Condizioni iniziali e specifiche della gara:

Le simulazioni dovranno essere fatte in FreeMode.

La gara dovrà iniziare con lo SPHERES blu posizionato inizialmente nel punto X=0, Y=-0.5m, Z=0, e lo SPHERES rosso posizionato inizialmente nel punto X=0, Y=0.5m, Z=0.

I due satelliti dovranno muoversi lungo una traiettoria che tocchi i punti:

Punto	SPHERES blu	SPHERES rosso
A	(-0.5m, -0.5m, 0)	(0.5m, 0.5m, 0)
B	(-0.5m, 0.5m, 0)	(0.5m, -0.5m, 0)
C	(0.5m, 0.5m, 0)	(-0.5m, -0.5m, 0)
D	(0.5m, -0.5m, 0)	(-0.5m, 0.5m, 0)
E	ritorno al punto di partenza.	ritorno al punto di partenza.

Si considera terminata la traiettoria nel momento esatto 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 1s di differenza fra i tempi in cui ciascuno dei due SPHERES termina la propria traiettoria;
- 0.2 punti in meno per ogni 1cm di distanza minima da ciascuno dei quattro punti indicati (A, B, C, D, E).
- 1 punto in meno per ciascuna situazione anomala macroscopica (ad es. il movimento dello SPHERES sbagliato; traiettoria corretta ma sul piano sbagliato; scambio dei colori; ecc.)

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

Soluzione proposta da Lorenzo Antonucci e Leonardo Lorenzoni
(2 programmi diversi per SPHERES **blue** e **red**)

preselezione_**blue** (parte 1)

```
ZRState myState;

float A[3], As[3], B[3], Bs[3],
      C[3], Cs[3], D[3], Ds[3],
      E[3], Es[3];

float PointTo[3];
float Ricerca;

int   m, q;
int   Punto;
float Delay;


void init(){

//--------- coordinate dei veri punti da raggiungere utilizzate con  Near_to

    A[0]=-0.50;   A[1]=-0.50;    A[2]= 0.00;
    B[0]=-0.50;   B[1]= 0.50;    B[2]= 0.00;
    C[0]= 0.50;   C[1]= 0.50;    C[2]= 0.00;
    D[0]= 0.50;   D[1]=-0.50;    D[2]= 0.00;
    E[0]=-0.50;   E[1]=-0.50;    E[2]= 0.00;

//--------- coordinate più lontane dei veri punti da raggiungere utilizzate
//--------- con setPositionTarget per non rallentare troppo presto ---------
//--------- individuate per tentativi, osservando le simulazioni ----------

   As[0]=-1.00;  As[1]=-0.50;   As[2]= 0.00;
   Bs[0]= 0.07;  Bs[1]= 1.15;   Bs[2]= 0.00;
   Cs[0]= 0.50;  Cs[1]=-1.50;   Cs[2]= 0.00;
   Ds[0]=0.382;  Ds[1]=-0.67;   Ds[2]= 0.00;
   Es[0]=-0.60;  Es[1]=-0.11;   Es[2]=-0.007;

    Delay   = 0.04;
    Punto   = 0;
    Ricerca = 2.00;
}                                          //--- fine init()
```

preselezione_**red** (parte 1)

```
ZRState myState;

float A[3], As[3], B[3], Bs[3],
      C[3], Cs[3], D[3], Ds[3],
      E[3], Es[3];

float PointTo[3];
float Ricerca;

int   m, q;
int   Punto;
float Delay;


void init(){

//--------- coordinate dei veri punti da raggiungere utilizzate con  Near_to

    A[0]= 0.50;   A[1]= 0.50;    A[2]= 0.00;
    B[0]= 0.50;   B[1]=-0.50;    B[2]= 0.00;
    C[0]=-0.50;   C[1]=-0.50;    C[2]= 0.00;
    D[0]=-0.50;   D[1]= 0.50;    D[2]= 0.00;
    E[0]= 0.50;   E[1]= 0.50;    E[2]= 0.00;

//--------- coordinate più lontane dei veri punti da raggiungere utilizzate
//--------- con setPositionTarget per non rallentare troppo presto ---------
//--------- individuate per tentativi, osservando le simulazioni ----------

   As[0]= 1.00;  As[1]= 0.50;   As[2]= 0.00;
   Bs[0]=-0.07;  Bs[1]=-1.15;   Bs[2]= 0.00;
   Cs[0]=-0.50;  Cs[1]= 1.50;   Cs[2]= 0.00;
   Ds[0]=-0.382; Ds[1]= 0.67;   Ds[2]= 0.00;
   Es[0]= 0.60;  Es[1]= 0.11;   Es[2]=-0.007;

    Delay   = 0.04;
    Punto   = 0;
    Ricerca = 2.00;
}                                          //--- fine init()
```

preselezione_**blue** (parte 2)

```
bool Near_to (float target[]){
       api.getMyZRState(myState);
       if     (myState[0] > target[0] - Delay
       &&      myState[0] < target[0] + Delay)
         if   (myState[1] > target[1] - Delay
         &&    myState[1] < target[1] + Delay)
           if(myState[2] > target[2] - Delay
           && myState[2] < target[2] + Delay){
              return 1;
           }
       return 0;
}
void loop(){
     if (Punto==0){
          if (Near_to(A)==1){
               Punto = 1;
          }
          else
               api.setPositionTarget(As);
     }
     if (Punto==1){
          if (Near_to(B)==1){
               Punto=2;
          }
      else
               api.setPositionTarget(Bs);
     }
   if (Punto==2){
      if (Near_to(C)==1){
         Punto=3;
      }
      else
               api.setPositionTarget(Cs);
   }
   if (Punto==3){
      if (Near_to(D)==1){
         Punto=4;
      }
      else
               api.setPositionTarget(Ds);
   }
   if (Punto==4){
               api.setPositionTarget(Es);
    }
}                                          //--- fine loop()
```

preselezione_**red** (parte 2)

```
bool Near_to (float target[]){
       api.getMyZRState(myState);
       if     (myState[0] > target[0] - Delay
       &&      myState[0] < target[0] + Delay)
         if   (myState[1] > target[1] - Delay
         &&    myState[1] < target[1] + Delay)
           if(myState[2] > target[2] - Delay
           && myState[2] < target[2] + Delay){
              return 1;
           }
       return 0;
}
void loop(){
     if (Punto==0){
          if (Near_to(A)==1){
               Punto = 1;
          }
          else
               api.setPositionTarget(As);
     }
     if (Punto==1){
          if (Near_to(B)==1){
               Punto=2;
          }
      else
               api.setPositionTarget(Bs);
     }
   if (Punto==2){
      if (Near_to(C)==1){
         Punto=3;
      }
      else
               api.setPositionTarget(Cs);
   }
   if (Punto==3){
      if (Near_to(D)==1){
         Punto=4;
      }
      else
               api.setPositionTarget(Ds);
   }
   if (Punto==4){
               api.setPositionTarget(Es);
    }
}                                          //--- fine loop()
```