

Este tutorial explica los pasos a seguir para desarrollar una táctica y así poder participar en la javaCup.

La javaCup es un torneo de fútbol virtual donde cada participante programa en java el comportamiento de un equipo de fútbol. Luego los códigos de las tácticas enviadas se medirán en un único torneo compitiendo por los premios ofertados.

Para desarrollar una táctica solo se debe escribir una clase que implemente la interfaz `Tactica`. Esto quiere decir que deberemos implementar todos los métodos definidos en dicha interfaz. Los métodos a implementar son los siguientes:

- 1) `public TacticaDetalle getDetalle();`
- 2) `public Posicion[] getPosicionSaca();`
- 3) `public Posicion[] getPosicionRecibe();`
- 4) `public List<Comando> ejecutar(SituacionPartido sp);`

El primer método denominado `getDetalle` retorna un objeto de la clase `TacticaDetalle` que especifica características de la táctica como son: el nombre del equipo, el país, el nombre del entrenador, los colores y estilo de la vestimenta. Y para cada jugador su nombre, número, colores, si es portero, su velocidad, su fuerza de remate y su precisión.

Para facilitar la implementación de este método se ha desarrollado un programa asistente que de manera grafica permite generar el código del objeto `TacticaDetalle` (luego explicaremos como realizar esto).

El segundo y tercer método especifican la ubicación que deben tomar los jugadores después de cada gol y cuando se inicie el juego. Se usara `getPosicionSaca` cuando se inicie "sacando" después de haber recibido un gol, y se usara `getPosicionRecibe` cuando se inicie "recibiendo" después de marcar un gol. Como el partido ocurre en un solo tiempo y por lo tanto no hay cambio de lado, al inicio ambas tácticas se ubicaran según `getPosicionRecibe`, para que ambas tácticas tengan posibilidades de controlar el balón primero que la otra. Ambos métodos deberán retornar un array de objetos `Posicion` de tamaño 11.

Al igual que en el primer método el programa asistente permitirá de manera grafica generar los códigos necesarios para implementar los métodos `getPosicionSaca` y `getPosicionRecibe`.

ASISTENTE PARA LA CREACIÓN DE EQUIPOS

Con el fin de facilitar la creación de tácticas se ha desarrollado un programa asistente que apoyara la generación de código, la administración de alineaciones, y la simulación remates. A continuación describiremos como se usa.

Para iniciar el asistente debes ejecutar la clase:
`org.javahispano.javacup.asistente.Asistente.`

El asistente consta de tres pestañas: “Equipo”, “Jugadores” y “Alineaciones y simulación de Remate”.

En la primera pestaña, se definen características descriptivas y de visualización del equipo. Los datos obligatorios a ingresar son el nombre del equipo, el país y el nombre del entrenador.



El botón “Al Azar” permitirá generar vestimentas al azar.

En la segunda pestaña se definen las características de los jugadores. Están las características de descripción (nombre y número), las visuales (color piel, color pelo y si es portero) y las aptitudes (velocidad, remate y error).



Cada aptitud posee un valor mínimo y otro máximo, el valor mínimo esta a la izquierda y equivale a cero crédito usado, por otro lado, el valor máximo esta a la derecha y equivale a un crédito usado. La manera de configurar esto es distribuyendo a gusto los créditos disponibles entre los jugadores y sus aptitudes.

El error se refiere a un error angular expresado en porcentaje. Por ejemplo un 15% de error significa un 15% sobre 180°, es decir $0.15 \cdot 180^\circ$, que equivale a 27° . Ahora si el remate está dirigido hacia los 90° , el ángulo final puede variar entre $90^\circ + 27^\circ/2$ y $90^\circ - 27^\circ/2$, ósea entre 76.5° y 103.5° .

Las alineaciones se configuran en la tercera pestaña. Recordemos que la interfaz Tactica, nos exige definir 2 alineaciones, una para cuando se inicia sacando y otra para cuando se inicia recibiendo. En el campo de juego que se muestra en la figura se pueden ubicar los jugadores y así configurar alineaciones.



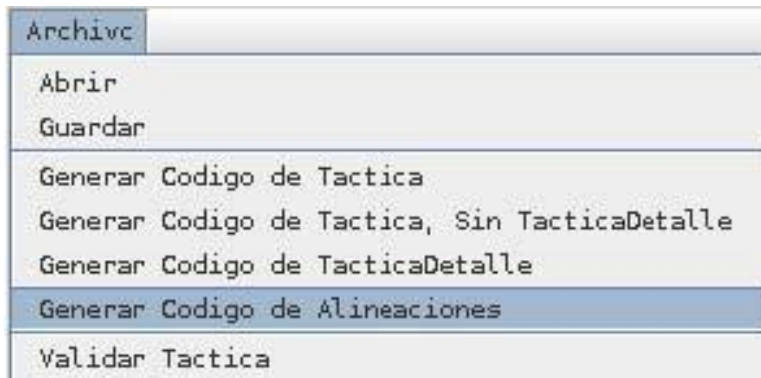
Se tienen 3 tipos de alineaciones: un tipo de alineación para ser usada durante el juego, otro tipo para cuando se inicie sacando, y otro cuando se inicie recibiendo. El tipo de alineación se configura en el segundo comboBox.

El primer comboBox mantiene la lista de alineaciones guardadas, los botones siguientes sirven para crear una nueva alineación, para eliminar la alineación seleccionada y para subir o bajar una alineación.

Los controles de la derecha nos permiten simular remates, indicando la fuerza [0-1], el ángulo de remate [0-360] y el ángulo vertical [0-Constantes.ANGULO_VERTICAL_MAX]. Luego con el botón ">" se simula el remate indicando las alturas del balón según la leyenda de colores ubicada en la zona inferior derecha.

Por defecto se dispone de 6 alineaciones de ejemplo: 1 para iniciar sacando, 1 para iniciar recibiendo y 4 alineaciones para usarlas durante la ejecución de partidos.

El menú del asistente permite abrir, guardar, generar código y validar una táctica.



Para generar código se tienen diversas alternativas:

- 1 La primera alternativa genera el código de la clase que implementa la interfaz `Tactica` y como clase interna se incluye la clase que implementa `TacticaDetalle`, además se incluye el código de las alineaciones.
- 2 La segunda opción solo genera la clase que implementa `Tactica`.
- 3 La tercera opción solo genera la clase que implementa `TacticaDetalle`.
- 4 la cuarta alternativa solo genera el código de las alineaciones.

La opción de validación nos muestra errores que podemos estar cometiendo, como por ejemplo que no estén definidos nombres para el equipo o entrenador, o que dos jugadores tengan el mismo número, o que se ocupen mas de los créditos disponibles, etc.

EL FRAMEWORK

El framework se compone de varios paquetes, siendo `org.javahispano.javacup.modelo` el más importante. Esta es una lista de las clases principales ubicadas dicho paquete:

Comando: Clase abstracta, que representa un comando que puede ejecutar un jugador.

ComandoIrA: Extiende de **Comando** e indica a un jugador hacia dónde dirigirse.

ComandoGolpearBalon: Extiende de **Comando** e indica a un jugador como rematar.

Partido: Clase usada para ejecutar partidos.

Posición: coordenadas espaciales con tipo primitivo `double`.

SituacionPartido: Retorna datos del estado del juego durante la ejecución de un partido.

Tactica: La interfaz que deberás implementar.

TacticaDetalle: La interfaz que genera el asistente.

Constantes: Clase donde están las constantes.

LÓGICA DE UNA TÁCTICA

```
public List<Comando> ejecutar(SituacionPartido sp);
```

En la implementación del método ejecutar de la interfaz Tactica, es donde se programa la lógica que modelará el comportamiento de tu equipo.

El método ejecutar recibe como parámetro un objeto SituacionPartido. Este objeto contiene información del estado en que se encuentra el partido, esta es la información que nos provee:

`double alturaBalon()` Retorna la altura del balón.

`Posicion balon()` Retorna la posición del balón.

`JugadorDetalle[] detalleJugadoresRivales()` Retorna el detalle (nombre, numero, velocidad, remate y precisión) de los jugadores rivales.

`JugadorDetalle[] detalleJugadores()` Retorna el detalle (nombre, numero, velocidad, remate y precisión) de mis jugadores.

`int golesPropios()` Retorna la cantidad de goles convertidos por mi equipo.

`int golesRival()` Retorna la cantidad de goles convertidos por el rival.

`int iteracion()` Retorna el numero de iteraciones cursadas en el partido.

`int[] iteracionesParaRematar()` Indica cuantas iteraciones deben pasar para que mis jugadores puedan volver a rematar.

`int[] iteracionesParaRematarRival()` Indica cuantas iteraciones deben pasar para que los jugadores rivales puedan volver a rematar.

`Posicion[] misJugadores()` Array de posiciones de mis jugadores.

`Posicion[] rivales()` Array de posiciones de rivales.

`int[] puedenRematar()` Array de índices de jugadores míos que están en condición de rematar.

`int[] puedenRematarRival()` Array de índices de jugadores rivales que están en condición de rematar.

`boolean sacaRival()` Retorna true si el rival realizara un saque, hasta que el rival no saque, mis jugadores se apartan de la posición de saque y no corre el tiempo.

`boolean saco()` Retorna true si tengo que realizar un saque, hasta que yo no saque, el rival se aparta de la posición de saque y no corre el tiempo.

`double[] getTrayectoria(int iteracion)` Retorna un array con las coordenadas x,y,z de la posición del balón en iteraciones futuras.

`int[] getRecuperacionBalon()` Retorna un array donde el primer elemento es la iteración donde se puede recuperar el balón, los siguientes números corresponden a los índices de los jugadores que pueden recuperar el balón en dicha iteración, ordenados desde el más cercano al más lejano del punto de recuperación.

Finalmente, `getVelocidadJugador(int idx)`, `getRemateJugador(int idx)`, `getErrorJugador(int idx)`, `getVelocidadJugadorRival(int idx)`, `getRemateJugadorRival(int idx)` y `getErrorJugadorRival(int idx)` retornan las aptitudes de mis jugadores y los rivales.

Todos estos métodos nos entregan información útil para tomar decisiones, y así poder indicar qué acciones deberán realizar los jugadores. Estas acciones serán la lista de comandos que retornara el método ejecutar. Sólo se tienen dos comandos disponibles; `ComandoIrA` y `ComandoGolpearBalon`.

Algunos Ejemplos de uso:

```
new ComandoIrA(0,new Posición(0,0)); /*Se le indica al jugador de índice 0 que debe dirigirse a la posición de coordenadas 0,0. Los índices de los jugadores van de 0 hasta 10. La posición 0,0 corresponde al centro del campo de juego.*/
```

```
new ComandoGolpearBalon(6, new Posición(60,30), 1, true); /*Se le indica al jugador de índice 6 que remate con dirección destino a las coordenadas (60;30), con fuerza 1, y que el remate sea por alto. La fuerza que corresponde al tercer parámetro y acepta valores double entre 0 y 1, cuando sea 1 el remate se realizara a la velocidad máxima que puede rematar el jugador de índice 6. El ultimo parámetro al ser true indica que el remate se realizara por alto. (El remate por alto se realiza con un ángulo vertical igual a Constantes.ANGULO_VERTICAL */
```

```
new ComandoGolpearBalon(6, 15, 1, true); /* Otra forma de indicar el destino es informar directamente cual es ángulo de remate, en este caso es 15° */
```

```
new ComandoGolpearBalon(6, 15, 1, 25); /* Otra forma es indicar explícitamente el ángulo vertical, 25 grados en este caso. El ángulo vertical puede estar entre 0 y Constantes.ANGULO_VERTICAL_MAX. */
```

```
new ComandoGolpearBalon(6); /* Este constructor se usa para simular la acción de avanzar con el balón.*/
```

DATOS ÚTILES SOBRE LA LOGICA DE UN PARTIDO

En cada partido se ejecutaran n iteraciones, en cada iteración se enviara a las tácticas la situación del partido, y seguido a esto las tácticas retornaran la lista de los comandos que especifiquen el método ejecutar. De acuerdo a esto se modificará el estado del partido para la siguiente iteración. En especifico se ejecutarán [Constantes.ITERACIONES](#) iteraciones.

Un jugador sólo puede ir a una dirección y sólo puede rematar de una forma durante la iteración. Es decir, se ejecutará sólo un comando “ir a” y sólo un comando “golpear balón” para cada jugador en cada iteración.

Cuando ocurra que en una misma iteración se indique más de un [ComandoIrA](#) para el mismo jugador, los primeros comandos se ignorarán y sólo se considerará el último. Análogo sucede con el comando [ComandoGolpearBalon](#).

Es importante comprender que todos los comandos actúan sobre el mismo instante de tiempo. Por esto no es válido pensar que primero ejecuto un comando “ir a” para acercarme al balón y luego que estoy cerca ejecuto golpear el balón.

Un jugador está en condiciones de rematar si se encuentra a una distancia menor o igual al balón que [Constantes.DISTANCIA_CONTROL_BALON](#) y además que el balón se encuentre a una altura menor o igual que [Constantes.ALTURA_CONTROL_BALON](#). Para evitar el caso de porteros perfectos, se aplica también una probabilidad de control del balón que depende de la velocidad del balón.

Dentro de los jugadores propios y rivales que estén en condiciones de rematar se seleccionará uno al azar y sólo es ese quien rematará. Al remate se le aplicara un error angular que dependerá del error del jugador que rematara.

Para evitar excesivos rebotes, un jugador que ya remató, sólo podrá volver a rematar en [Constantes.ITERACIONES_GOLPEAR_BALON](#) iteraciones después.

Cada vez que el balón sale del campo de juego, deberá sacar el equipo rival del que lanzó el balón, en la posición donde salió el balón. Si un equipo lanza el balón por detrás de su propio arco se produce un tiro de esquina, que no es más que un saque normal pero en la posición de córner. En los saques de costado se reduce la potencia de remate a un 75%.

Durante las iteraciones entre que sale el balón y cuando se realiza el saque el tiempo no avanza, y los jugadores del equipo que no saca se alejarian en un radio [Constantes.DISTANCIA_SAUQUE](#) desde la posición de saque. Sin embargo, si pasa un tiempo excesivo y no se realiza el saque, se multara al equipo que no sacó, permitiendo sacar al otro equipo.

A continuación se muestra un código ejemplo de implementación del método ejecutar:

```
//Lista de comandos
LinkedList<Comando> comandos = new LinkedList<Comando>();

@Override
public List<Comando> ejecutar(SituacionPartido sp) {
    //Limpia la lista de comandos
    comandos.clear();
    //Obtiene las posiciones de tus jugadores
    Posicion[] jugadores = sp.misJugadores();
    for (int i = 0; i < jugadores.length; i++) {
        //Ordena a cada jugador que se ubique segun la alineacion1
        comandos.add(new ComandoIrA(i, alineacion1[i]));
    }
    //Si no saca el rival
    if (!sp.sacaRival()) {
        //Obtiene los datos de recuperacion del balon
        int[] recuperadores = sp.getRecuperacionBalon();
        //Si existe posibilidad de recuperar el balon
        if (recuperadores.length > 1) {
            //Obtiene las coordenadas del balon en el instante donde
            //se puede recuperar el balon
            double[] posRecuperacion = sp.getTrayectoria(recuperadores[0]);
            //Recorre la lista de jugadores que pueden recuperar
            for (int i = 1; i < recuperadores.length; i++) {
                //Ordena a los jugadores recuperadores que se ubique
                //en la posicion de recuperacion
                comandos.add(new ComandoIrA(recuperadores[i],
                    new Posicion(posRecuperacion[0], posRecuperacion[1])));
            }
        }
    }
    //Instancia un generador aleatorio
    Random r = new Random();
    //Recorre la lista de mis jugadores que pueden rematar
    for (int i : sp.puedenRematar()) {
        //Si el jugador es de indice 8 o 10
        if (i == 8 || i == 10) {
            //condicion aleatoria
            if (r.nextBoolean()) {
                //Ordena que debe rematar al centro del arco
                comandos.add(new ComandoGolpearBalon(i, Constantes.centroArcoSup, 1, 12 + r.nextInt(6)));
            } else if (r.nextBoolean()) {
                //Ordena que debe rematar al poste derecho
                comandos.add(new ComandoGolpearBalon(i, Constantes.posteDerArcoSup, 1, 12 + r.nextInt(6)));
            } else {
                //Ordena que debe rematar al poste izquierdo
                comandos.add(new ComandoGolpearBalon(i, Constantes.posteIzqArcoSup, 1, 12 + r.nextInt(6)));
            }
        } else
    }
}
```

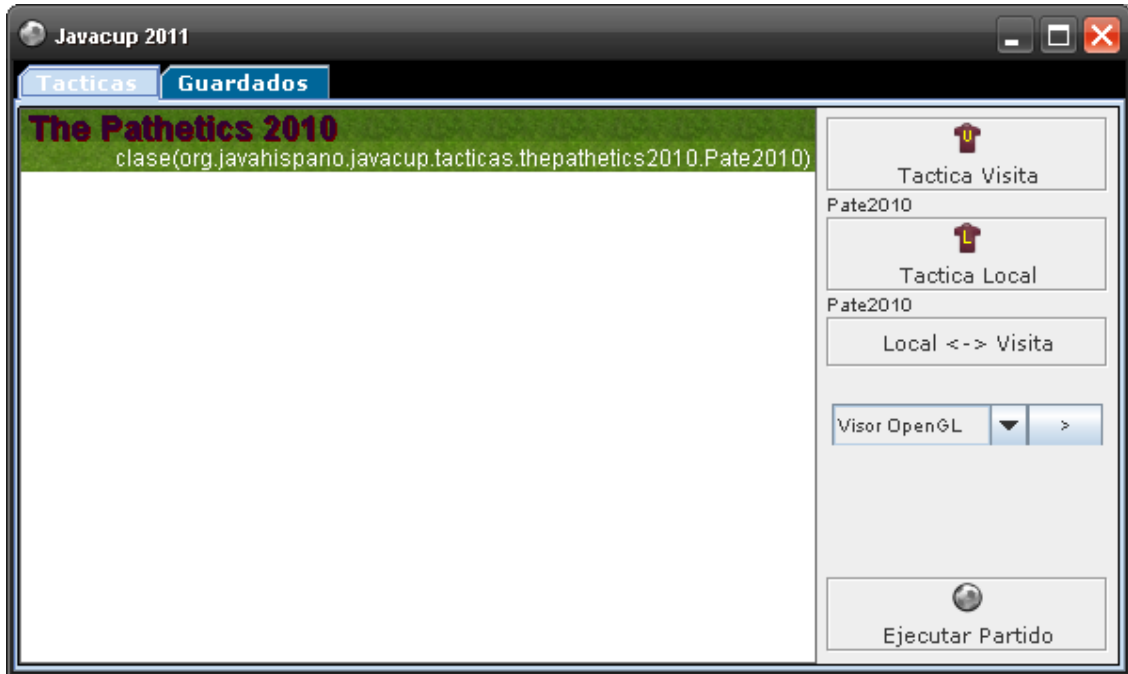
```

{
    //inicia contador en cero
    int count = 0;
    int jugadorDestino;
    //Repetir mientras el jugador destino sea igual al jugador que remata
    while (((jugadorDestino = r.nextInt(11)) == i
        //o mientras la coordenada y del jugador que remata
        //es mayor que la coordenada y del que recibe
        || jugadores[i].getY() > jugadores[jugadorDestino].getY())
        //Y mientras el contador es menor a 20
        && count < 20) {
        //incrementa el contador
        count++;
    }
    //Si el receptor del balon es el que remata
    if (i == jugadorDestino) {
        while ((jugadorDestino = r.nextInt(jugadores.length)) == i) {
        }
    }
    //Ordena que el jugador que puede rematar que de un pase
    //al jugador destino
    comandos.add(new ComandoGolpearBalon(i, jugadores[jugadorDestino], 1, r.nextInt(45)));
}
}
//Retorna la lista de comandos
return comandos;
}

```

VISOR DE PARTIDOS

Para usar el visor de partidos debes ejecutar la clase `org.javahispano.javacup.principal.Principal`.



En la pestaña “Tácticas” aparecerán las tácticas que agregues al proyecto.

Al seleccionar una táctica y presionar “Tactica Visita” asignas el equipo visita, análogo para asignar el local.

Con el botón “Local <-> Visita” se invierte el local y la visita.

En el comboBox seleccionas visualizar el partido mediante el Visor OpenGL o mediante el visor normal.

Con el botón “>” configuras los parámetros de la visualización.

Finalmente “Ejecutar Partido” para ver el partido.

Durante el juego al presionar + y – se activa el zoom.

Al presionar F1 se activa o desactiva el estadio.

Al presionar F2 se activa o desactiva el entorno.

Al presionar F3 muestra los frames por segundo.

Al presiona F4 activa o desactiva los números y nombres.

Por último en la pestaña “Guardados” están los partidos que has salvado, al visualizar uno de estos partidos puedes, avanzar, retroceder (con las flechas), marcar el una sección (con la tecla “inicio” y “fin”) y puedes cortar la sección (con tecla “supr”).