido != null){
        contenido_mensaje_recibido = msg_recibido.getContent();
        contrincante = msg_recibido.getSender().getLocalName();
        myAgent.doWait(500);

      }
```

```
      if(contenido_mensaje_recibido.equals("Saco") ||
contenido_mensaje_recibido.equals("Le doy") ) {
        rol = "pasador";
      }
      else if(contenido_mensaje_recibido.equals("¡Fallé!")){
        System.out.println(myAgent.getLocalName()+"->"+"HE GANADO!!!!
Conseguí "+numToques+" toques");

        ACLMessage msg_enviado = new ACLMessage(ACLMessage.REQUEST);

        // Construimos un objeto de tipo Identificador de Agente (Agent
IDentifier)
        AID p = new AID(controlador, AID.ISLOCALNAME);
        // Añadimos como receptor del mensaje el AID
        msg_enviado.addReceiver(p);
        // Añadimos el contenido del mensaje
        contenido_mensaje_enviado="He ganado"+","+numToques+","+contrincante;
        msg_enviado.setContent(contenido_mensaje_enviado);

        // Enviamos el mensaje
        myAgent.send(msg_enviado);

        limpiarMensajesRonda();
        primeraVez = true;
      }
    }
  }
}
```

6.6. Clase VentanaChat

Java

```
public class VentanaChat extends javax.swing.JFrame {

  private GuiAgent owner;
  public final int SENT_TYPE = 0;
  /**
     * Creates new form VentanaChat
```

```
     */
    public VentanaChat(GuiAgent a) {
        initComponents();
        owner = a;
    }

    /**
     * This method is called from within the constructor to initialize the
form.
     * WARNING: Do NOT modify this code. The content of this method is
always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated
Code">//GEN-BEGIN:initComponents
    private void initComponents() {

        jLabel1 = new javax.swing.JLabel();
        tfResponder = new javax.swing.JTextField();
        taSent = new javax.swing.JTextField();
        jButton1 = new javax.swing.JButton();
        jButton2 = new javax.swing.JButton();
        jButton3 = new javax.swing.JButton();
        jScrollPane1 = new javax.swing.JScrollPane();
        taReceived = new javax.swing.JTextArea();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        getContentPane().setLayout(null);

        jLabel1.setText("Información sobre el campeonato");
        getContentPane().add(jLabel1);
        jLabel1.setBounds(30, 10, 440, 16);

        tfResponder.setName("tfResponder"); // NOI18N
        tfResponder.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                tfResponderActionPerformed(evt);
            }
        });


        jButton1.setText("Mostrar clasificación actual");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });
        getContentPane().add(jButton1);
```

```
    jButton1.setBounds(30, 420, 200, 35);

    jButton2.setText("Jugar siguiente ronda");
    jButton2.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton2ActionPerformed(evt);
      }
    });
    getContentPane().add(jButton2);
    jButton2.setBounds(30, 420, 200, 35);

    jButton3.setText("Mostrar podio final");
    jButton3.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton3ActionPerformed(evt);
      }
    });
    getContentPane().add(jButton3);
    jButton3.setBounds(30, 420, 200, 35);

    taReceived.setColumns(20);
    taReceived.setRows(5);
    taReceived.setName("taReceived"); // NOI18N
    jScrollPane1.setViewportView(taReceived);

    getContentPane().add(jScrollPane1);
    jScrollPane1.setBounds(30, 40, 300, 360);

    pack();
  }// </editor-fold>//GEN-END:initComponents

  private void tfResponderActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_tfResponderActionPerformed
    // TODO add your handling code here:
  }//GEN-LAST:event_tfResponderActionPerformed

  private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton2ActionPerformed
    System.out.println("Has pulsado el botón Limpiar");
    jButton2.setVisible(false);
    taReceived.setText("");
    GuiEvent ge = new GuiEvent(this, SENT_TYPE);
    ge.addParameter(tfResponder.getText());
    ge.addParameter(taSent.getText());
    owner.postGuiEvent(ge);
    taSent.setText("");
  }//GEN-LAST:event_jButton2ActionPerformed
```

```
  private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton1ActionPerformed
    jButton1.setVisible(false);
    taReceived.setText("");
    System.out.println("Has pulsado el botón Enviar");
    GuiEvent ge = new GuiEvent(this, SENT_TYPE);
    ge.addParameter(tfResponder.getText());
    ge.addParameter(taSent.getText());
    owner.postGuiEvent(ge);
    taSent.setText("");

  }//GEN-LAST:event_jButton1ActionPerformed
  private void jButton3ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton1ActionPerformed
    jButton3.setVisible(false);
    taReceived.setText("");
    GuiEvent ge = new GuiEvent(this, SENT_TYPE);
    ge.addParameter(tfResponder.getText());
    ge.addParameter(taSent.getText());
    owner.postGuiEvent(ge);
    taSent.setText("");

  }//GEN-LAST:event_jButton1ActionPerformed

  public void setBoton1Visible(boolean visible) {
    jButton1.setVisible(visible);
  }
  public void setBoton2Visible(boolean visible) {
    jButton2.setVisible(visible);
  }
  public void setBoton3Visible(boolean visible) {
    jButton3.setVisible(visible);
  }
  public void setTaSent(String destinatario){
    taSent.setText(destinatario);
  }
  public void setTfResponder(String mensaje){
    tfResponder.setText(mensaje);
  }

  // Variables declaration - do not modify//GEN-BEGIN:variables
  private javax.swing.JButton jButton1;
  private javax.swing.JButton jButton2;
  private javax.swing.JButton jButton3;
  private javax.swing.JLabel jLabel1;

  private javax.swing.JScrollPane jScrollPane1;
  public javax.swing.JTextArea taReceived;
```

```
    private javax.swing.JTextField taSent;
    private javax.swing.JTextField tfResponder;
    // End of variables declaration//GEN-END:variables
}
```