Condizioni iniziali e specifiche della gara:
Le simulazioni dovranno essere fatte in FreeMode.
La gara dovrà iniziare con lo SPHERES blu fermo posizionato nell'origine, mentre lo SPHERES rosso (posizionato inizialmente nel punto X=0, Y=0.5m, Z=0) dovrà muoversi lungo una traiettoria che tocchi i punti:

1. A=(0.5m, 0.5m, 0)
2. B=(0.5m, -0.5m, 0)
3. C=(-0.5m, -0.5m, 0.5m),
4. ritornando e fermandosi al punto di partenza.

Si considera terminata la traiettoria nel momento in cui la distanza dal punto di partenza scende al di sotto di 1cm o, nel caso questo non accada, quando la traiettoria raggiunge la distanza minima dal punto di partenza.

Verranno assegnati:

- 10 punti ad ognuno;
- 0.1 punto in meno per ogni 1s di maggior tempo rispetto al minore assoluto;
- 0.1 punto in meno per ogni 1cm/s di velocità al termine della traiettoria;
- 0.2 punti in meno per ogni 1cm di distanza minima da ciascuno dei quattro punti indicati (A, B, C, punto di partenza).
- 1 punto in meno per ciascuna situazione anomala macroscopica (ad es. il movimento dello SPHERES sbagliato; traiettoria corretta ma sul piano sbagliato, ecc.)

Il punteggio che in certe situazioni viene visualizzato durante la simulazione NON ha alcun valore, in quanto si riferisce ad altre gare.

Soluzione di Daniele Giudice e Siria Domeniconi: preselection (parte 1)

```
//------------------------------Controllo distanza
ZRState myState;
float v[3];
float distMin;
float distance;
//-----------------------------------Matrice punti
float points[4][3];

//-----------------------------------Circonferenza
float raggio, omega, alpha,
      cosalpha, sinalpha;
float w[3], c[3];
bool circumference;

//-----------------------------------Tratto finale
int currentPoint;
float speedTarget;


void init(){
//--------------------PUNTI DA RAGGIUNGERE
//--------------------Distanza P-A: 0.5 m
//-------------------Punto A
    points[0][0] = 0.5;
    points[0][1] = 0.5;
    points[0][2] = 0;
//--------------------Distanza A-B: 1 m
//-------------------Punto B
    points[1][0] =  0.5;
    points[1][1] = -0.5;
    points[1][2] =  0;
//-----------------------------------Distanza B-C: 1.118033988 m
//-----------------------------------Punto C
    points[2][0] = -0.5;
    points[2][1] = -0.5;
    points[2][2] =  0.5;
//-----------------------------------Distanza C-P: 1.224744871 m
//-----------------------------------Punto P
    points[3][0] = 0;
    points[3][1] = 0.5;
    points[3][2] = 0;
//-----------------------------------CONTROLLO DISTANZA
    distMin = 0.05;

//-----------------------------------CIRCONFERENZA DA 'P' A 'B'
//-----------------------------------x^2+y^2 - 0.5x - 0.25 = 0
//-------------------------raggio = sqrt(0.3125) = 0.559016994374
//-----------------------------------centro(0.25 , 0)
    c[0] = 0.25;
    c[1] = 0.0;
    c[2] = 0.0;
    raggio = 0.529;
    alpha  = 0.15;
    omega  = 0.15;
    cosalpha =  cosf(alpha);
    sinalpha = -sinf(alpha);
    circumference = false;
//-----------------------------------TRATTO DA 'B' A 'P'
    currentPoint = 2;          //---Comincia dal punto C

}           //--- fine init
```

Soluzione (parte 2)

```
void loop(){
    if (distanceToPoint(3) < 0.16f
    &&  circumference){
        v[0]=myState[3]*-50.0f;
        v[1]=myState[4]*-50.0f;
        v[2]=myState[5]*-50.0f;
        api.setForces(v);
    }
    else
    if ( !circumference ){
       if(distanceToPoint(1) > distMin)
          circumference2D();
       else{
          circumference = true;
          c[0] = 0.0;
      }
    }
    else
          manualPositionTarget(2);
}                              //--- fine loop
```

Soluzione (parte 3)

```
//-----------------------------------Funzioni di servizio

void mathVecMultipl (float *v,  float *a,
                     float k,    int lenght){
    for(int i = 0; i<lenght ; i++)

        v[i] = a[i] * k;
}



float distanceToPoint(int point){

    api.getMyZRState(myState);
    mathVecSubtract(v, points[point], myState, 3);
    return  mathVecMagnitude(v,3);
}
```

```
//-----------------------------------------------------Funzione Circonferenza
void circumference2D(){
    api.getMyZRState(myState);
    v[0] = myState[0];
    v[1] = myState[1];
    v[2] = myState[2] = 0.0;

    mathVecSubtract (v,v,c,3);                  //----Traslazione centro
    mathVecNormalize(v,3);

    w[0] = c[0] + raggio * (v[0] * cosalpha - v[1] * sinalpha);
    w[1] = c[1] + raggio * (v[0] * sinalpha + v[1] * cosalpha);
    w[2] = 0.0;

    mathVecSubtract  (v, w, myState, 3);
    mathVecNormalize (v, 3);
    mathVecMultipl (v, v, omega*raggio, 3) ;
    api.setVelocityTarget(v);
}
```

```
//-----------------------------------------------Funzione  manualPositionTarget
void manualPositionTarget (float speed){
      distance = distanceToPoint (currentPoint);
      if (distance < distMin*2){
            currentPoint++;
            api.setVelocityTarget(c);
            return;
      }
      speedTarget = distance*speed - distance*distance*0.1;
      if(speedTarget > 0.055)
        speedTarget = 0.055;

      for (int i = 0; i < 3; i++)
            v[i] = speedTarget * (points[currentPoint][i] - myState[i]) /distance;

      api.setVelocityTarget(v);
}
```