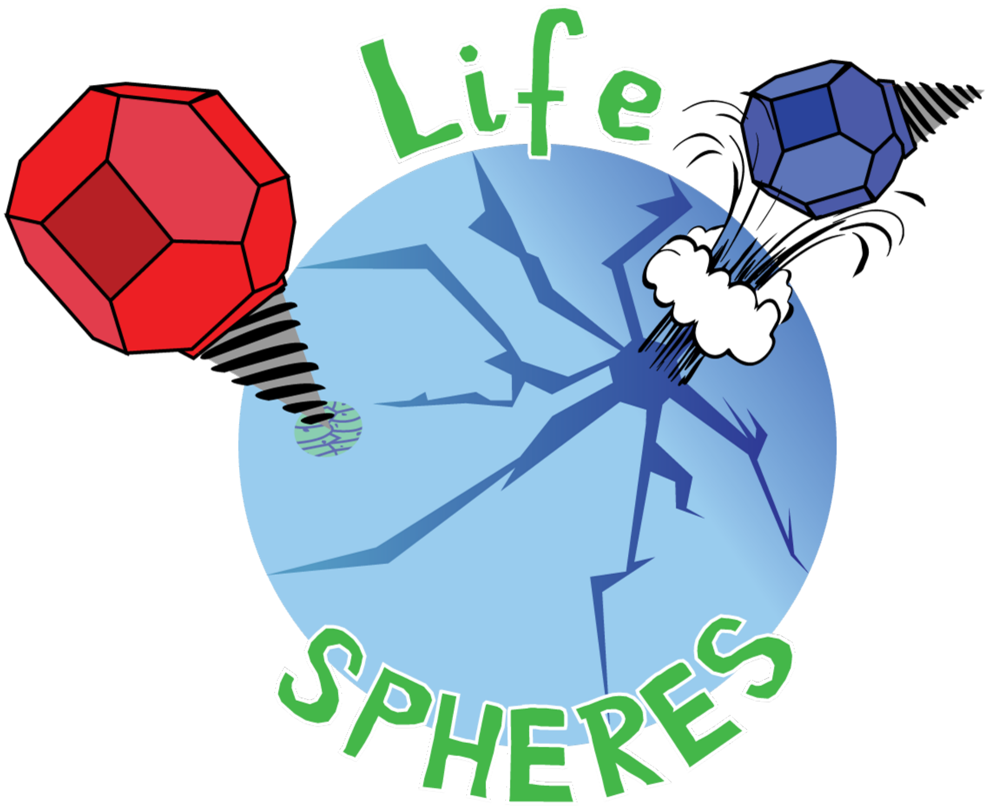
ZERO ROBOTICS

HIGH SCHOOL 2017

GAME MANUAL

Version: 3D-1.2



Living-microorganisms Investigation of Enceladus SPHERES Program

NOTE: Changes from the 2D to the 3D version of the game are HIGHLIGHTED in MAGENTA.

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 are 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.

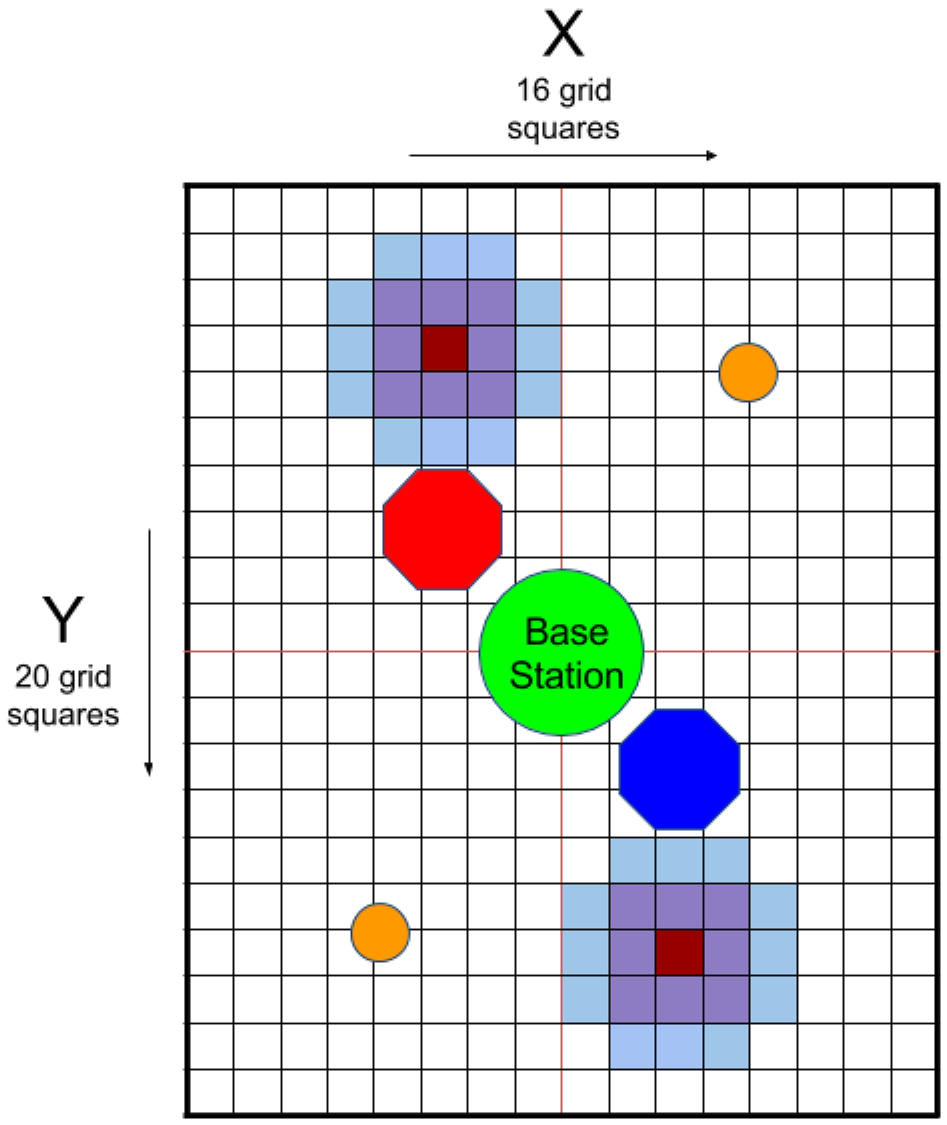
Good luck to all participating space engineers.

Alvar Saenz-Otero
SPHERES Lead

Table of Contents

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 180° along the tank axis. Once a sample is picked up, it may be analyzed to determine the concentration in that sample square. Successfully picking up a sample grants some points. There are two analysis options: (1) wait until delivery to the Base Station or (2) pick-up a portable *analyzer* and analyze in-situ immediately after pick-up. 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 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).

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.

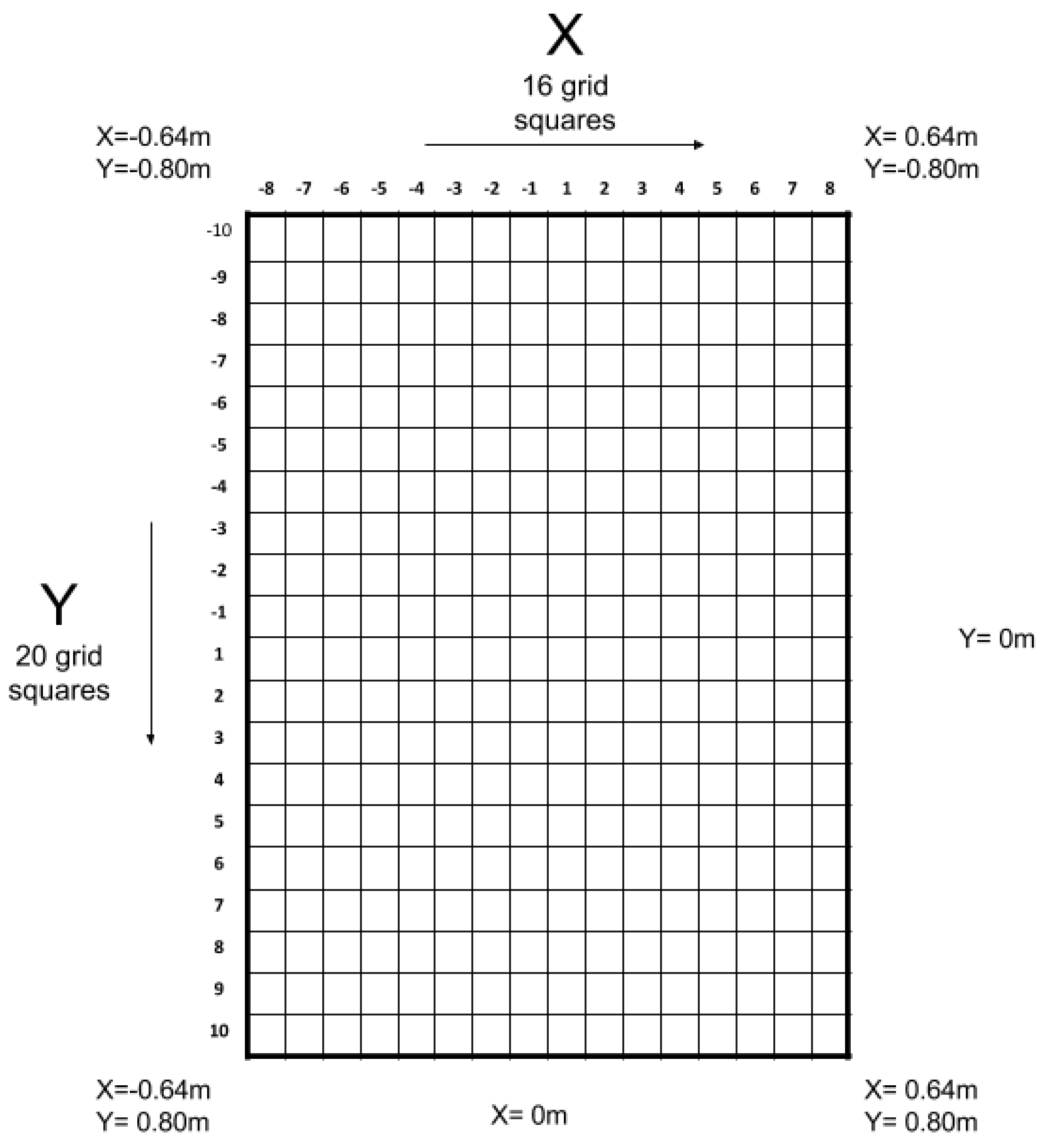
The Interaction Zone for the game has the following dimensions:

Interaction Zone Dimensions

	2D	3D	Alliance
X [m]	[-0.64 : +0.64]	[-0.64 : +0.64]	[-0.64 : +0.64]
Y [m]	[-0.80 : +0.80]	[-0.80 : +0.80]	[-0.80 : +0.80]
Z [m]	n/a	[-0.64 : +0.64]	[-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 squares (or cubes for 3D play), 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

	2D	3D	Alliance
Blue			
X [m]	0.2	0.2	TBA
Y [m]	0.2	0.2	TBA
Z [m]	0.0	0.0	TBA
Red			
X [m]	-0.2	-0.2	TBA
Y [m]	-0.2	-0.2	TBA
Z [m]	0.0	0.0	TBA

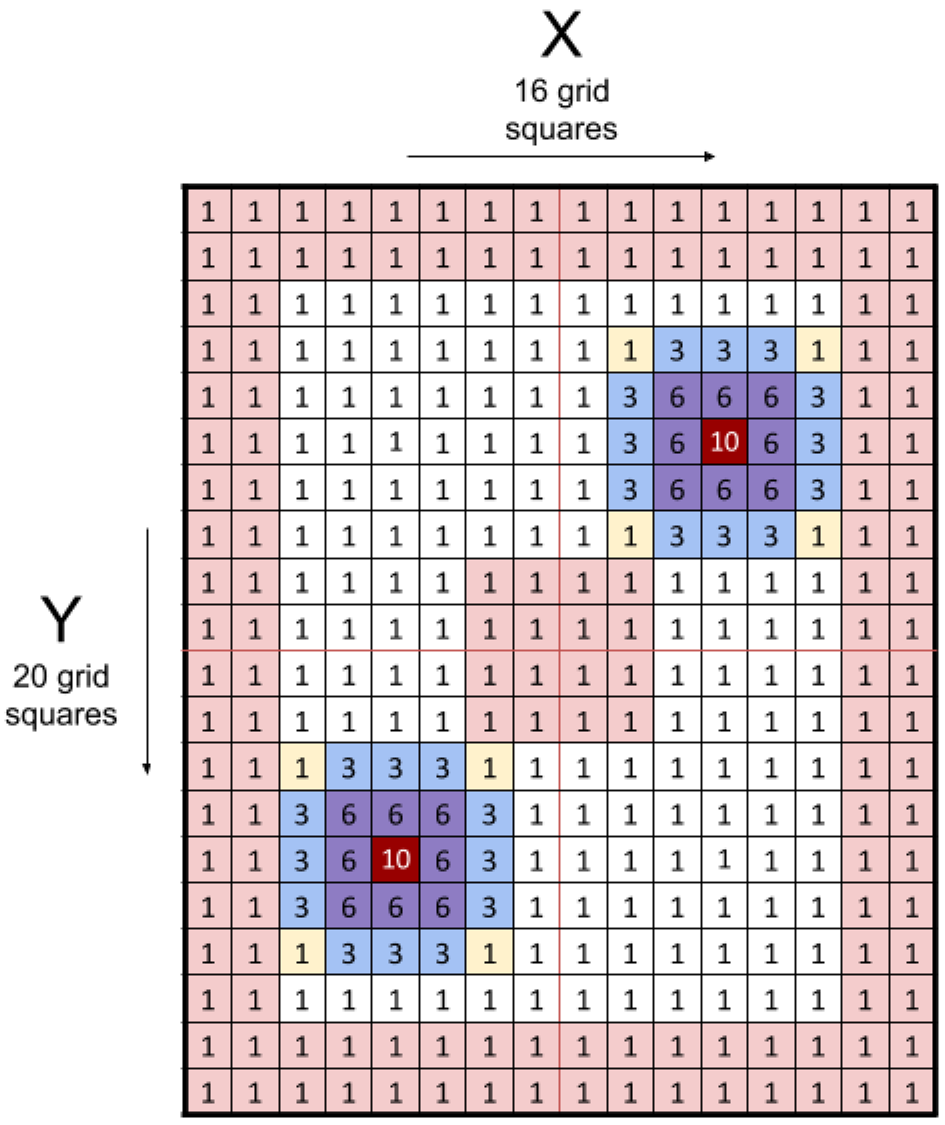
The satellite radius is 0.11m.

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.

The microorganisms are located at the “surface” of the moon, which is at Z=0.48m.

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



1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	3	3	3	1	1	1
1	1	1	1	1	1	1	1	1	3	6	6	6	3	1	1
1	1	1	1	1	1	1	1	1	3	6	10	6	3	1	1
1	1	1	1	1	1	1	1	1	3	6	6	6	3	1	1
1	1	1	1	1	1	1	1	1	1	3	3	3	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	3	3	3	1	1	1	1	1	1	1	1	1	1
1	1	3	6	6	6	3	1	1	1	1	1	1	1	1	1
1	1	3	6	10	6	3	1	1	1	1	1	1	1	1	1
1	1	3	6	6	6	3	1	1	1	1	1	1	1	1	1
1	1	1	3	3	3	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

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.

2.3 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 `getConcentrations` immediately
    b. 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. *(Reminder: the SPHERES satellite radius is 0.11m, see Section 2.1.1.)* The following figure illustrates the required start drill conditions:

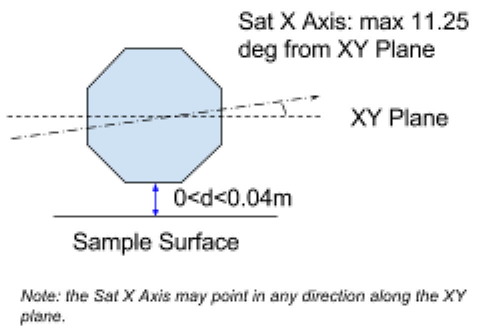

Sat X Axis: max 11.25 deg from XY Plane
XY Plane
0<d<0.04m
Sample Surface
Note: the Sat X Axis may point in any direction along the XY plane.

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 about the tank (Z) axis 90° while remaining at the same sample square where the drill was started, with a speed of under 0.01m/s, and maintaining the X Axis of the satellite aligned within 11.25° of the XY plane .

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.

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

	2D	**3D**	**Alliance**

Analyzer 0	X = 0.30m
	Y = -0.48m
		X = 0.30m
		Y = -0.48m
		Z = -0.16m
			X = 0.30m
			Y = -0.48m
			Z = -0.36m

Analyzer 1	X = -0.30m
	Y = 0.48m
		X = -0.30m
		Y = 0.48m
		Z = -0.16m
			X = -0.30m
			Y = 0.48m
			Z = -0.36m

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 analyzers*, once the satellite has picked up an analyzer, it will be able to conduct 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 function `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.

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

Teams lose points by:

- Drilling incorrectly
- Causing a geyser too close to the Base Station

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
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 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).

2.5.4 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.

2.6 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.

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.

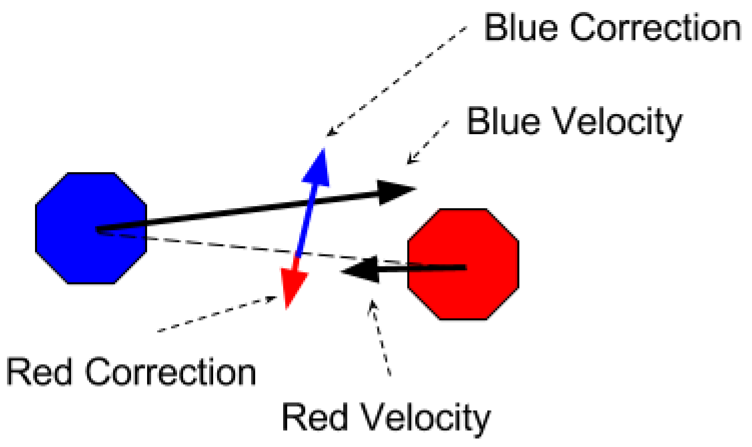
Any action that requires firing the thrusters (rotating, accelerating, decelerating), whether it was commanded by the player, due to activate collision avoidance or out-of-bounds breaking, or other penalties of the game play, will consume virtual fuel allocation.

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 MATLAB functions, the actual calling syntax is the one shown in the Table below.

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])` 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])` MATLAB: `pos = square2pos(square)`	Returns the position in meters of the center of the given sample `square` (see Section 2.1).
C: `int hasAnalyzer()` MATLAB: `result = hasAnalyzer()`	Returns the number of Sample Analyzers held by the SPHERES: 0 = no analyzers 1 = has only analyzer 0 2 = has only analyzer 1 3 = has both analyzers 0 and 1
C: `void getAnalyzer(bool pickedUp[2])` MATLAB: `pickedUp = getAnalyzer()`	Returns the status of the two analyzers in the boolean array `pickedUp`: true if the analyzer has been picked up by either sat false if the analyzer has not been picked up
C: `int getDrills(float pos[2])` 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])` 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()` MATLAB: `result = startDrill()`	Starts drilling the square under the current position of satellite. Returns true if started successfully false if drilling incorrectly, point penalty may result
C: `bool checkSample()`	Returns true if the sample currently being drilled is ready for pick-up. Returns false if the drill was not started or the drilling motion requirements are not complete.

MATLAB: `result = checkSample()`	
C: `int pickupSample()` MATLAB: `result = pickupSample()`	Attempts to collect sample. Must be run while drill is running. Returns index into sample array (sample #) if collected -1 if sample space is full -2 for incorrect drilling procedure / drill is off
C: `void getSamplesHeld(bool samples[5])` MATLAB: `samples = getSamplesHeld()`	Populates a list of booleans indicating whether a sample space has been filled or is empty: Returns true if a sample space is full (picked-up OK) false if a sample space is empty
C: `int getNumSamplesHeld()` MATLAB: `result = getNumSamplesHeld()`	Returns the total number of samples held (0 = none, up to 5).
C: `void getConcentrations(float concentrations[5])` MATLAB: `concentrations = getConcentrations()`	Returns the list of concentrations of each of the sample spaces that have been analyzed: Returns concentration of sample -1 if the sample has not been analyzed -2 if the sample is empty
C: `float dropSample(int sampleNumber)` 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. Returns concentration if dropped correctly at base 0.0 if not dropped correctly or sample empty
C: `bool atBaseStation()` 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()` MATLAB: `stopDrill()`	Stops drilling and resets any drill errors if they occurred.
C: `bool getDrillEnabled()` MATLAB: `result = getDrillEnabled()`	Returns true if the drill is currently enabled, false if it is disabled.
C: `bool getDrillError()` 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()` MATLAB: `result = getNumGeysers()`	Returns number of active geysers
C: `void getGeyserLocations(int geyser[10][2])` 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])` 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])` 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()` MATLAB: `myScore = getScore()`	Returns player's score

C: `float getOtherScore()` MATLAB: `otherScore = getOtherScore()`	Returns opponent's score
C: `float getFuelRemaining()` 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)` MATLAB: `sendMessage(inputMsg)`	Sends `inputMsg` to other satellite
C: `unsigned char receiveMessage()` 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.

Finally, it is important to note that after every competition phase, (2D, 3D, Alliance phase) all ranking data is refreshed and teams start from scratch.

5 Tournament Structure

5.1 2D Practice Simulation Competition

All teams that complete a valid registration are eligible to participate in the 2D practice simulation competition.

Note: Several International teams from non-ISS nations may participate in the 2D and 3D phases of the tournament, however, they will not continue into the alliance phase. These teams will participate by invitation only.

5.2 3D Simulation Competition

All USA, Russian, and Australian teams that complete a valid registration and submit code that ***achieves a nonzero score when competed against a "blank player"*** (a player with no code) by the 2D practice competition deadline are eligible to participate in the 3D simulation competition. However, only the 25 highest scoring teams from each ESA member state will continue into the 3D phase.

When the 3D competition starts the game will be updated with new challenges and the corresponding TBA values will be announced.

5.3 Alliance Formation Event

The top ranked 84 teams on the leaderboard at the end of 3D simulation play will form 28 alliances of three (3) teams each. The 28 alliances will work cooperatively to complete the semifinals and, if not eliminated, the finals. Teams ranked below rank 84 will be invited to participate in the Virtual Finals. See "Virtual Finals Simulation Competition" section below.

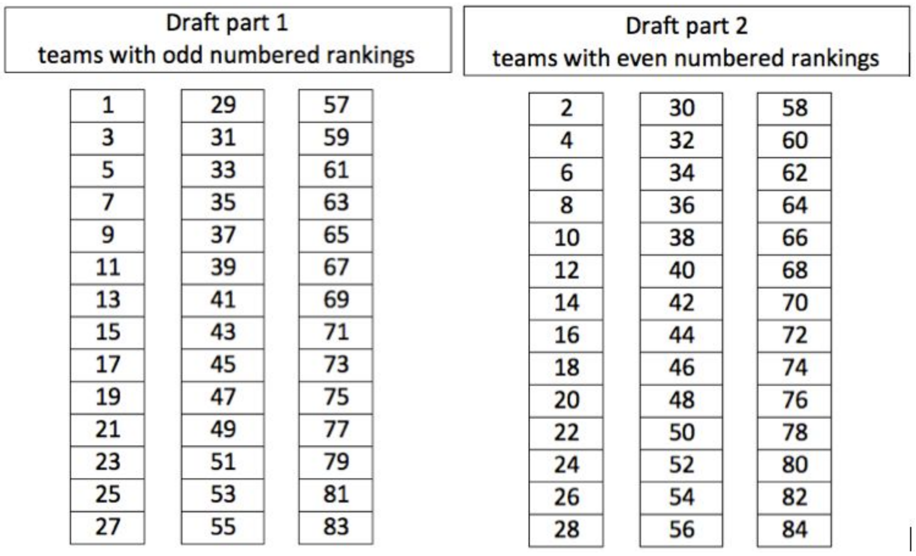
The ranking of the teams will be determined by the rankings at the close of the 3D leaderboard.

After the 3D leaderboard rankings are announced, advancing teams will have three days to contact each other to discuss their alliance preferences. Any teams that do not wish to continue in the Tournament will have the opportunity to cede their position to the next ranked team.

The Alliance Formation Event will take place on November 4th. The alliance formation event or "draft day" is an online event during which teams will follow the alliance selection process described below to invite other teams into alliance and/or accept their place within an alliance. At least one representative from each team must participate in the online "draft day" event for the team to continue to the alliance phase. Additional details about the online “draft day” and how to participate will be provided in advance of the event.

Due to the large number of teams involved, the draft will be split into two parts as shown in the below figure. All teams ranked with odd numbers will participate in Draft part 1 and all teams ranked with even numbers will participate in Draft part 2. Each draft will include 42 teams.

Division of Teams for the Drafts



Draft part 1 teams with odd numbered rankings			Draft part 2 teams with even numbered rankings		
1	29	57	2	30	58
3	31	59	4	32	60
5	33	61	6	34	62
7	35	63	8	36	64
9	37	65	10	38	66
11	39	67	12	40	68
13	41	69	14	42	70
15	43	71	16	44	72
17	45	73	18	46	74
19	47	75	20	48	76
21	49	77	22	50	78
23	51	79	24	52	80
25	53	81	26	54	82
27	55	83	28	56	84

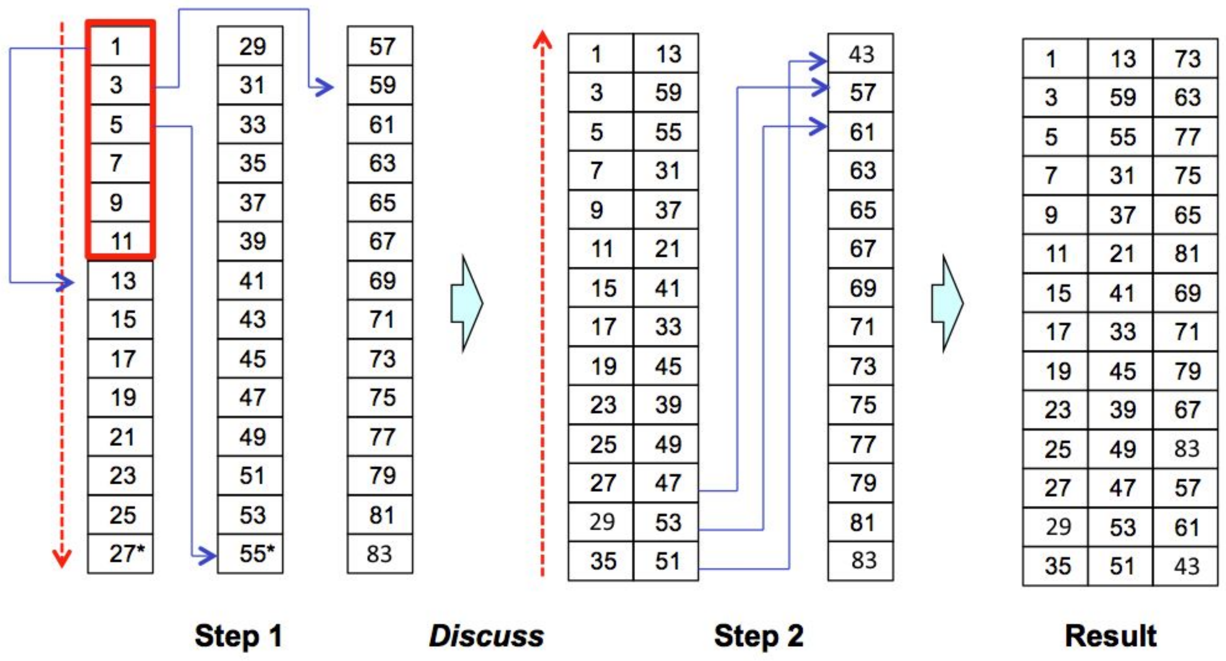
Note: The times of the draft parts are To Be Announced

The alliance selection process will follow a serpentine pattern (illustrated in the Figure below):

- The highest ranked team selects their partner from any except the top 6 ranked teams in their draft.
- The next highest ranked team selects their partner from any of the remaining teams except the top 6 ranked teams in their draft
- 14 pairs are created in this manner
- A break takes place for the new pairs to discuss their selection for the 3rd alliance team
- The “lowest” ranked pair then selects their 3rd team from the remaining 14 teams
- The “2nd lowest” rank pair make the next selection
- Continue until all 14 alliances are formed.
- This process is duplicated in both draft events to end with a total of 28 alliances

Rule: No more than 2 of the teams in an alliance can be from the same continent.

Alliance Creation Process demonstrated for teams with odd number rankings



1	29	57
3	31	59
5	33	61
7	35	63
9	37	65
11	39	67
13	41	69
15	43	71
17	45	73
19	47	75
21	49	77
23	51	79
25	53	81
27*	55*	83

1	13	43
3	59	57
5	55	61
7	31	63
9	37	65
11	21	67
15	41	69
17	33	71
19	45	73
23	39	75
25	49	77
27	47	79
29	53	81
35	51	83

1	13	73
3	59	63
5	55	77
7	31	75
9	37	65
11	21	81
15	41	69
17	33	71
19	45	79
23	39	67
25	49	83
27	47	57
29	53	61
35	51	43

(*The example shown above is approximate.)

5.4 Semifinal Simulation Competition

The 28 alliances created during the Alliance Formation Event will participate in the semifinal 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 alliance teams in preparing competition submissions.

5.5 ISS Final Competition

The top 14 alliances on the leaderboard at the end of semifinal play will advance to the ISS Finals Competition. Alliances ranked below rank 14 will be invited to participate in the Virtual Finals. See “Virtual Finals Simulation Competition” section below for details about the Virtual Finals.

The ISS finals will take place aboard the International Space Station with live transmission to MIT. Teams will be invited to live broadcast events at MIT (US), an ESA location (EU), and at University of Sydney (Australia).

5.5.1 Overview and Objectives

Running a live competition with robots in space presents a number of real world challenges that factor into the rules of the competition. Among many items, the satellites use battery packs and CO2 tanks that can be exhausted in the middle of a match, and the competition must fit in the allocated time. This section establishes several guidelines the Zero Robotics team intends to follow during the competition. Keep in mind that as in any refereed competition, additional real time judgments may be required. Please respect these decisions and consider them final.

Above all, the final competition is a demonstration of all the hard work teams have put forward to make it to the ISS. The ZR staff's highest priority will be making sure every alliance has a chance to run on the satellites. It is also expected that the competition will have several "Loss of Signal" (LOS) periods where the live feed will be unavailable. We will attempt to make sure all teams get to see a live match of their player, but finishing the competition will take priority.

To summarize, time priority will be allocated to:

1. Running all submissions aboard the ISS at least once
2. Completing the tournament bracket
3. Running all submissions during live video

We hope to complete the tournament using only results from matches run aboard the ISS, but situations may arise that will force us to rely on other measures such as simulated matches.

5.5.2 Competition Format

The alliances will be divided into 2 conferences for the ISS competition. All teams ranked with odd numbers will participate in Conference A; all teams ranked with even numbers will participate in Conference B, as shown in this figure.

Division of Teams between Conferences

Conference A Alliance ranks	Conference B Alliance ranks
1,3,5,7,9,11,13	2,4,6,8,10,12,14

Each conference will include one "bye" team (alliances ranked #1 and #2 automatically advance to the conference semifinals) and 2 brackets of 3 alliances each. Each bracket will play 3 matches in round-robin style: alliance 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 ISS Competition Bracket figure.) The following rules determine the winner:

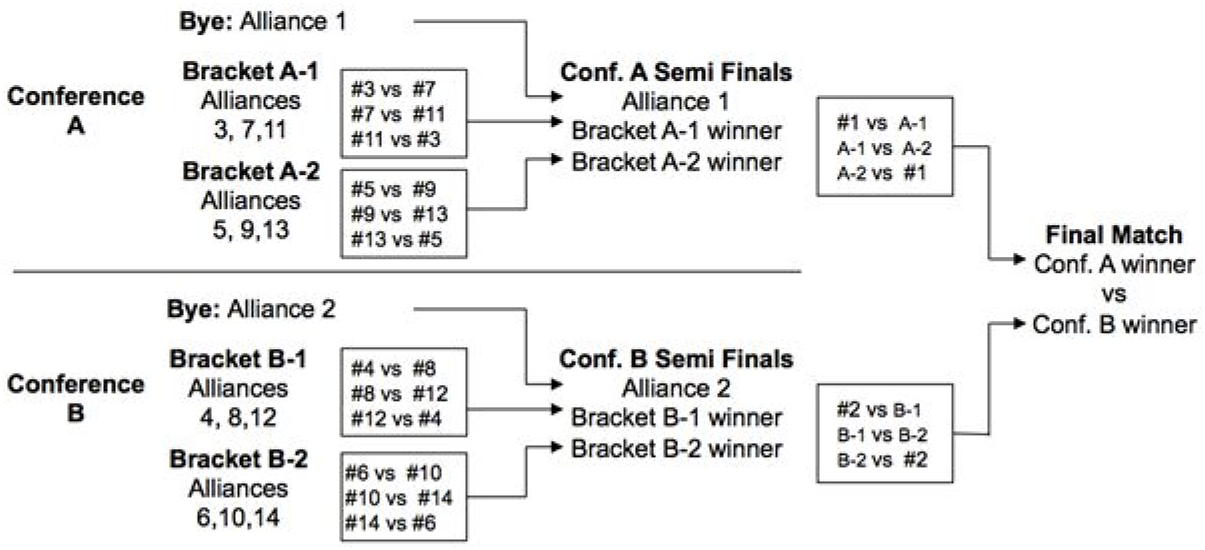
1. The alliance with the most wins advances
2. If alliances are tied for wins, the alliance with the highest total score advances
3. If scores are tied, simulation results will be used to break the tie

The semifinal match between the top 2 bracket winners and the “bye” team will also be played in round-robin style. The winner of this match is determined in the same way as the bracket winners:

1. The top 2 alliances with the most wins in their bracket, advance
2. If there is a tie for wins, the alliance(s) with the highest total score in their bracket advance
3. If scores are tied, simulation results will be used to break the tie

The winning alliance from each conference will play a single match to determine the Zero Robotics ISS Champion. The losing alliance will be awarded 2nd place.

ISS Competition Bracket



5.5.2.1 Definition: Successful Match

- Both satellites move correctly to initial positions
- Both satellites have normal motion throughout the test
- Both satellites return a valid score
- Neither satellite expends its CO_2 tank during a test run

5.5.2.2 Definition: Simulated Match

In advance of the competition, the ZR Team will run a simulated round robin competition between all participating teams. The results from matches in this competition will be used in place of ISS tests if necessary (see below.) The results of a simulated match will only be announced if they are used in the live competition.

5.5.2.3 Scoring Matches

Scores in the scoring matches will be determined according to these rules:

Case 1: Successful Match, Both Satellites Return Unique Score (e.g. 130, 151)
- The scores will be recorded as the official score for the match.

Case 2: Either Satellite Returns an Invalid Score (e.g. 130, 255)
- If the first run of a match is not successful, the match will be rerun, time permitting.
- If the second run of a match is not successful, the results from a simulated match will be used.

5.6 Virtual Finals Simulation Competition

All teams participating in the 3D competition that do not advance to the Semi Final competition (Alliance Phase) and all teams participating in the Semi Final Competition that do not advance to ISS Finals will be invited to participate in the Virtual Finals.

The Virtual Finals game will be identical to the Semi Final competition (Alliance Phase) game. Teams participating in the Virtual Finals will submit to a Virtual Finals Leaderboard.

Teams may choose to participate in the Virtual Finals as individual teams or as alliances. Teams that did not participate in the original alliance formation event are welcome to create alliances, if desired. However, once a team/alliance submits code to the Virtual Finals Leaderboard the team/alliance composition cannot be changed.

Once the Virtual Finals Leaderboard closes, the top ranked 2 teams/alliances will advance to the Championship Match of the Virtual Finals. Time permitting, the final Championship Match of the Virtual Finals competition will be competed aboard ISS during the ISS Finals.

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
2D-1.0	2017-09-09	Initial release for 2D game.	ASO
3D-0.2	2017-09-18	Corrected Peak Zone in Page 4 example.	ASO
3D-0.3	2017-09-27	Corrected description of peak concentration keep out zones	ASO
3D-0.4	2017-09-28	Added section about Collision Avoidance	ASO
3D-1.0	2017-09-29	Release of 3D game manual.	ASO
3D-1.1	2017-10-13	Modifications for game balancing are highlighted and include: increase sample number from 3 to 5, increase base station radius to .24m, modify Z location of planet surface, and analyzers.	WRF, MCA
3D-1.2	2017-10-20	Corrects error as highlighted in game manual API.	WRF