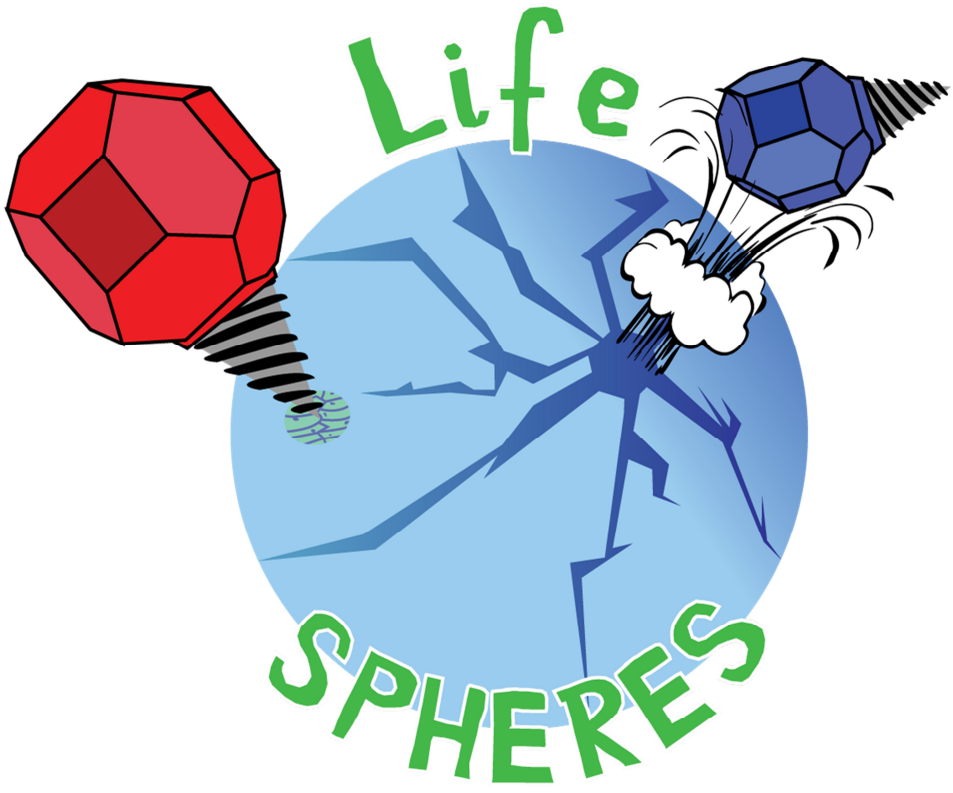
# ZERO ROBOTICS

## HIGH SCHOOL 2017

# GAME MANUAL

Version: Italian-1.0




***Living-microorganisms Investigation of Enceladus SPHERES Program***

ZERO ROBOTICS HIGH SCHOOL 2017

TO: All Zero Robotics Teams

RE: ATTENTION: CALLING TEAMS TO FIND LIFE ON ENCELADUS!

The SPHERES program has been tasked with a unique opportunity and urgently needs space engineers to study the status of life on Enceladus by gathering ice samples containing viable microorganisms. As you may have heard, scientists have discovered life on Enceladus, a moon of Saturn. The life forms found are various forms of bacteria. We wish to study these bacteria to see how they differ from life forms on Earth and to understand what humans might need to be able to live on Enceladus.

The *Living-microorganisms Investigation of Enceladus* (LIFE) program seeks to traverse the dangerous surface of Enceladus's southern pole and drill ice samples. Your priority is to secure the best samples possible.

Program the satellites to drill up samples and bring them back to the base station. Analyze these samples in order to find the region with the richest concentration of bacteria. Keep in mind that the beneath the surface of the southern region of Enceladus exist large amounts of high pressure gases, therefore your drilling could activate powerful geysers. While the SPHERES are strong enough to withstand damage from the geysers, the force of the fast moving gases will push them off-course.

Also, be aware that we are not the only team in Enceladus. The Enceladus Vitality Inspection and Liquefaction group also is there. Their nefarious goal is to melt the samples to sell them as gourmet alien water! It is of paramount importance that you collect more bacteria samples, by getting the highest concentrations possible, than this rival team.

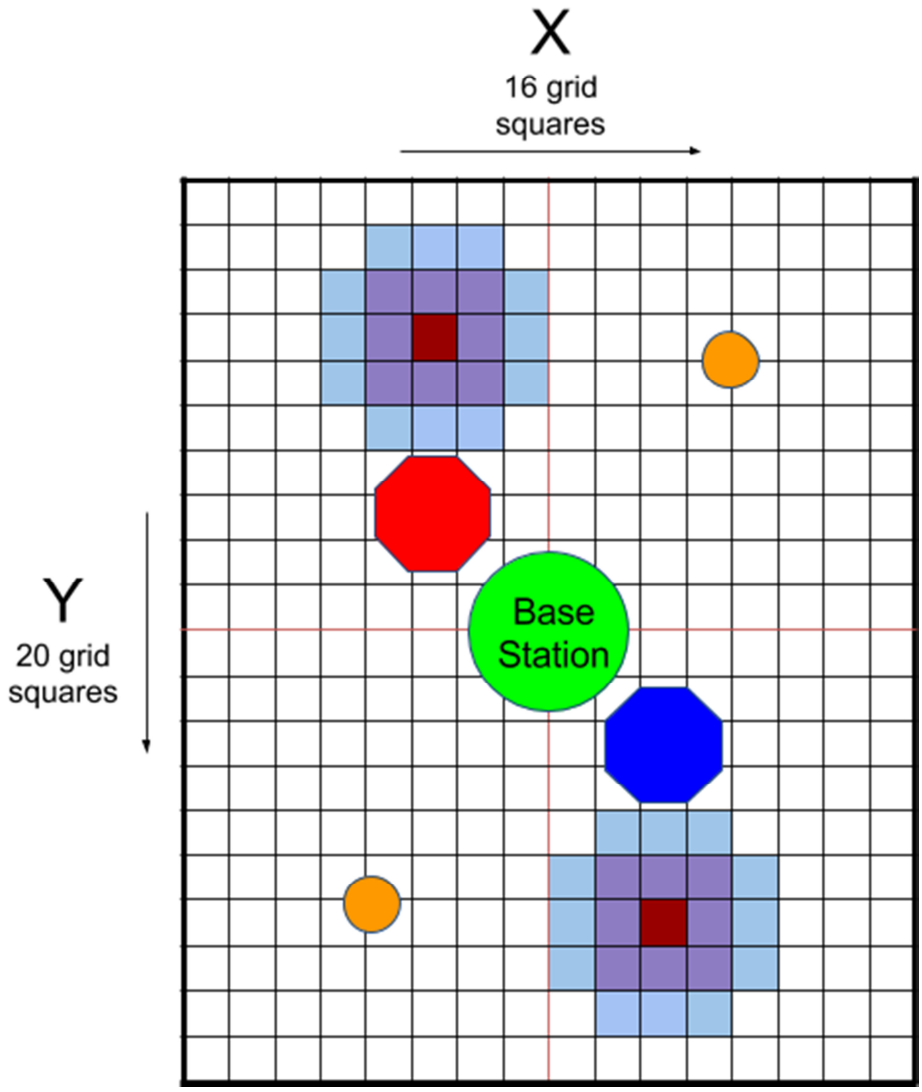
Good luck to all participating space engineers.

Alvar Saenz-Otero
SPHERES Lead




# 1 Game Overview

The game centers around finding the highest concentration *Micro-organism Zone*, collecting *Biological Samples* by *drilling*, and delivering the samples to the *Base Station*, while avoiding creating *Geysers*. Points are generated by drilling for biological samples and returning them to the common base station. The playing field "surface" has been discretized into a matrix of 16x20 sample squares 0.08m on each side. All drilling within the area of a sample square is considered as drilling for the same sample in the same square. The Game Overview figure, below, shows an overview of the playing field.




Game Overview

While the full playing field has a low concentration of microorganisms, there are two areas in the playing field where the microorganism concentration in the ice is highest. The concentration decreases with distance from these points. The location of the high-concentration area changes randomly between each match.

To get points, the SPHERES satellite must be commanded to *drill* over a sample square, which is done by rotating 90° about the Z axis of the playing field. The concentration of a sample square can be obtained in three ways: (1) pick-up a portable *analyzer* and do remote analysis without picking up a sample; (2) pick-up a portable *analyzer* and pick-up a sample, and analyze it *in-situ*; or (3) pick-up a sample and deliver it to the Base Station where it will be automatically analyzed. Successful delivery to the Base Station grants the most points.

Drilling a sample square weakens the surface of that square, creating an increasing probability of activating a geyser on that square. The geysers will get activated if a sample square is drilled four or more times. A geyser will push the SPHERE away from it, overriding any commanded motion by the players' code. A geyser activates for a limited period of time, but while it is active, it will affect both satellites. A geyser hit will make the satellite drop the last sample it picked up.

A SPHERES satellite can carry at most five samples at one time. The samples may be delivered to the Base Station or can be discarded anywhere in the playing field (when discarded they are lost completely and cannot be picked up again) to free up sample spaces.

Matches of LIFE-SPHERES are played between two players, controlled by programs written by two separate teams. Each team will compete to have the most points when the match time is up. Each match lasts 180 seconds.

# 2 Gameplay

In order to be victorious over the opposing team, each satellite should collect the most biological samples, return them to the base station, and avoid geysers, all while managing their fuel, their time, and their location on the gameplay area.

***Note: Satellite position relative to all game features is determined by the location of the center of the satellite as stored in the state variables (ZR state or SPHERES state) unless indicated otherwise.***

## 2.1 Playing Field

The playing field is determined by the size of the area available for the SPHERES satellites to move inside the International Space Station. The simulation creates an *interaction zone* of the same size. If players leave the Interaction Zone, they are considered out of bounds.

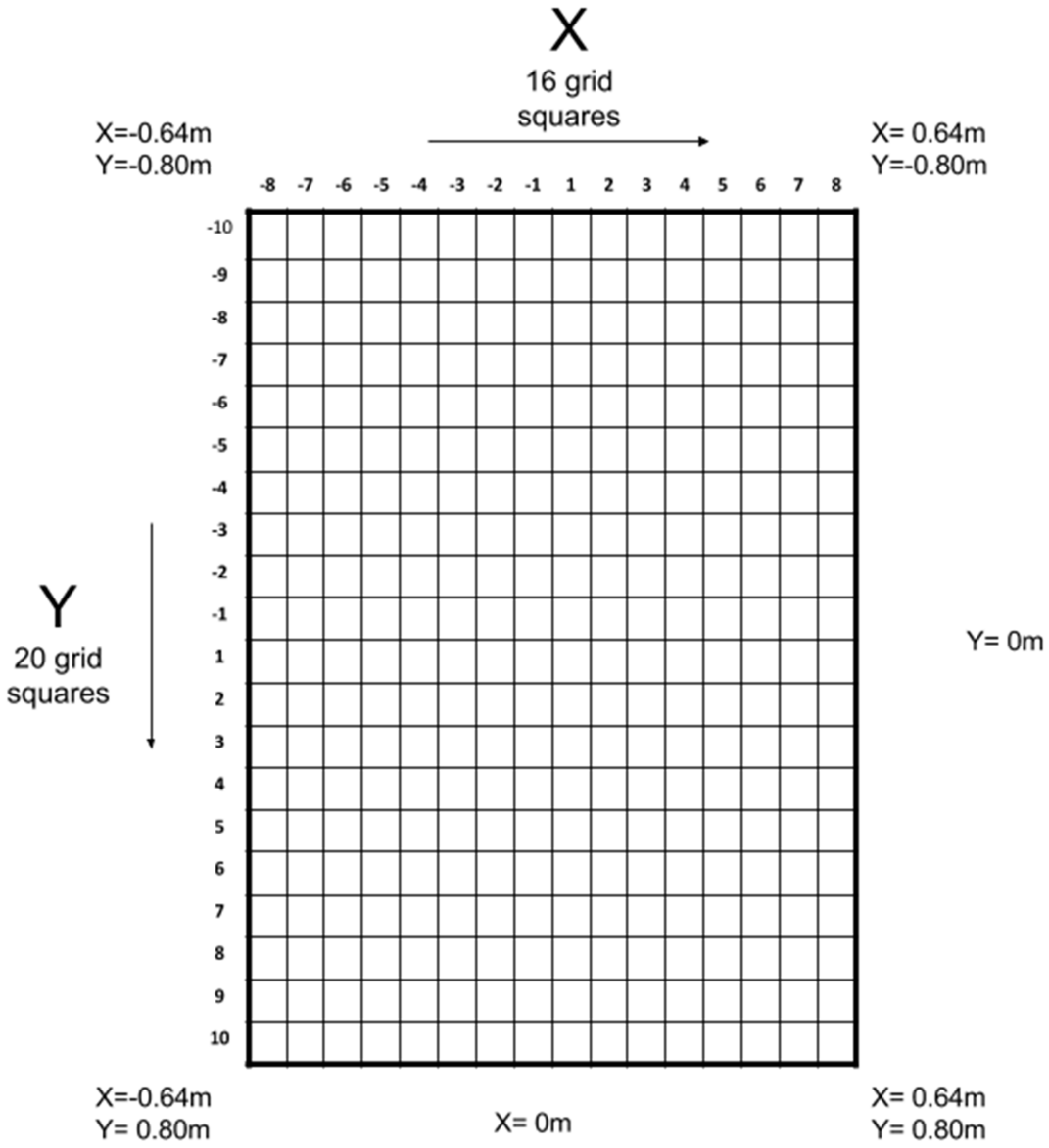
The Interaction Zone for the game has the following dimensions:

Interaction Zone Dimensions

| | Dimension |
|---|---|
| **X [m]** | [-0.64 : +0.64] |
| **Y [m]** | [-0.80 : +0.80] |
| **Z [m]** | [-0.64 : +0.64] |

A player will be penalized a portion of its fuel allocation for every second it remains out of these bounds.

For LIFE-SPHERES, the interaction playing field has been discretized into 0.08m cubes, creating a limited number of sample squares that can be drilled. The center of all the squares is within the interaction zone, but the edges of the outermost sample squares match the limits of the interaction zone. The sample square grid used for LIFE-SPHERES is presented in the figure below:




Playing Field Area and Interaction Zone Dimensions

The Z Axis has the same dimensions as the X Axis (+/- 0.64m) and is also divided into 16 sample squares. *Note: the Z Axis is aligned in the same way as with the ISS: +Z points towards Earth, which usually feels as "down".*

The function `pos2square` can be used to obtain the sample square index of the indicated current location in meters, for example the location of the satellite obtained from the ZR State Vector.


The function `square2pos` returns the position, in meters, of the center of the indicated square using the coordinated shown above.

### 2.1.1 Initial Position

The Blue and Red SPHERES satellites are deployed to the same initial position every game as follows:

Initial Positions

| | Initial Position |
|---|---|
| **Blue** | |
| **X [m]** | 0.2 |
| **Y [m]** | 0.2 |
| **Z [m]** | 0.0 |
| **Red** | |
| **X [m]** | -0.2 |
| **Y [m]** | -0.2 |
| **Z [m]** | 0.0 |

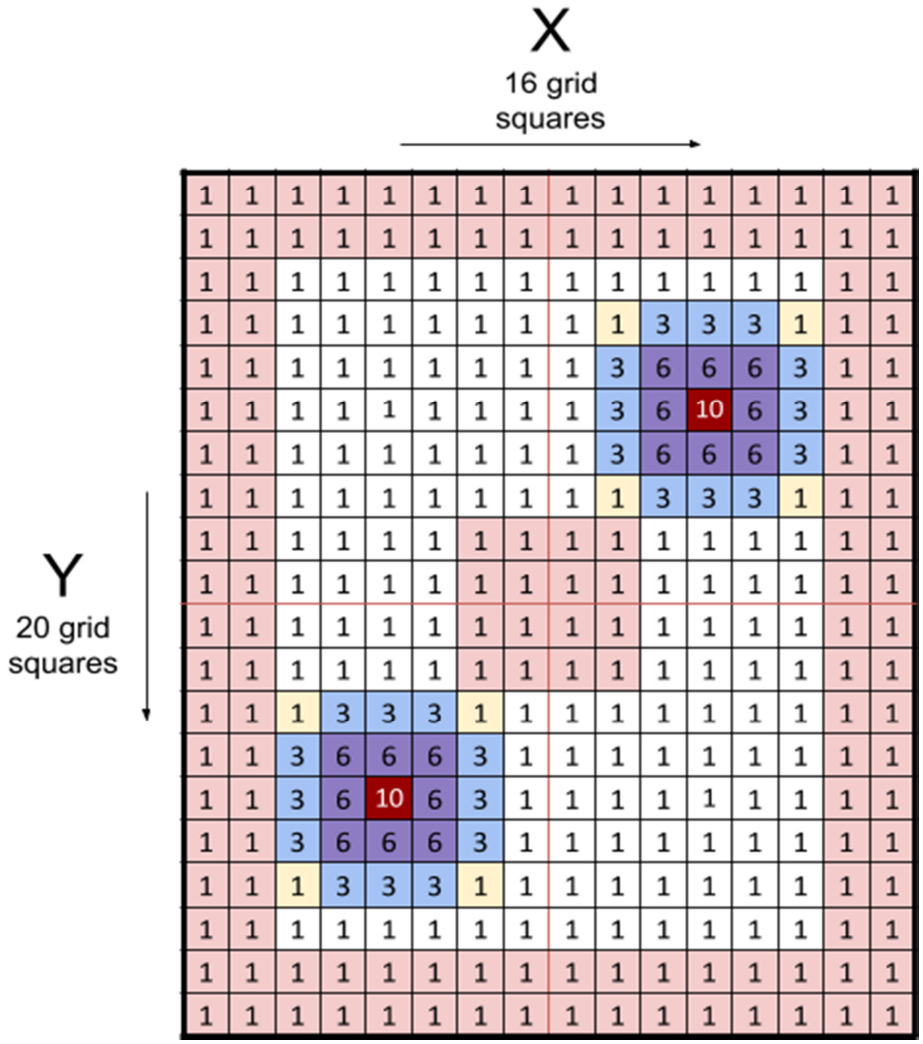
The satellite radius is 0.11m.

## 2.2 Microorganism Zones

The center of the Microorganism Zones (100% concentration) is randomly generated within the game grid for each match and may be located in any sample square except those in the keep-out zones shown in the figure below of [X,Y] = [-2..2, -2..2] and [|X|,|Y|] = [7..8, 9..10] in sample-square grid coordinates (see Section 2.1). The squares immediately next to the 100% concentration have 60% and those two squares away have 30%. The rest of the grid has 10%. The two 100% concentration zones will always be mirrored about the [X=0, Y=0] center of the XY plane.

Section 2.2.1 details the Terrain Z geometry.

The figure below shows an example of a microorganism concentration zone at the start of a match:




Example Microorganism Concentration Zones (10=100%, 6=60%, 3=30%, 1=10%) and Keep-out Area (pink) of 100% Concentration Sample Square.

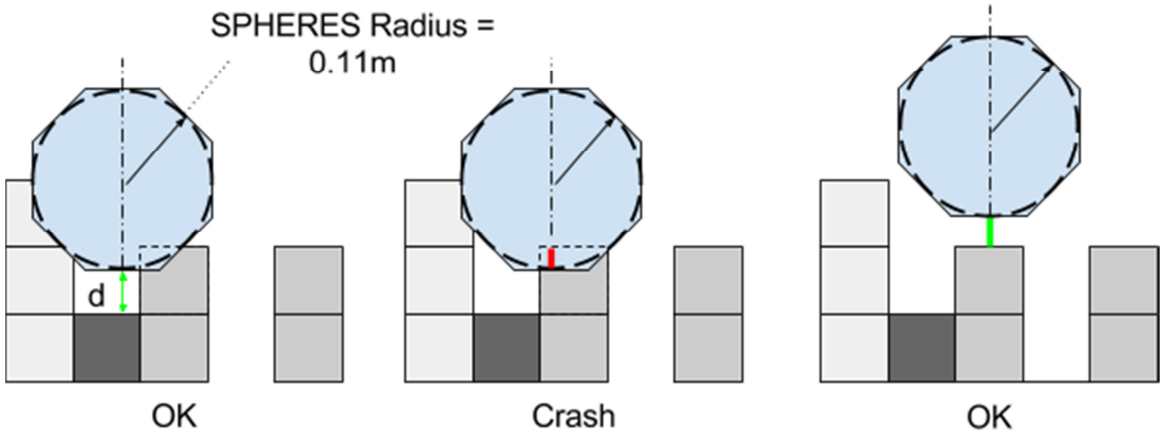
*Note: the outside corners of the 30% sample squares (yellow) may be either 10% or 30% based on rounding differences during the zone allocation function.*

The player will need to analyze the samples in order to figure out where the highest concentration is during each match.

### 2.2.1 3D Terrain

The game does not have a flat microorganism surface, rather it is three dimensional. The surface has randomly assigned heights for each square, at heights of Z = [0.40, 0.48, 0.56,

0.64]m. The heights are restricted to these discrete values. The satellite must not crash onto the surface in 3D. The figure below illustrates how crashing is defined in the game:




drill: 0 < d < 0.04m
crash: d < 0

*NOTE: d is measured from the closest point of the satellite to the projection of the center of the satellite to the surface*

As illustrated, it is possible for the “sides” of the satellite to make contact with a surface, as long as the “bottom” of the satellite does not make contact. Therefore, it is possible for the satellite to go into a “valley” and drill successfully, as long as its “bottom” edge stays within the “valley” at all times.

If a satellite crashes onto the 3D terrain, it will asses a 0.5 points/second penalty.

## 2.3 Biological Samples

There are three main steps to get points from a sample:

1. Drill
2. Analyze (optional)
    a. If the analyzer has been picked-up, call `analyzeTerrain` immediately
    b. If the analyzer has been picked-up, call `getConcentrations` as soon as a sample is picked-up
    c. If the player does not have an analyzer, take to Base Station to get concentration after dropping the item
3. Drop at Base Station

The satellite has space for at most five (5) samples at one time. Once the sample spaces are full, the satellite must drop the sample at either the Base Station to get points or anywhere in the field to discard the sample. If the sample is discarded, it cannot be picked up again by any satellite, it is lost completely.

### 2.3.1 Drilling / Collecting Samples

In order to pick up a a sample, a SPHERES must complete a “drilling” motion. The drilling task is started by calling `startDrill` with a translation speed less than 0.01m/s and rotation rate of under 0.04rad/s (~2.3°/s) while the **edge** of the SPHERES satellite is within 0.04m of the surface but does not crash onto the surface, with the satellite X Axis aligned within 11.25° of the XY plane; the Y and Z axis of the satellite can be pointing in any direction. *(Reminder: the SPHERES satellite radius is 0.11m, see Section 2.1.1.)* The figure below illustrates the required start drill conditions:

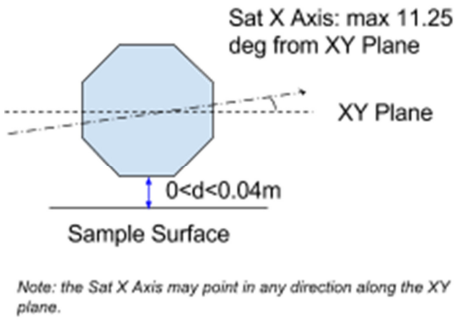



Illustration of initial drilling conditions.

The drill must be stopped before calling `startDrill` (the function `getDrillEnabled` may be used to check the status of the drill). After starting the drill, the satellite must rotate 90° about the **Z axis of the playing field**. The satellite must remain at the same sample square where the drill was started, maintaining a speed of under 0.01m/s, and maintaining the X Axis of the satellite aligned within 11.25° of the XY plane. The figure below shows an example of the drilling motion requirement:

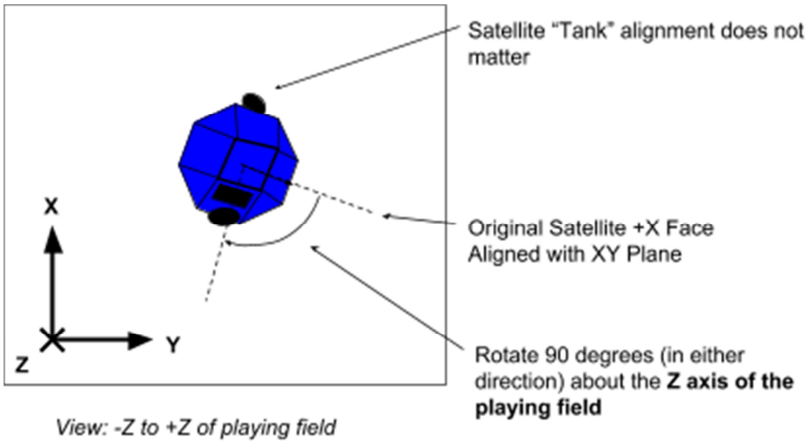



Illustration of drilling rotation about the Z axis of hte playing field.

The function `checkSample` can be used to determine when a sample is ready for pick-up. Once the sample is ready, the function `pickupSample` is called to collect the sample. The drill must be stopped by calling `stopDrill` before a new drill operation can start.

The satellite can hold at most five samples at any one time. The user can call `getSamplesHeld` to obtain the status of each of the five sample spaces (true if full; false if empty), or `getNumSamplesHeld` to obtain the total numbers of sample held at the time.

When a 100% concentration sample is picked-up, a bonus of 2.5 points is given every time.

The drill can be damaged if it is not used correctly by the following cases:

- Starting the drill when moving (translation or rotation) too fast (speed > 0.01m/s, rotation rate > 0.04rad/s)
- Translating once the drill has started (moving out of the square where the drill was started or gaining a speed > 0.01m/s)
- Not pointing the satellite X face along the XY plane (within 11.25°).

If the drill gets damaged, the player gets a point penalty. The function `getDrillError` can be used for the player to determine if drill damage has occurred. To stop the damage penalty, the drill must be disabled using `stopDrill` before starting again.

The number of drills performed at a sample square is incremented once a drill operation is complete, whether the sample is picked-up or not. However, a geyser may activate only after a sample has been picked up by calling `pickupSample`. (See Section 2.4.)

### 2.3.2 Sample Analyzers

The Sample Analyzers are always located as indicated in the table below:

Analyzer Positions

| | **Position** |
|---|---|
| **Analyzer 0** | X = 0.30m<br>Y = -0.48m<br>Z = -0.16m |
| **Analyzer 1** | X = -0.30m<br>Y = 0.48m<br>Z = -0.16m |

The analyzers are not assigned to a specific satellite; one satellite may pick-up both analyzers, although no direct benefit is gained from having both.

The analyzers are automatically picked-up by stopping at the location of the analyzers within a SPHERES satellite radius (0.11m) for 3 seconds with a translation velocity under 0.01m/s. The function `getAnalyzer` indicates if the analyzers are available for pickup. The function `hasAnalyzer` can be called to determine if the current player has picked up an analyzer.

These are *portable remote analyzers*, once the satellite has picked up an analyzer, it will be able to conduct remote analysis of the sample square directly under the satellite and in-situ analysis of any samples obtained, without having to go to the Base Station and without any conditions to conduct an analysis.

Once an analyzer has been picked up, the functions `analyzeTerrain` and `getConcentrations` become available.

`analyzeTerrain` can be called to obtain the concentration of the sample directly under the satellite (based on the [X,Y] state of the satellite), as long as its speed is under 0.04m/s. The function will return -1 if the analyzer has not been picked up and -2 if the speed is too fast. Note that calling `analyzeTerrain` does *not* pick-up a sample, it only provides remote analysis of the surface.

`getConcentrations` can be called in order to obtain the concentration information of any samples held. The function will return -2 if a sample is empty, or -1 if the player has not picked up an analyzer. If the player has picked up at least one analyzer, the function will reveal the concentration of the samples.

### 2.3.3 Base Station

The Base Station is where the SPHERES must deposit the samples. The Base Station is a circle of radius 0.24m centered at the origin. To deposit a sample, the SPHERES must be over the Base Station, moving with a translational speed < 0.01m/s and a rotation rate < 0.04rad/s (~2.3°/s) and the -X face of the satellite pointing in the -Z direction (within 11.25°). The function `atBaseStation` may be called to determine if the satellite is ready for delivery.

To deliver the sample, the player needs to execute the `dropSample` command. If the satellite is correctly positioned at the Base Station, the sample is delivered and the team will accumulate points. The function will return the concentration of that sample if it is dropped correctly at the Base Station. If the command is executed when the satellite is not correctly at the Base Station, the SPHERES will lose the sample (the sample is *discarded*) and the function will return 0.0.

## 2.4 Geysers

When drilling, the probability of a geyser being activated in that square is $\frac{D^3}{64}$, where D is the number of times that sample square has been drilled. The player may use the function `getDrills` to determine how many times a specific sample square has been drilled so far. The resulting probability of starting a geyser when picking up a sample is as follows:

Geyser Probability

| Drill # | Probability of geyser activated |
|---|---|
| 1 | 1/64 = 1.6% |
| 2 | 8/64 = 12.5% |
| 3 | 27/64 = 42.2% |
| 4+ | 64/64 = 100% |

When a geyser is created, it imparts a force of F=0.2N (maximum SPHERES satellite actual thrust) to *any* satellite that is within a SPHERES satellite radius (0.11m) of the sample square center. The force imparted overrides all commands in the Z direction and pushes the satellite towards -Z, without affecting the commanded X and Y motion. *Note: the fuel used to move the satellite due to geyser activity **does** count towards the player total fuel allocation (See Section 2.7.1)*. The force overrides user commands for 5 seconds.

*Note: the geysers impart only forces and not torque, therefore they do not have an effect on the attitude of the satellite.*


If a satellite is hit by a geyser it will drop the "last" [1] sample picked up at the location of the geyser. The sample is discarded and lost completely immediately when the geyser hits.

A geyser goes off on the same iteration when the sample is picked up and remains active for 30 seconds. At most 8 geysers may be active at any one time (if 8 geysers are already active, drilling operations will no longer create geysers).

The function `getNumGeysers` returns the number of active geysers. The function `getGeyserLocations` returns a matrix with the [X,Y] sample square coordinates of any active geysers. The matrix is not in any specific order (it depends on when and which player activated the geyser); any index of the matrix may have an active geyser, otherwise an invalid sample square coordinate [-100,-100] is returned. The function `isGeyserHere` may be called to determine if a geyser is active at a specific location of interest to the player.

## 2.5 Scoring

Teams get points by:

- Drilling successfully
- Delivering a sample to the Base Station
- Picking up a 100% sample

Teams lose points by:

- Drilling incorrectly
- Causing a geyser too close to the Base Station
- Crashing into the 3D terrain

The functions `getScore` and `getOtherScore` can be used to compare the scores between the players within the user code.

### 2.5.1 Drilling

Successful drilling a square and completing the sample pick-up (calling `pickupSample`) grants the following points based on the number of times that specific sample square has been drilled by any satellite:

| Drill Attempt | Points |
|---|---|
| 1 | 1 |

[1] The geyser will drop one of the samples picked-up at the location of the geyser, it will not necessarily be the the highest sample number.

| | |
|---|---|
| 2 | 2 |
| 3 | 3 |
| 4+ | 0 |

### 2.5.2 Dropping Off Sample

Successfully dropping a sample at the base stations gives the following points:

5*C + 2 where C = concentration of the sample analyzed (%)

### 2.5.3 Picking Up a 100% Sample

When a team successfully picks up a sample with 100% concentration, 2.5 bonus points are awarded immediately upon pick-up.

### 2.5.4 Incorrect Drilling

If the SPHERES drill is activated (by calling `startDrill)` and it violates the drilling conditions (see Section 2.3.1), these improper drilling techniques will damage the drill. This is quantified through a 0.25 point/second deduction while the drill is on (after `startDrill` is called and before `stopDrill` is called).

### 2.5.5 Geyser Too Close to Base Station

Creating a geyser close to the base station is very dangerous. Thus, any geyser created within two grid squares of the center of the play area (where the Base Station is located, sample squares [-2..2][-2..2]) will result in a 10 point deduction.

### 2.5.6 Crashing into 3D Terrain

When a satellite crashes into the 3D Terrain it will asses a 0.5 point/second penalty.

## 2.6 End of game

The game ends after 180 seconds. Whichever team has more points wins. In the unlikely case of a tie, whoever picked up the most number of samples will win. If a further tie is needed, then the team closest to the center of the base station at the end of the game will win.

## 2.7 SPHERES Satellites

Each team will write the software to command a SPHERES satellite to move in order to complete the game tasks. A SPHERES satellite can move in all directions using its twelve thrusters. The actual SPHERES satellites aboard the ISS, like any other spacecraft, have limited fuel (in this case liquid carbon dioxide), power (in this case AA battery packs), and computer (a digital-signal processor). These resources are limited and must be used wisely. Therefore, the players of Zero Robotics are limited in the use of these resourced by virtual limits within the game. The use of batteries is limited by having a fixed game time of 180 seconds. The other limitations are detailed below.

The *nominal* properties of the SPHERES satellites are summarized as follows:

| Property | Value | Units |
|---|---|---|
| Radius | 0.11 | M |
| Diameter | 0.22 | m |
| Mass | 4.0 | kg |
| Single Thruster Force | 0.11 | N |

These are called *nominal* properties because they are not exact. The Mass changes up to 0.2kg as fuel is consumed. The single thruster force is affected by how many thrusters are open at one time, varying up to 20% of the nominal force. In addition, while every attempt was made to align the thrusters with the satellite body axes, there are imperfections in the alignment (within 2°), therefore not all the force goes in the exact desired direction.

### 2.7.1 Fuel

Each player is assigned a virtual fuel allocation of 60 seconds of total accumulated thruster firing time. This is calculated by summing individual thruster firing during the game. Once the allocation is consumed, the satellite will not respond to the player SPHERES control commands. It will fire thrusters only to avoid leaving the Interaction Zone or colliding with the other satellite.

Any action that requires firing the thrusters (rotating, accelerating, decelerating), whether it was commanded by the player, due to activate collision avoidance or out-of-bounds breaking, or other penalties of the game play, will consume virtual fuel allocation.

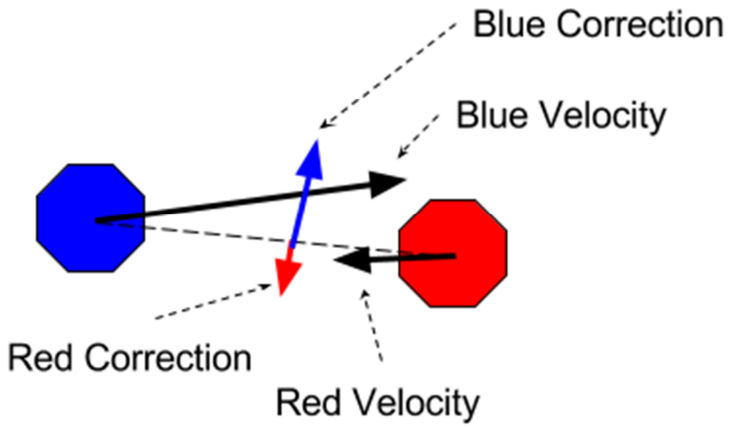
The function `getFuelRemaining` can be called to obtain the fraction of fuel still available during a match. The function returns 1.00 for a full tank, and then a fractional value (e.g. 0.50 =

half a tank) until reaching 0.0. Once the function returns 0.0 all user motion commands will be ignored.

### 2.7.2 Collision Avoidance

The Zero Robotics game implements a Collision Avoidance algorithm, ultimately in order to prevent actual contact between the satellites aboard ISS. The algorithm is also enabled in simulation.

The algorithm activates for any satellite that is moving with a translational velocity higher than 0.01m/s. If a satellite is moving below 0.01m/s the algorithm will not affect that satellite. If the algorithm detects an imminent collision between the satellites, it will overwrite the forces commanded to the satellites by adding a force perpendicular to the vector between the center of the satellites, with a magnitude proportional to the speed of the satellite. The direction of the perpendicular force is undefined, except that it is always opposite between the two satellites. The Collision Avoidance forces are illustrated in the figure below:




Collision Avoidance Algorithm (not to scale)

***Note:*** the fuel used in order to prevent collisions **does** count towards the total allocated fuel for the satellite.

### 2.7.3 Code Size

A SPHERES satellite can fit a limited amount of code in its memory. Each project has a specific code size allocation. When you compile your project with the “Code Size Estimate” menu option, the compiler will provide the percentage of the code size allocation that your project is using. Formal competition submissions require that your code size be 100% or less of the total allocation.

### 2.7.4 Noise

It is important to note that although the two competitors in a match will always be performing the same challenge and have identical satellites, the SPHERES simulations create noise similar to that experienced by the satellites aboard the ISS. This noise means two main things:

- The satellites will never know exactly where they are; their "estimate" of their location reflects that the sensors are not perfect, so at all times even if the satellite is supposed to be completely still, their location will vary approximately ±0.005m independently on every axis.
- The satellites will not thrust perfectly. While the thrusters use identical designs, each thruster has a slight variation in thrust, and the total thrust varies by the number of thrusters used at one time. This means that the thrust of the satellites may vary in general 10% and up to 20% in some cases.

This is fully intended as part of the challenge and reflects uncertainties in real aerospace engineering systems, which have imperfect dynamic models, sensors, and actuators. The best performing solutions will be those that prove to be robust to these variations and a wide variety of different initial conditions of the gameplay features.

### 2.7.5 Inter-satellite Communications

The satellites have the ability to communicate with each other using binary messages. The API functions `sendMessage` and `receiveMessage` may be used to send data between the satellites if different players decide to collaborate during a game.

# 3 Game API

The following table lists the functions available in this year's game. In order to use these functions, use the syntax `game.functionName(inputs)`, for example:

```
game.dropSample(1)
```

For the general Zero Robotics functions use the syntax `api.functionName(inputs)`, for example:

```
api.setPositionTarget(posTarget)
```

(unless they are math functions, which can be called without reference to the instance).

The generic ZR user API (in both C and MATLAB languages) is available at:
**http://static.zerorobotics.mit.edu/docs/tutorials/ZR_user_API_2017.pdf**

LIFE-SPHERES API Reference

| Name | Description |
|---|---|
| C: `void pos2square(float pos[3], int square[3])`<br><br>MATLAB: `square = pos2square(pos)` | Returns the sample `square` coordinates (see Section 2.1) for the position `pos[X,Y,Z]` in meters (e.g., the ZR state). |
| C: `void square2pos(int square[3], float pos[3])`<br><br>MATLAB: `pos = square2pos(square)` | Returns the position in meters of the center of the given sample `square` (see Section 2.1). |
| C: `float getTerrainHeight(float pos[2])`<br><br>MATLAB: `result = getTerrainHeightPos(pos)` | Returns the height of the terrain surface (playing field Z axis value) of the sample square at the given `pos[X,Y]` in meters. |
| C: `float getTerrainHeight(int square[2])`<br><br>MATLAB: `result = getTerrainHeightSquare(square)` | Returns the height of the terrain surface (playing field Z axis value) of the sample square at location `square[X,Y]` (see section 2.1 for grid coordinates). |
| C: `int hasAnalyzer()`<br><br>MATLAB: `result = hasAnalyzer()` | Returns the number of Sample Analyzers held by the SPHERES:<br>0 = no analyzers<br>1 = has only analyzer 0<br>2 = has only analyzer 1<br>3 = has both analyzers 0 and 1 |
| C: `void getAnalyzer(bool pickedUp[2])`<br><br>MATLAB: `pickedUp = getAnalyzer()` | Returns the status of the two analyzers in the boolean array `pickedUp`:<br>true if the analyzer has been picked up by either sat<br>false if the analyzer has not been picked up |
| C: `float analyzeTerrain()`<br><br>MATLAB: `result = analyzeTerrain()` | Returns the concentration of the sample square directly below the satellite (based on the satellite [X,Y] state), if the analyzer has been picked up and the satellite is moving slow enough (< 0.04m/s):<br>-2 = moving too fast<br>-1 = analyzer not picked up |

| | |
|---|---|
| | *float* = concentration of sample square |
| C: `int getDrills(float pos[2])`<br><br>MATLAB: `result = getDrillsPos(pos)` | Returns the total number of times drilling has been completed (sum of drills by both satellites) in the square that contains the position (e.g., from the satellite ZR state) indicated by the variable input `pos[X,Y]` in meters. *Note: even in 3D, only the [X,Y] coordinates are used, since the samples are in the plane Z=0.48.* |
| C: `int getDrills(int square[2])`<br><br>MATLAB: `result = getDrillsSquare(square)` | Returns the total number of times drilling has been completed (sum of drills by both satellites) in the square that contains the sample square indicated by the variable input `square[X,Y]` (see Section 2.1 for grid coordinates). *Note: even in 3D, only the [X,Y] coordinates are used, since the samples are in the plane Z=0.48* |
| C: `bool startDrill()`<br><br>MATLAB: `result = startDrill()` | Starts drilling the square under the current position of satellite.<br>Returns true if started successfully<br>false if drilling incorrectly, point penalty may result |
| C: `bool checkSample()`<br><br>MATLAB: `result = checkSample()` | Returns true if the sample currently being drilled is ready for pick-up.<br>Returns false if the drill was not started or the drilling motion requirements are not complete. |
| C: `int pickupSample()`<br><br>MATLAB: `result = pickupSample()` | Attempts to collect sample. Must be run while drill is running.<br>Returns index into sample array (sample #) if collected<br>-1 if sample space is full<br>-2 for incorrect drilling procedure / drill is off |
| C: `void getSamplesHeld(bool samples[5])`<br><br>MATLAB: `samples = getSamplesHeld()` | Populates a list of booleans indicating whether a sample space has been filled or is empty:<br>Returns true if a sample space is full (picked-up OK)<br>false if a sample space is empty |
| C: `int getNumSamplesHeld()`<br><br>MATLAB: `result = getNumSamplesHeld()` | Returns the total number of samples held (0 = none, up to 5). |
| C: `void getConcentrations(float concentrations[5])`<br><br>MATLAB: `concentrations = getConcentrations()` | Returns the list of concentrations of each of the sample spaces that have been analyzed:<br>Returns concentration of sample<br>-1 if the sample has not been analyzed<br>-2 if the sample is empty |
| C: `float dropSample(int sampleNumber)`<br><br>MATLAB: `result = dropSample(sampleID)` | Attempts to drop the specified sample (0, 1, 2, 3 or 4). If the satellite meets the condition for a base station delivery and the sample is not empty, the function returns the sample concentration. If there is no sample, or the satellite is not at the base station, it returns 0.<br>Returns concentration if dropped correctly at base<br>0.0 if not dropped correctly or sample empty |
| C: `bool atBaseStation()`<br><br>MATLAB: `result = atBaseStation()` | Returns true if the satellite currently meets all the requirements to drop a sample at the Base Station for points. |
| C: `void stopDrill()`<br><br>MATLAB: `stopDrill()` | Stops drilling and resets any drill errors if they occurred. |
| C: `bool getDrillEnabled()`<br><br>MATLAB: `result = getDrillEnabled()` | Returns true if the drill is currently enabled, false if it is disabled. |
| C: `bool getDrillError()`<br><br>MATLAB: `result = getDrillError()` | Returns true if a drill error has occurred since `drillStart` was called and has not been reset by calling `drillStop`. |
| C: `int getNumGeysers()` | Returns number of active geysers |

| | |
|---|---|
| MATLAB: `result = getNumGeysers()` | |
| C: `void getGeyserLocations(int geyser[10][2])`<br><br>MATLAB: `geyser = getGeyserLocations()` | Returns a matrix with the 2D sample square grid coordinates (see section 2.1) of any active geysers. The location is set to [-100, -100] for inactive geysers. *Note: even in 3D, only the [X,Y] coordinates are used, since the geysers cover the full Z axis.* |
| C: `bool isGeyserHere (float pos[2])`<br><br>MATLAB: `result = isGeyserHerePos(pos)` | Returns true if there is a geyser in the sample square at location `pos[X,Y]` in meters (e.g. from ZR state). *Note: even in 3D, only the [X,Y] coordinates are used, since the geysers cover the full Z axis.* |
| C: `bool isGeyserHere (int square[2])`<br><br>MATLAB: `result = isGeyserHereSquare(square)` | Returns true if there is a geyser in the sample square at location `square[X,Y]` (see section 2.1 for grid coordinates). *Note: even in 3D, only the [X,Y] coordinates are used, since the geysers cover the full Z axis.* |
| C: `float getScore()`<br><br>MATLAB: `myScore = getScore()` | Returns player's score |
| C: `float getOtherScore()`<br><br>MATLAB: `otherScore = getOtherScore()` | Returns opponent's score |
| C: `float getFuelRemaining()`<br><br>MATLAB: `fuel = getFuelRemaining()` | Returns remaining fuel as a fraction of total fuel allowed (1.00 = full tank; 0.50 = 50% remaining; 0.00 = empty tank). |
| C: `void sendMessage(unsigned char inputMsg)`<br><br>MATLAB: `sendMessage(inputMsg)` | Sends `inputMsg` to other satellite |
| C: `unsigned char receiveMessage()`<br><br>MATLAB: `msg = receiveMessage()` | Returns the most recent message sent by other satellite |


# 4 The Leaderboard

## 4.1 Introduction

The Zero Robotics Leaderboard has adapted over the years, and is now based on the Whole History Rating (WHR) system. The WHR approach tracks ratings throughout each phase of the competition in order to rank players on the leaderboard. Rating is calculated based on the probability of your team beating another team during a match (probability of wins and losses). Scores from individual matches are not factored into the calculation. See the “Calculating Ratings” section below for details.

The Leaderboard calculates ratings daily from the beginning of a competition until the submission deadline. All matches during the competition period count toward the rating in the competition. At the end of each competition phase, the final standings on the Leaderboard will determine which teams advance to the next phase.

## 4.2 Playing Matches

Each day, at 21:59:59 UTC, (except on competition deadlines where posted times apply) your most recently submitted code is collected by an automated system and played against other teams submissions, as described in the section titled “Standard play” below. The “Standard Play” cycle is repeated a total of five (5) times using the team placement from the previous cycle as the starting point for the next cycle. Results are posted on the website after all five (5) cycles have been completed. While the daily competition is ongoing the leaderboard will not be visible and clicking on the Leaderboard will return the message: “Results will be posted once the daily leaderboard run is complete”. Multiple “Standard play” cycles are completed for the following reasons: 1) To settle or converge results daily. (This ensures a more accurate reflection of rank for teams entering a competition leaderboard for the first time.) 2) To provide more match data for teams at the top and bottom of the leaderboard since these teams play fewer matches overall.

## 4.3 Standard play

As a default you will play the 9 teams above you and the 9 teams below you. If there are less than 9 teams above you, you will play all teams above you plus the 9 below you. If there are less than 9 teams below you, you will play all the teams below you and the 9 teams above you . Teams are randomly assigned as Blue or Red SPHERES during each match.


## 4.4 Initial submission rule

When your team first submits code to the competition you will play matches against the bottom quarter of the teams already on the leaderboard. On the first day with submissions, those teams will play as per “Standard Play”.

## 4.5 Last day of the competition

On the last day of the competition, the leaderboard will be run multiple times to converge/ stabilize the results: to calculate entry position of teams submitting for the first time and to assess final placement of all teams.

## 4.6 Calculating Ratings

A team’s standing in a competition is determined by a value called rating. The Leaderboard tracks all matches a team has played against other players in the course of the competition and creates a rating based on the outcomes.

Your rating is the factor R0 in the equation for the probability of your team beating another payer of rating R1 in a single match.

$$\text{Probability}(W) = e^{R0} / (e^{R0} + e^{R1})$$

The variable R0 is computed using an algorithm based on Bayesian inference. The probability of a rank r given a series of game outcomes G is then:

$$P(r|G) = P(G|r)P(r) / P(G)$$

is the prior distribution of ratings, which is assumed to vary in a random walk process with a standard deviation over each day of the competition of 0.1, chosen after testing for stability of the board. This allows all previous ratings to change, as implied by the name Whole History Rating. Ignoring P(G) because it is a normalizing constant with no effect of the final rating r, and incorporating the standard deviation of ratings, having each day of the competition representing k:

$$\prod P(r|G) = P(g_k|r_k)P(r_k|r_{k-1})$$

The final term P(r 0) is set to 0.

The algorithm uses newton’s method to maximize this probability as a function of your rating. The rating at this maximum is your new rating.


## 4.7 Summary

Not all wins are equal. Wins and losses are valued by their relation to the projected win probability. Your rating corresponds to a 50% win probability, meaning you are more than 50% likely to win against teams with lower ratings than yours, but not against teams with higher scores. Unexpected match results, either favorable or unfavorable, will leave the most noticeable impacts on your rating.

Although this process is based on probability, it is wholly reliable. The rank computation algorithm corrects errors in rank history throughout the competition, and optimization and stabilization safeguards have been added to calculate rank with extreme precision. Also, the leaderboard is designed to allow data to propagate throughout the system. If two players don't run against each other because their rank difference is more than 10 slots, overlapping matches update both teams' ratings. Every team's rating is relative to all other ratings.

To summarize-- there are several factors that affect a team's rating:

- Match Outcomes: A team that consistently wins matches will usually have a higher rating.
- Opponent Rank Winning against a higher ranked team will usually improve a team's rating.
- Other Match Outcomes The Leaderboard takes into account all matches played by all teams. Even if two teams do not have a direct encounter, their match outcomes will have an effect as they filter through third parties.

The team with the highest rating will be rank 1 on the leaderboard. The best way to stabilize and even improve your rank is the most logical way: keep working at your algorithms. There are no surefire alternatives. Also, don't let fear of a bad match ruining your team's prospects keep you from submitting early. Even though every submission is factored into your match history, results of past competitions demonstrate that teams that make many submissions tend to perform better.

# 5 Tournament Structure

## 5.4 Simulation Competition

All teams will participate in the simulation competition.

When the semifinal competition starts, the game will be updated with new challenges and the corresponding TBA values will be announced. These new challenges are intended to be substantial enough to require participation of all teams in preparing competition submissions.

## 5.5 Final Competition

The top 6 teams on the leaderboard at the end of semifinal play will advance to the Finals Competition (**NOT aboard ISS, but simulated**).

The Finals will take place during an event (location TBD) where all teams are invited to participate.

### 5.5.2 Finals Format

The teams will be divided into 2 brackets for the Final Competition. All teams ranked with odd numbers will participate in bracket A; all teams ranked with even numbers will participate in bracket B, as shown in this figure.

Division of Teams between brackets

| Bracket A teams | Bracket B teams |
|---|---|
| 1,3,5 | 2,4,6 |

Each bracket will play 3 matches in round--robin style: team A vs. B, B vs. C, and C vs. A.

After the round--robins are complete, there will be a winner of each bracket (shown as BR1, BR2 in the Competition Bracket figure.) The following rules determine the winner:

1. The team with the most wins advances
2. If teams are tied for wins, the team with the highest total score advances
3. If scores are tied, one more competition is run between the tied teams

The winning team from each bracket will play a single match to determine the Zero Robotics Italian Champion. The losing team will be awarded 2nd place.
Third and fourth place will be awarded after a single match between the 2$^{nd}$ teams of each bracket.

# 6 Season Rules

## 6.1 Tournament Rules

All participants in the Zero Robotics High School Tournament 2017 must abide by these tournament rules:

- The Zero Robotics team (MIT / Aurora/ ILC) can use/reproduce/publish any submitted code.
- In the event of a contradiction between the intent of the game and the behavior of the game, MIT will clarify the rule and change the manual or code accordingly to keep the intent.
- Teams are expected to report all bugs as soon as they are found.
- A “bug” is defined as a contradiction between the intent of the game and behavior of the game.
    - The intent of the game shall override the behavior of any bugs up to code freeze.
    - Teams should report bugs through the online support tools. ZR reserves the right to post any bug reports to the public forums (If necessary, ZR will work with the submitting team to ensure that no team strategies are revealed).
- Code and manual freeze will be in effect 3 days before the submission deadline of a competition.
- Within the code freeze period the code shall override all other materials, including the manual and intent.
- There will be no bug fixes during the code freeze period. All bug fixes must take place before the code freeze or after the competition.
- Game challenge additions and announcement of TBA values in the game manual may be based on lessons learned from earlier parts of the tournament.

## 6.2 Ethics Code

- The ZR team will work diligently upon report of any unethical situation, on a case by case basis.
- Teams are strongly encouraged to report bugs as soon as they are found; intentional abuse of an unreported bug may be considered as unethical behavior.
- Teams shall not intentionally manipulate the scoring methods to change rankings.
- Teams shall not attempt to gain access to restricted ZR information.
- We encourage the use of public forums and allow the use of private methods for communication.
- Vulgar or offensive language, harassment of other users, and intentional annoyances are not permitted on the Zero Robotics website.
- Code submitted to a competition must be written only by students.


- Players may not access the implementation instance of the game or modify any variables of the object. In particular, the api and game objects should not be duplicated or modified in any capacity.
- Simulation requests may only be done manually via the website interface, API calls for simulation are not allowed (even if doable).

# 7 Revision History

| Revision | Date | Changes Made | By |
|---|---|---|---|
| Italian-1.0 | 2018-01-31 | Initial release for Italian game. | LR |