ZERO ROBOTICS
MS SUMMER 2017

Game Manual
Ver. 1.0 DRAFT

SPACE SPHERES

SPACE-S (Satellite Positioning and Constructing Entities - SPHERES)
TO: SPACE-SHERES Teams

RE: ATTENTION! CALLING TEAMS TO BUILD SATELLITES!

All engineers in Satellite Positioning and Constructing Entities division,

The Earth Governing System urgently needs space engineers to facilitate
relocation of humans to Mars.

As you are well aware, the Earth is becoming uninhabitable and we have to
relocate to Mars permanently. For a smooth transportation to Mars, we must first scout
out the best locations for residences and other buildings. Fastest way of identifying
these locations is to build surveying satellites to orbit Mars.

The Earth Governing System already launched the necessary satellite pieces into
the space. Engineering teams are asked to construct these pieces into satellites in
“ideal zones”, identified based on current solar and meteorite activity. However, the
coordinates of ideal zones are unknown until teams place their three Satellite
Positioning System devices and get a reading. Once the SPS are placed, teams can
assemble their satellite pieces at their zones.

Good luck to all participating space engineers.

On behalf of the Earth Governing System,

Alvar Saenz-Otero

====================================================================================================

RE: RE: ATTENTION! CALLING TEAMS TO BUILD SATELLITES!

New information!

We are informed that the space engineers from our rival company SPACE-Y are also
trying to build satellites to go to Mars. The team which will be able to construct
their satellites fastest will be made a contract. But beware, SPACE-Y might want some
of the satellite pieces that you’ve already collected for themselves. We wish all
engineering teams luck in this competition.

May the best team win.

On behalf of the Earth Governing System,

Alvar Saenz-Otero
# Table of Contents

1. Game Overview  
   1.1 Overview of Game Features  
      1.1.1 Satellite Positioning Systems (SPSs)  
      1.1.2 Assembly Zones  
      1.1.3 Satellite Piece Items  
      1.1.4 Locations of Items and SPACE-S  
      1.1.5 Docking  
   1.2 Game Layout  
      1.2.1 Interaction Zone Dimensions  
   1.3 Satellite  
      1.3.2 Fuel  
      1.3.3 Inter-satellite Communications  
      1.3.4 Code Size  
      1.3.5 Initial Position  
      1.3.6 Player ID  
      1.3.7 Noise  
   1.4 Gameplay  
      1.4.1 Satellite Positioning Systems  
      1.4.2 Docking  
      1.4.3 Items  
      1.4.3.1 Item Types and Locations  
      1.4.3.2 Point Values  
      1.4.3.3 Other Item Details  
      1.4.4 Scoring Summary  
   1.5 Game Modes  
   1.6 Game Setup  
   2. Game Play  
   3. Game Rules  
   4. Game Objectives  
   5. Game Scoring  
   6. Game胜利条件  
   7. Game Challenges  
   8. Game Tips  
   9. Game FAQ  
   10. Game License  
   11. Game Credits  
   12. Game Contact
1.4.5 Penalties
   Table 13: Penalties
1.4.6 End of game

2. Tournament
   Table: Tournament Key Dates 2017
2.1 Regional Simulation Competition
   2.1.1 Competition Periods
   2.1.2 Submitting Code
   2.1.3 Competition Format – Regional Competition
2.2 Collaboration for ISS Finals
2.3 ISS Final Competition
   2.3.1 Overview and Objectives
   2.3.2 Competition Format
   Figure 7: ISS Competition Bracket
   2.3.3 Scoring Matches

3. Season Rules
   3.1 Tournament Rules
   3.2 Ethics Code

4. ZR User API

Revision History
1. Game Overview

Matches of SPACE-SPHERES will be played between two SPHERES, controlled by programs written by two separate teams. Each team will compete to have the most points when the round time is up. Each round lasts 180 seconds. Points may be generated by collecting the items (representing the satellite pieces) spread across the playing field and dropping them off inside one’s assembly zone, representing the ideal place to construct a satellite.
1.1 Overview of Game Features

This game features Satellite Positioning Systems (SPSs), Assembly Zones, and Satellite Piece Items.

1.1.1 Satellite Positioning Systems (SPSs)

At the start of the game, there are three Satellite Positioning Systems (SPSs) given to each SPHERES. Teams need to place these three SPSs to form a triangle in order to learn the location of their assembly zone. If the first and second SPS are too close together you will not be able to drop your 2nd SPS. Also the area of the triangle formed must be at least \(XX\) m\(^2\) in order to get a good estimated location. The SPHERES will not be able to drop its final SPS unless the area of triangle formed by the 3 dropped SPSs is larger than \(XX\) m\(^2\). Read more about SPSs in section 1.4.1 and assembly zones in section 1.1.2.

1.1.2 Assembly Zones

The assembly zone is a spherical location which does not interfere with existing satellite orbits. The location is chosen randomly from a specific range of coordinates. It is about the size of one SPACE-S, with a diameter of 0.2 meters. Once all three SPSs are placed, the player will be able to use the function

```
bool game.getZone(float zoneInfo[3])
```

...to learn the zone location. Assembly Zone locations change each time you play the game.
1.1.3 Satellite Piece Items

Once the SPACE-S knows where to assemble the final satellite, it may begin moving satellite piece items to its own specified assembly zone. Satellite items come in three sizes: small, medium, and large. Teams accumulate points for every second that an item is left in their zone. In order for a SPACE-S to pick up an item, it must dock to the item (docking information is in section 1.4.2). The items are distributed symmetrically on the map with the reflection point for each one being about the origin. This way, each item and its reflected pair are the same distance away from each SPACE-S.

1.1.4 Locations of Items and SPACE-S

The locations of satellite items, and the opponent SPACE-S are available to the teams. Teams enter the
specified item ID’s (described in section 1.4.3) and use the function

```c
void getItemLoc(float pos[], int itemID)
```

to get the location of the item. A SPACE-S can move anywhere on the map and have fixed initial positions (described in section 1.3.5).

1.1.5 Docking

To pick up a satellite piece, a SPACE-S needs to approach the item, slow down to below .01m/s, be pointing at the item and calling the docking function. The SPACE-S needs to stop in front of the item at a certain distance to pick up and the item will be released at the same distance away (described in section 1.4.2).

In addition a SPACE-S needs to approach the “correct” side of the item, marked with a different color than rest of the item (described in section 1.4.2).

1.2 Game Layout

The Zero Robotics Middle School Summer Program 2017 will be conducted in simulation. The game is played in an area called the Interaction Zone. If players leave the Interaction Zone, they will be considered out of bounds. The location of the SPHERES is measured from the center of the satellite.

The Interaction Zone for the game has the following dimensions:

<table>
<thead>
<tr>
<th>Table 1: Interaction Zone Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MS 2D</strong></td>
</tr>
<tr>
<td>X [m]</td>
</tr>
<tr>
<td>Y [m]</td>
</tr>
<tr>
<td>Z[m]</td>
</tr>
</tbody>
</table>
The game arena is a plane with only X and Y dimensions (Z=0). The initial item locations will be at fixed points which create a symmetric map. There are a total of six items: two small, two medium and two large. There is more information on items in section 1.4.3.

Penalties are applied for a) colliding with items, and b) for staying in the opponent’s assembly zone for longer than 10 cumulative seconds. See section 1.4.5 for more information about penalties.

All Satellite Items can only be docked from one side. See details in section 1.4.2

1.3 Satellite

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 satellite, like any other spacecraft, has a fuel source (in this case liquid carbon dioxide) and a power source (in this case AA battery packs). These resources are limited and must be used wisely. Therefore, the players of Zero Robotics are limited in the use of real fuel and batteries by virtual limits within the game. This section describes the limits to which players must adhere to wisely use real SPHERES resources.
1.3.2 Fuel

Each player is assigned a virtual fuel allocation of 60 seconds, which is the total sum of fuel used in seconds of individual thruster firing. Once the allocation is consumed, the satellite will not be able to respond to 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 such as rotating, accelerating or moving consumes fuel.

<table>
<thead>
<tr>
<th>Table 3: Fuel Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuel Allocation [s]</td>
</tr>
</tbody>
</table>

1.3.3 Inter-satellite Communications

This section is not used for the Middle School game.

1.3.4 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 a code size estimate, 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.

1.3.5 Initial Position

The Blue Sphere starts at the X, Y, Z of [0.0, 0.15, 0.0]. The Red Sphere starts at the X, Y, Z of [0.0, -0.15, 0.0].

The satellite radius is 0.11m, but satellite position relative to game features is determined by the location of the center of the satellite.

<table>
<thead>
<tr>
<th>Table 5: Initial Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
</tr>
<tr>
<td>X [m]</td>
</tr>
<tr>
<td>Y [m]</td>
</tr>
<tr>
<td>Z[m]</td>
</tr>
<tr>
<td>Red</td>
</tr>
<tr>
<td>X [m]</td>
</tr>
<tr>
<td>Y [m]</td>
</tr>
<tr>
<td>Z[m]</td>
</tr>
</tbody>
</table>
1.3.6 Player ID

Users will identify themselves as “playerID = 1” and opponents as “playerID = 2” for all games, whether or not they are the red SPHERES or the blue one respectively.

1.3.7 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 two satellites may be affected by random perturbations in different ways, resulting in small or even large variations in score. This is fully intended as part of the challenge and reflects uncertainties in the satellite dynamic and sensing models. The best performing solutions will be those that prove to be robust to these variations and a wide variety of object parameters.

1.4 Gameplay

In order to be victorious over the opposing team, each satellite should place their three SPSs, dock satellite items to place in their zones, all while managing their fuel, their time, and their location on the gameplay area.

1.4.1 Satellite Positioning Systems

There are 3 SPSs held by each SPACE-S at the start of all the games. Like GPS, the SPSs function based on the principle of trilateration. SPACE-S teams must place the SPSs in a triangle to get information about the location of their assembly zone’s center. The SPSs must form a triangle with an area of at least XXX. SPSs are deployed using the void dropSPS() function. If the first SPS is too close to the 2nd SPS you will not be able to drop it. SPSs must be at least XXm apart. If the area of the triangle is too small, the final SPS will not drop when void dropSPS() function is called. Upon successfully dropping your final SPS you will receive points equal to XX*(SPS Area).

SPSs cannot be placed outside game boundaries.

1.4.2 Docking

To dock with an item a SPACE-S:

1. Must approach an item’s docking face
   a. The satellite items can be docked from one side only, from the “pointing face” of the item. This face is marked with a different color than the rest of the item. The SPACE-S needs to point towards the center of the item’s docking/pointing face with a tolerance of 0.25 radians (The angle between the satellite’s facing vector and the vector between the center of the satellite and the center of any face of the item is within 0.25 radians). SPACE-S teams need to check if their attitudes face the attitudes of items.
Boo isFacingCorrectItemSide(int itemID) returns true if the SPACE-S is facing the correct side of the given item. (ItemID’s can be found in section 1.4.3).

2. Must not be moving faster than .01m/s
3. Must be within a specified distance of an item. (see Table 7: Item Docking Distances)

4. Must call the bool dockItem(int itemID) function to dock. bool dockItem(int item ID) returns true when docking is successful.

The SPHERES docks with its negative x (-x) face.

Figure 4: SPHERES points and docks with its -x face

<table>
<thead>
<tr>
<th>Item Type</th>
<th>Docking distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>Between 0.124 and 0.146 meters from the center</td>
</tr>
<tr>
<td>Medium</td>
<td>Between 0.138 and 0.160 meters from the center</td>
</tr>
<tr>
<td>Large</td>
<td>Between 0.151 and 0.173 meters from the center</td>
</tr>
</tbody>
</table>
1.4.3 Items

1.4.3.1 Item Types and Locations

There are six Satellite items (of three sizes) scattered around the interaction zone. Each has a unique numeric identifier from 0 to N-1, where N is the number of items. In the middle school game the initial position and the attitude of items 0-5 are provided in Table 8.

<table>
<thead>
<tr>
<th>Item Number</th>
<th>Item Type</th>
<th>Location (X, Y, Z)</th>
<th>Attitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Large</td>
<td>(.23, .23, 0)</td>
<td>(-1,0,0)</td>
</tr>
<tr>
<td>1</td>
<td>Large</td>
<td>(-.23, -.23, 0)</td>
<td>(+1,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>Medium</td>
<td>(.36, -.36, 0)</td>
<td>(+1,0,0)</td>
</tr>
<tr>
<td>3</td>
<td>Medium</td>
<td>(-.36, .36, 0)</td>
<td>(-1,0,0)</td>
</tr>
<tr>
<td>4</td>
<td>Small</td>
<td>(-.50, .50, 0)</td>
<td>(0,+1,0)</td>
</tr>
<tr>
<td>5</td>
<td>Small</td>
<td>(.50, -.50, 0)</td>
<td>(0,-1,0)</td>
</tr>
</tbody>
</table>
1.4.3.2 Point Values

In this game satellite items are of varying sizes. The larger objects are worth more points.

The satellite items come with three sizes of which there are two each.

- Small items:
  - Point Value: If collected in the zone, it gives 0.1pts per second.
  - Size: The Small item has ⅛ size of SPHERES

- Medium items:
  - Point Value: When in the zone it gives 0.15 pts per second.
  - Size: The Medium item has ⅝ size.

- Large items:
  - Point Value: It gives 0.2 pts per second in the zone.
  - Size: The large item has ⅞ size of SPHERES.

1.4.3.3 Other Item Details

Call the game function  \texttt{int hasItem(int itemID)} to determine whether the item is held by nobody, you, or your opponent.

A SPACE-S will be penalized if it collides with an item or with another SPACE-S. The penalty will be determined by the SPACE-S’ speed when it collides with the item. This penalty applies everywhere except inside a SPACE-S team’s own Assembly Zone where collisions with items will not be penalized. See section 1.4.5 for more information about penalties.

The diagram below shows how the items are placed symmetrically, at the same distance from the diagonal.
1.4.4 Scoring Summary

Your score is largely based on the items you have in your Assembly Zone. Items are worth different point values depending on their sizes.

The scoring calculation is as follows:

<table>
<thead>
<tr>
<th>Table 12 : Point Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Point Values</strong></td>
</tr>
<tr>
<td>Small Item [pts/sec]</td>
</tr>
<tr>
<td>Medium Item [pts/sec]</td>
</tr>
<tr>
<td>Large Item [pts/sec]</td>
</tr>
<tr>
<td>Dropping final SPS</td>
</tr>
</tbody>
</table>

1.4.5 Penalties

Your score may also change due to various penalties that will be enforced as you play the game.
The penalties will be enforced as follows:

<table>
<thead>
<tr>
<th>Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collisions with items or SPHERES*</td>
</tr>
<tr>
<td>(-5.0 x your SPHERES speed.)</td>
</tr>
<tr>
<td>Remaining in opponent’s zone</td>
</tr>
<tr>
<td>-0.4/sec (after cumulative 10 sec)</td>
</tr>
</tbody>
</table>

*There is no penalty applied for collisions with items located inside the player’s own Assembly Zone.

1.4.6 End of game

The game ends after 180 seconds. Whichever team has more points wins. In the unlikely case of a tie, whoever is closer to the center/origin of the playing field at game end wins.

2. Tournament

A Zero Robotics tournament consists of several phases called competitions. The following table lists the key deadlines for the 2017 tournament season:

<table>
<thead>
<tr>
<th>Session 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Some variations may apply. ZR schedule reflects partial scheduling for 4th of July week. All programs in a state must follow the same schedule. Please check with your state coordinator to confirm your schedule and submission dates.</td>
</tr>
<tr>
<td>June 12 (Mon)</td>
</tr>
<tr>
<td>Start of Session 1</td>
</tr>
<tr>
<td>Week 1</td>
</tr>
<tr>
<td>June 19 (Mon) or June 20 (Tues)</td>
</tr>
<tr>
<td>Field Day</td>
</tr>
<tr>
<td>Week 2</td>
</tr>
<tr>
<td>June 30 (Fri), 5:00 pm local time</td>
</tr>
<tr>
<td>Practice Code Deadline</td>
</tr>
<tr>
<td>Week 3</td>
</tr>
<tr>
<td>Fourth of July week</td>
</tr>
<tr>
<td>If operating – Regional Code Deadline due July 7th, 5:00 pm local time</td>
</tr>
<tr>
<td>Week 4/4.5</td>
</tr>
<tr>
<td>July 12 (Wed), 5:00 pm, local time</td>
</tr>
<tr>
<td>Alternate Regional Code Deadline</td>
</tr>
<tr>
<td>July 14 (Fri), 5:00 pm, local time</td>
</tr>
<tr>
<td>ISS Code Deadline</td>
</tr>
<tr>
<td>Week 5/5.5</td>
</tr>
<tr>
<td>July 19 (Wed), 5:00 pm local time</td>
</tr>
<tr>
<td>Alternate ISS Code Deadline</td>
</tr>
<tr>
<td>Mid-Aug</td>
</tr>
<tr>
<td>ISS Finals</td>
</tr>
</tbody>
</table>

| Session 2                                                                 |

Table: Tournament Key Dates 2017
• Some variations may apply. ZR schedule reflects partial scheduling for 4th of July week. All programs in a state must follow the same schedule. Please check with your state coordinator to confirm your schedule and submission dates.

<table>
<thead>
<tr>
<th>Week of June 26th</th>
<th>Alternate Start of Session 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>July 3 (Mon) -Fourth of July Week</td>
<td>Start of Session 1</td>
</tr>
<tr>
<td>July 10 (Mon) or July 11 (Tues)</td>
<td>Field Day</td>
</tr>
<tr>
<td>July 21 (Fri), 5:00 pm local time</td>
<td>Practice Code Deadline</td>
</tr>
<tr>
<td>July 28 (Fri), 5:00 pm, local time</td>
<td>Regional Code Deadline</td>
</tr>
<tr>
<td>Aug 3 (Thu), 5:00 pm, local time</td>
<td>ISS Code Deadline</td>
</tr>
<tr>
<td>Mid-Aug</td>
<td>ISS Finals</td>
</tr>
</tbody>
</table>

2.1 Regional Simulation Competition

2.1.1 Competition Periods

The program starts with two phases of regional simulation competition:

• **Practice Regional Competition** At the end of Week 3 of the summer program, teams will submit their code and a competition will be run. The results of this competition are not official and are intended to guide teams in improving their code during Week 4. The submission deadline is 5 PM local time on the Thursday/Friday of Week 3. (See Tournament Key Dates Table for date specific to your session.)

• **Regional Competition** At the end of Week 4 of the summer program, teams will submit their updated code and a competition will be run. The results of this competition determine the regional 1st, 2nd, and 3rd place champion. The submission deadline is 5 PM local time on the Thursday/Friday of Week 4. (See Tournament Key Dates Table for date specific to your session.)

2.1.2 Submitting Code

To enter a program in a competition the team must use the Submit tool located under the Simulate menu on the IDE page of the Zero Robotics website. You may change your submission as many times as you like before the submission deadline, but only the last program that has been submitted before the deadline will be used. No programs submitted after the deadline will be accepted unless the Zero Robotics staff determines that emergency circumstances made timely submission impossible.

2.1.3 Competition Format – Regional Competition

The regional competition will be a round robin, with every team playing every other at least once. Each team will play as close as possible to half its matches with each satellite (blue and red). The team with the
most wins will be the champion. In the event of a tie, the team that won the most head-to-head matches against the other tied team(s) will be the champion. If this procedure fails to resolve a tie, the tied team with the highest total score (that is, the scores from all of its matches added together) will be the champion. The results of regional competitions will be released by 8 AM ET on the Monday after the competition submission deadline. The Zero Robotics team will release them earlier if possible. All regional results may not be released simultaneously.

2.2 Collaboration for ISS Finals

During the first several days of week 5 of the summer program all teams in each region will have an opportunity to collaborate to try to improve their 1st place regional winner’s code prior to ISS submittal deadline. Teams from the same region are encouraged to try to beat the regional winner’s code and then share their solution with the regional winner. The regional winner will submit the final code from their region for the ISS Competition. The submission deadline is 5pm on the Thursday of Week 5.

2.3 ISS Final Competition

The final code submitted by the regional winner from each region will compete in the ISS finals. The finals will take place aboard the International Space Station with live video transmission. All teams will be invited to watch the live broadcast.

2.3.1 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 team 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.
2.3.2 Competition Format

A total of 13 teams will compete on ISS during the Middle School ISS Final Competition this year. The thirteen teams will be divided into 2 conferences for the ISS competition (as shown in the Figure below).

Each conference will include 2 brackets of 3 teams each plus Conference B will have one team which will move directly into the semi-final round. For 2017, this extra team will be the Russian team. 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 A1, A2, B1, B2 in the figure below.) 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

Semi-final matches between the bracket winners in each conference will determine the conference winners.

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.

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 CO₂ tank during a test run

**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.

**2.3.3 Scoring Matches**
If the match is successful, the scores will be recorded as the official score for the match. If the first run of a match is not successful, the match will be re-run, time permitting. If the second run of a match is not successful, the results from a simulated match will be used.

**3. Season Rules**

**3.1 Tournament Rules**
All participants in the Zero Robotics Middle School Summer Program 2017 must abide by these tournament rules:
- The Zero Robotics team (MIT / ILC / Aurora) 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.

**3.2 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).

4. ZR User API

The following reference table explains how to use common API and game functions for the SPACE-S MS 2D game.

**SPHERES Controls API Functions** These functions used to control a SPHERES satellite in Zero Robotics. These functions do not change from game to game.
Note for teams using the text editor: All SPHERES control functions except DEBUG are accessed as members of the api object. In order to use these functions, use the syntax api.function(arguments). For example:

```
api.setPositionTarget(mypos); //instructs the SPHERE to move to mypos
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOTE: This function is only available for use with the graphical editor</td>
<td>Moves the player’s satellite to the given x, y, and z coordinates.</td>
</tr>
<tr>
<td>void setPositionTarget(float posTarget[3])</td>
<td>Moves the player’s satellite to a point of your choice. You can select a point by creating a three element array, where each element represents an x, y, or z coordinate.</td>
</tr>
<tr>
<td>void setAttitudeTarget(float attTarget[3])</td>
<td>Rotates the player’s satellite to face along the x, y, or z axis. You can select the direction by creating a three element array, where each element represents the x, y, or z unit vector of the direction you want to face. For more information, see the More Simple Arrays and setAttitudeTarget Function tutorial on the ZeroRobotics website.</td>
</tr>
</tbody>
</table>
### void getMyZRState(float myState[12])
Retrieves the state of your SPHERE (location, velocity, attitude, and angular velocity). The state will be stored in a twelve element array that you create beforehand. After calling this function, the first three elements of your array will hold the x, y, and z coordinate of your SPHERE’s location; the next three elements will hold the x, y, and z components of the velocity; the next three elements will hold the x, y, and z components of the attitude vector; and the final three elements will hold the x, y, and z components of the angular velocity.

### void getOtherZRState(float otherState[12])
Same as getMyZRState but gets the state of the opponent’s satellite.

### unsigned int getTime()
Returns the time (in seconds) elapsed since the beginning of the game.

### DEBUG("Some text!"))
Prints the supplied text to the console. If you are coding in the text editor, do not type api. before this function and make sure to use double parentheses.

*DEBUG is found in the Debug section, not SPHERES Controls, and it is not an API function

### Game Specific Functions - SPACE-S MS 2D
The functions in the table below are specific to the game SPACE-S MS 2D
Note for teams using the text editor: All game functions are accessed as members of the game object. In order to use these functions, use the syntax game.function(arguments). For example:
```
game.dropSPS(); //instructs the game to drop an SPS
```

### SPACE-S MS - SPS/Zone

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void dropSPS()</td>
<td>Drops SPSs.</td>
</tr>
<tr>
<td>bool dropSPS()</td>
<td>Returns true if the player has dropped an SPS.</td>
</tr>
<tr>
<td>bool getZone(float zoneInfo[3])</td>
<td>If all sps have been placed, returns true and stores zone location in zoneInfo. Otherwise returns false. Can only be called once per second.</td>
</tr>
</tbody>
</table>
SPaCE-SPHERES Game Manual

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int getNumSPSHeId()</td>
<td>Returns the number of SPSs still held by the sphere</td>
</tr>
</tbody>
</table>

**SPaCE-S MS - Dock/Drop**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool isFacingCorrectItemSide(int itemID)</td>
<td>Returns true if the player is facing the correct side of the item for docking.</td>
</tr>
<tr>
<td>bool dockItem(int itemID)</td>
<td>Returns true if player picks up specific item successfully</td>
</tr>
<tr>
<td>void dropItem()</td>
<td>Drops the item that the sphere is currently holding</td>
</tr>
</tbody>
</table>

**SpySPHERES MS - Items**

*NOTE: Item Id is the number assigned to a particular Item (see Game Manual for details)*

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void getItemLoc(float pos[3], int itemID)</td>
<td>Copies the location of a given item into the given array. Stores the location of an Item with the id number id in a three element array of your choice. After calling this function, each index in your array will hold the x, y, or z coordinate of the Item’s location.</td>
</tr>
<tr>
<td>bool itemInZone(int itemID)</td>
<td>Returns whether the specified item is placed in your zone. Returns false if the item is being held or is not in the zone.</td>
</tr>
<tr>
<td>int hasItem(int itemID)</td>
<td>Tells who has a given item. @param itemID The integer identifier of a given item. return 0 if no one has picked up the specified item, 1 if you have picked up item, or 2 if your opponent has picked up the item. The graphical editor uses a custom version of this function shown in the next row.</td>
</tr>
<tr>
<td></td>
<td>Checks who (me/other/no-one) has a specified item (items 0-8). Returns true if</td>
</tr>
</tbody>
</table>

*MIT, ILAB, Aurora, CASIS, ESA, NASA*
me/other/no-one has picked up the specified item. Returns false otherwise.

### SpySPHERES MS - Other

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>float getScore()</td>
<td>Returns player’s current score if “my” is selected.</td>
</tr>
<tr>
<td>float getOtherScore()</td>
<td>Returns opponent’s current score if “other’s” (is selected.</td>
</tr>
<tr>
<td>float getFuelRemaining()</td>
<td>Tells the player how much fuel remains. Returns a float indicating how many seconds of fuel remain.</td>
</tr>
</tbody>
</table>
Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Changes Made</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0 DRAFT</td>
<td>05/12/17</td>
<td>Initial release</td>
</tr>
</tbody>
</table>