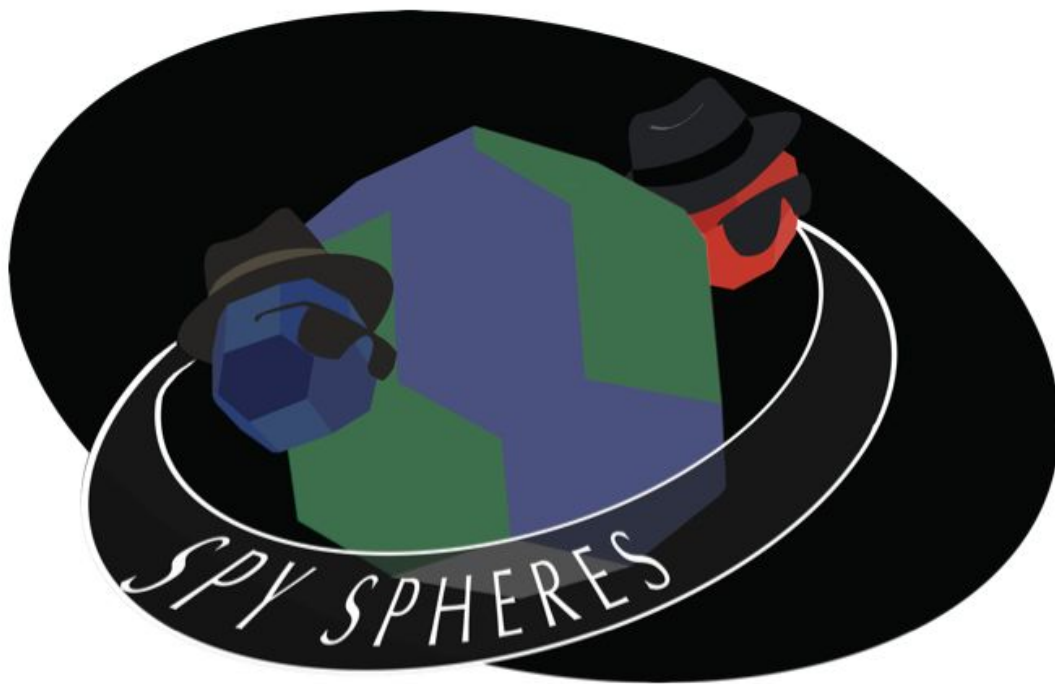


# ZERO ROBOTICS

---

MS SUMMER PROGRAM 2016

Zero Robotics Middle School Summer Program 2016



## **SpySPHERES Game Manual**

Ver.1.3

FWD: FWD: FWD: URGENT! ALL EMPLOYEES NEEDED!

Hello employees of BLU Industries, I have an interesting opportunity for all of you! One of NASA's old satellites has malfunctioned and spontaneously broken into pieces in low Earth orbit. It was holding very valuable data. NASA has stated that any company that wishes to try their luck at gathering this data is free to do so. This represents a huge opportunity for BLU!

In order to be the first company out there, we've outfitted our HYPER-SPHERE™ with the newest propulsion and control systems from R&D, products of trillion-dollar research. It is powered by solar energy, so it can recharge whenever it's under the sun.

Members of the BLU family, I'm calling on you today to develop an autonomous piloting system for the HYPER-SPHERE™, so that when it reaches the NASA satellite it will be able to collect as much data as possible as quickly as possible. While doing so, your pilot should also be wary of how much energy it has, recharging from the sun or from the strewn about batteries of the broken satellite as needed.

Thank you for your work,  
Alvar Saenz-Otero  
BLU CEO

=====

RE: FWD: FWD: FWD: URGENT! ALL EMPLOYEES NEEDED!

Hello again employees, the game has changed. I have just received news that RED corporation, led by their CEO Evil Alvar, have made their own plans to recover the orbiting data. Were it any other company, I would not be worried, as our HYPER-SPHERE™ would easily get there first. Unfortunately, however, RED have their own MEGA-SPHERE™, and I have a hunch that they too will be pulling out their newest technology for this venture. It looks as though we'll have some competition.

There is, however, a silver lining to this new development. If our R&D department can get their hands on information about RED's new transportation technology, especially pictures, it would be even more valuable to us than NASA's data. Thankfully, the HYPER-SPHERE™ is already equipped with a camera, although it isn't powerful enough to take pictures without light from the sun.

Therefore, your job has expanded. While collecting the debris is still important, your pilot should focus on gaining intel about our competitor's satellite. We also fear that RED will attempt to take pictures of us, so additionally work to prevent that as best you can.

Let's get that tech!  
Alvar Saenz-Otero  
BLU CEO



# Table of Contents

## [1. Game Overview](#)

[Figure: Game Overview](#)

### [1.1 Game Layout](#)

[Figure: Interaction Zones](#)

[Mechanic Summary Table](#)

### [1.2 Satellite](#)

#### [1.2.1 ZR User API](#)

#### [1.2.2 Time](#)

#### [1.2.3 Fuel](#)

[Table: Fuel Allocation](#)

#### [1.2.4 Code Size](#)

#### [1.2.5 Initial Position](#)

[Table: SPHERES Satellite Deployment Locations](#)

#### [1.2.6 Player ID](#)

#### [1.2.7 Noise](#)

### [1.3 Gameplay](#)

#### [1.3.1 Energy](#)

#### [1.3.2 Light and Dark Zones](#)

[Table: Light and Dark Zone Properties](#)

[Figure: Light & Dark Zones](#)

#### [1.3.3 Items:](#)

[Figure: Item Collection](#)

[Figure: 3D Item Locations & Types](#)

[Table: Item Locations](#)

#### [1.3.4 Picture Taking](#)

[Figure: Reasons to Fail a Picture](#)

[Figure: Conditions for a Successful Picture](#)

#### [1.3.5 Uploading Pictures](#)

#### [1.3.6 Scoring Summary](#)

#### [1.3.7 End of game](#)

## [2. Tournament](#)

[Table: Tournament Key Dates 2016](#)

### [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: ISS Competition Bracket](#)

[2.3.3 Scoring Matches](#)

[3. Season Rules](#)

[3.1 Tournament Rules](#)

[3.2 Ethics Code](#)

[4. ZR User API](#)

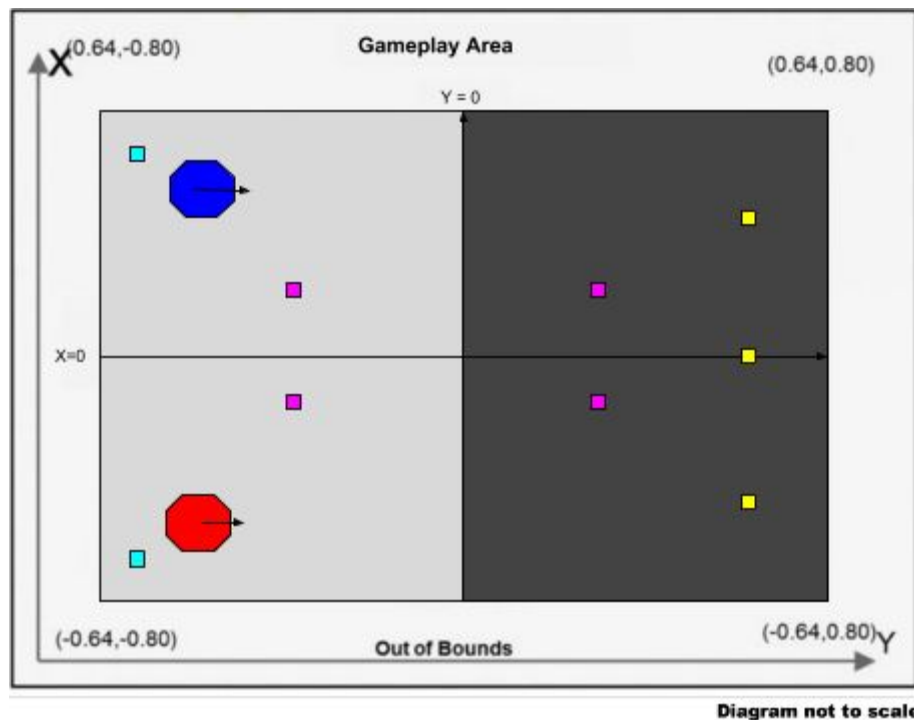
[Revision History](#)



# ZRMS 2016 Game Manual

## 1. Game Overview

Figure: Game Overview



Matches of SPY SPHERES will be played between two SPHERES satellites, controlled by programs written by two separate teams. Each team will compete to have the most points when the round time is over. Each round lasts 180 seconds. Points may be generated by taking pictures of the other satellite and uploading them, or by collecting one of the score-generating items (representing the pieces of the NASA satellite) spread across the playing field.

In this game, there are dynamic Light and Dark zones that have various impacts on the satellites. These represent how a satellite in space, in Low Earth Orbit, is half of the time illuminated by the sun and the other half in Earth's shadow (also called eclipse).

Additionally, real space satellites have small amounts of power available for all of their equipment. Therefore, in SpySPHERES, each satellite has a finite amount of energy for any game actions. Real satellites launch with batteries and use solar panels to replenish their



energy when exposed to direct sunlight. In SpySPHERES, energy can be generated by being in the Light Zone or by picking up energy items.

Pictures cost energy to take, and can only be successfully taken when the target is in the Light Zone.

The Light and Dark zones will change positions over the course of the round.

Lastly, there are mirror items which, when deployed, prevent the user from taking pictures but also reflect any pictures your opponent takes back at them, making them worthless.

## 1.1 Game Layout

The Zero Robotics Middle School Summer Program 2016 competition 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:

X: [-0.64: +0.64]

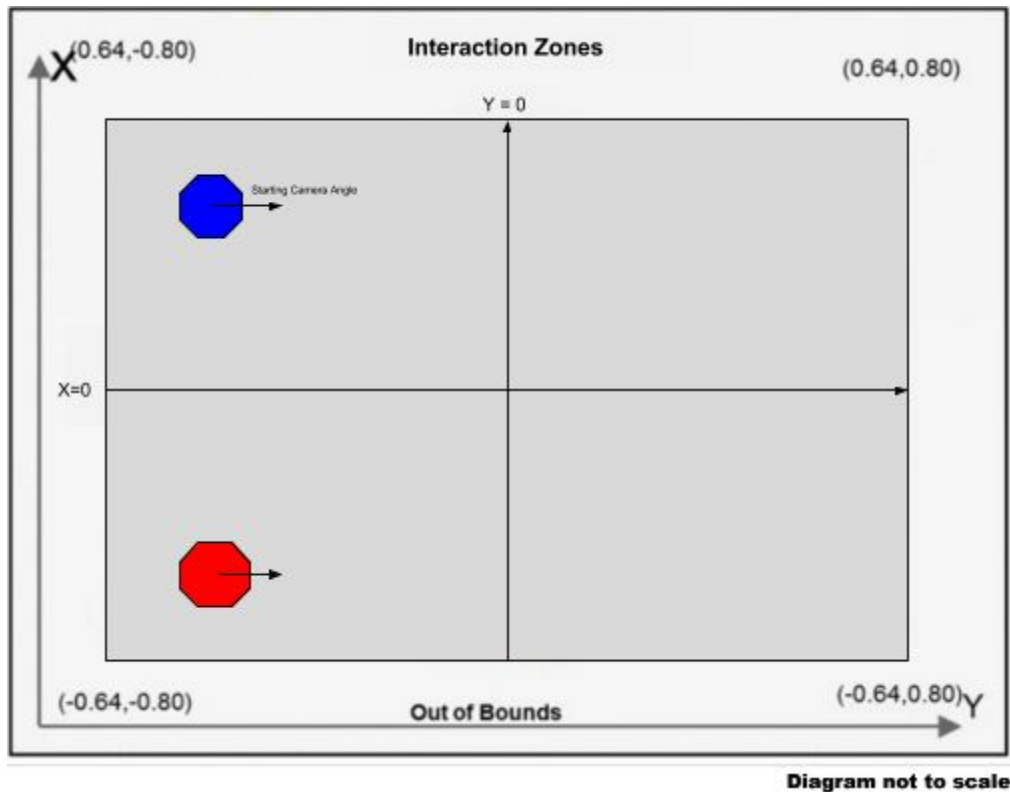
Y: [-0.8: +0.8]

Z: [-0.64: +0.64]

Satellites that go out of these limits will be stopped from moving further out of bounds.



Figure: Interaction Zones



The arena is a plane with only X and Y cardinal dimensions. The light and dark each take up either the negative or the positive half of the Y-axis, and switch positions. There is more info on the light and dark zones in 1.3.2. There is also more information about the three Score+ items, four Energy Packs and two Mirror types in 1.3.3.

## 1.2 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. (For the middle school game, the ability to move to a different Z-coordinate has been disabled.) 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.



### 1.2.1 MS ZR 2016 User API

Game specific functions, along with the standard ZR User API functions, are provided both in **Section 5** of this manual and at the following link:.

<http://zerorobotics.mit.edu/tournaments/22/info/116/0/>

### 1.2.2 Time

Each round lasts 180 seconds. After 180s scores will be final and compared.

### 1.2.3 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 or moving consumes fuel.

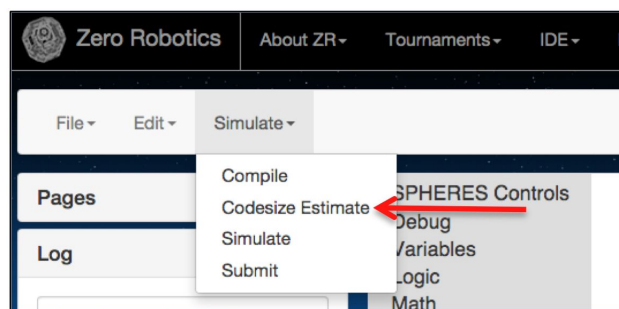
Table: Fuel Allocation

Fuel Allocation [s]	60s
---------------------	-----

### 1.2.4 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. To check your project's code size –open your project in the IDE then select “Code Size Estimate” under the simulation tab as shown in the figure below. The percent usage will be displayed in the Log.

Figure: How to Check Project Code Size





### 1.2.5 Initial Position

Teams should write code assuming that their player is the Blue SPHERE. It is not necessary for teams to account in their code for the possibility of being either red SPHERES or blue SPHERES. This adjustment will be made automatically.

The Blue Sphere starts at the X, Y, Z of [0.4,-0.6, 0.0].

The Red Sphere starts at the X, Y, Z of [-0.4,-0.6, 0.0].

Table: SPHERES Satellite Deployment Locations

Blue	
X [m]	0.4
Y[m]	-0.6
Z[m]	0.0
Red	
X [m]	-0.4
Y[m]	-0.6
Z[m]	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.

### 1.2.6 Player ID

Users will identify themselves as “playerID = 0” and opponents as “playerID = 1” for all games, whether or not they are the red SPHERES satellite or the blue one.

### 1.2.7 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.

## 1.3 Gameplay

In order to be victorious over the opposing team, each satellite should make use of the consumable items and their camera in order to take pictures of the opponent and gain points, all while managing energy, fuel and their position in light or darkness.



### 1.3.1 Energy

Energy is the most prohibitive resource the satellites utilize. Both players start with 5.0 energy at the start of the game, which is also the maximum energy a satellite can have. If the satellite is in the light zone, it gains 0.5 energy every second. To maneuver the satellite, 0.15 energy is used per 1 second of fuel used. Other activities that cost energy include taking pictures ( 1.0 energy when camera is on) and calling `float getPicPoints()` (0.1 energy when the camera is on) as described later.

- The satellites can check how much energy it has left by calling the game function

 `float getEnergy()`

- The satellite may also check the energy of the opponent by calling `float getOtherEnergy()`.

### 1.3.2 Light and Dark Zones

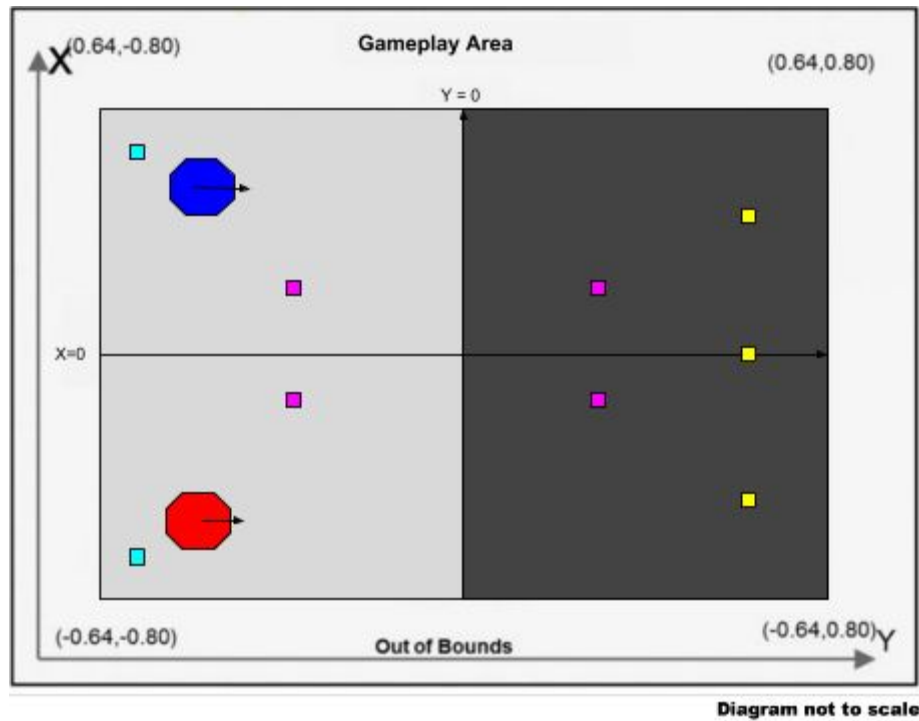
The Light and Dark Zones affect picture taking and energy recharge. Here is a table with their properties:

Table: Light and Dark Zone Properties

	Picture Can be Taken of Me	Does Energy Recharge
Light Zone	Yes	Yes
Dark Zone	No	No
Light and Dark Zones switch positions after 60 seconds and 150 seconds of game time		



Figure: Light &amp; Dark Zones



The positive Y sector starts out in the Dark Zone and the negative Y section starts out in the Light Zone. At 60 seconds and 150 seconds into the game, the Light Zone and the Dark Zone will switch position.

Light in all “Out of Bounds” areas are the same as the adjacent in-bounds areas. If a satellite goes “Out of Bounds” in the dark zone and runs out of energy, it will be stuck “Out of Bounds” until the light zone switches to the satellite’s side of the game grid. Once the satellite’s energy is recharged the satellite will be able to resume normal operations.


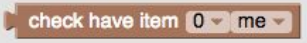

- The user can call the function `int getLightSwitchTime()` to determine how long, in seconds, until the zone switches.

### 1.3.3 Items:



There are three types of items scattered around the interaction zone: Energy Packs, Score+ Items, and Mirrors. Each has a unique numeric identifier from 0 to N-1, where N is the number of items.

There are nine total items (0-8), with all three item types. Each of them may only be used once, and they do not regenerate.



- You can find item locations using the function:  `void getItemLoc(float pos[3], int itemID)`
- Call the game function  `bool checkHaveItem(itemID)` to determine whether the item is held by nobody, you, or your opponent.
- Call the game function  `float[3] getItemLoc(int item_id)` to obtain the location of the item.

#### Item Types:

- Energy Pack - Upon pickup this item is instantly used. It refills the satellite to its maximum energy level, 5.0.
- Score+ - Once picked up this item is also used immediately. It adds 1.5 to the satellite's score.
- Mirror - Unlike the other two items, this is simply held on to once picked up. Once it is deployed, the holder has 24 seconds during which they cannot take pictures, but any pictures the opposing satellite takes of them will be worth no points in 2D. To clarify, the mirrors prevent pictures from being taken. Some related functions are:
  -  `void useMirror()` - Deploys a mirror.
  -  `int getNumMirrorsHeld()` - Returns the number of mirrors the user has.

#### Item Pick Up:

In order to collect an item, the center of the user's satellite must be no greater than 0.05 meters away from the item and it must be moving at slower than .01 meters per second.



Figure: Item Collection

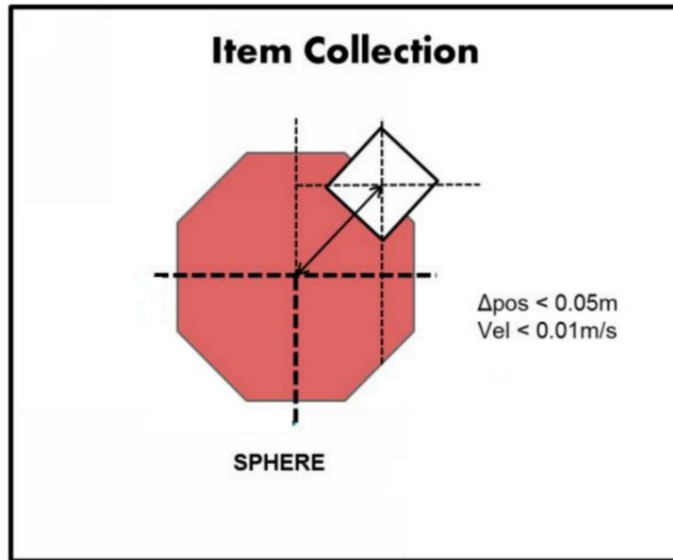


Figure: Item Locations & Types

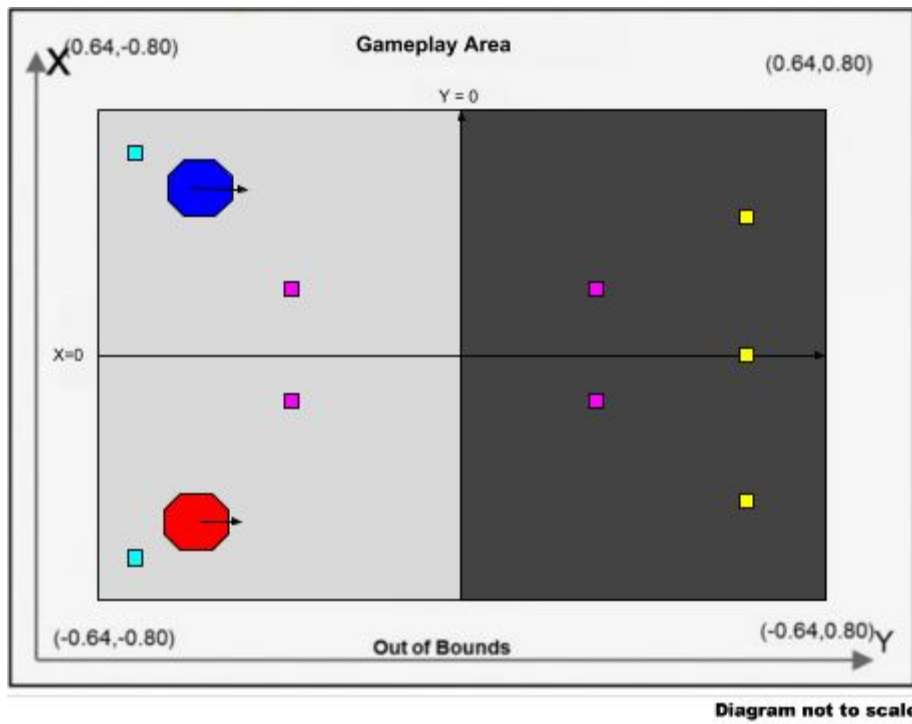


Table: Item Locations

ID #	Item	Location
0	Energy Pack	( 0.25, -0.4)
1	Energy Pack	(-0.25, -0.4)
2	Energy Pack	( 0.25, 0.4)

3	Energy Pack	(-0.25, 0.4)
4	Score+	( 0.0, 0.6)
5	Score+	( 0.4, 0.6)
6	Score+	(-0.4, 0.6)
7	Mirror	( 0.6,-0.7)
8	Mirror	(-0.6, -0.7)

### 1.3.4 Picture Taking

The way to score the largest possible amount of points is by taking pictures of the other satellite. There are multiple requirements for taking a picture:


- The user satellite must be facing the opposing satellite. (The angle between the satellite's facing vector and the vector between the positions of the two satellites is within 0.5 radians.)

- The attitude needed to point to the other satellite can be found by calling the

function:  `void getAttToOther(float AttToOther[3])`

- You can check using the function:  `bool isFacingOther()`

- The opposing satellite must not be in the Dark Zone (that is, it must be in the Light Zone).

- You can check using the function:  `bool checkInLight(int Player) me player=0; other player=1`

- The user satellite must have at minimum 1 energy.

- The user satellite's camera must be on.


- You can check using the function:  `bool isCameraOn()`

- The user must not have an active mirror item.

- You can check using the function:  `bool useMirror ()`

- The user's satellite must be at least 0.5m away from the opposing satellite.

- The user must call the function  `float takePic()`.

- The function  `float takePic()` will then take a picture if all of the conditions are met. Regardless of whether the picture was successfully taken, `takePic()` will consume 1 energy and shut down the camera for 3 seconds when it is called.

- The function `take pic` `float takePic()` will return 0 if picture-taking was unsuccessful, and the point value of the picture otherwise. Every picture, whether successful or not, adds 0.01 points to the user's score as a tiebreaker.

The amount of points each picture is worth is determined by the distance between the two satellites when the picture is taken. It follows this formula:

$$\text{points} = 2.1 + 0.1 / (\text{distance} - \text{PHOTO\_MIN\_DISTANCE} + 0.1)$$

Where PHOTO\_MIN\_DISTANCE is 0.5. If the opposing satellite is using an active mirror, the amount of points the picture is worth will be zero.

- The function `get pic points` `float getPicPoints()` is available to check the points a picture will be worth before taking it. Calling it costs 0.1 energy, but does not disable the camera. Note that this function can sense whether or not the opponent has deployed a mirror.

**Figure: Reasons to Fail a Picture**

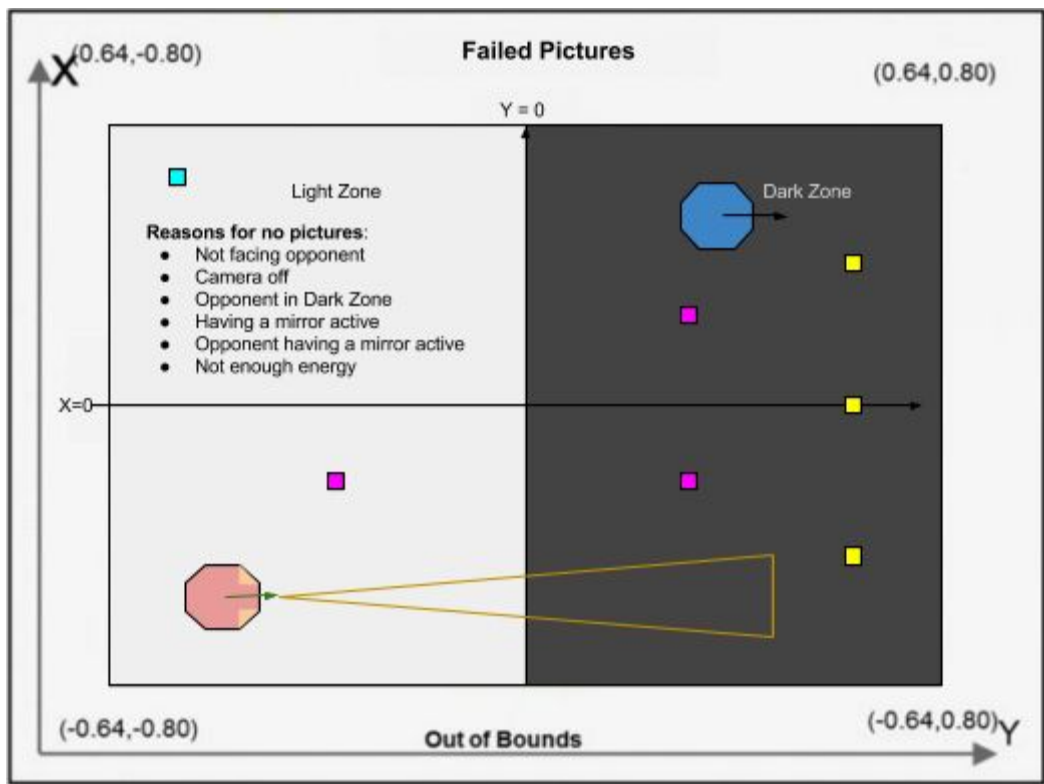
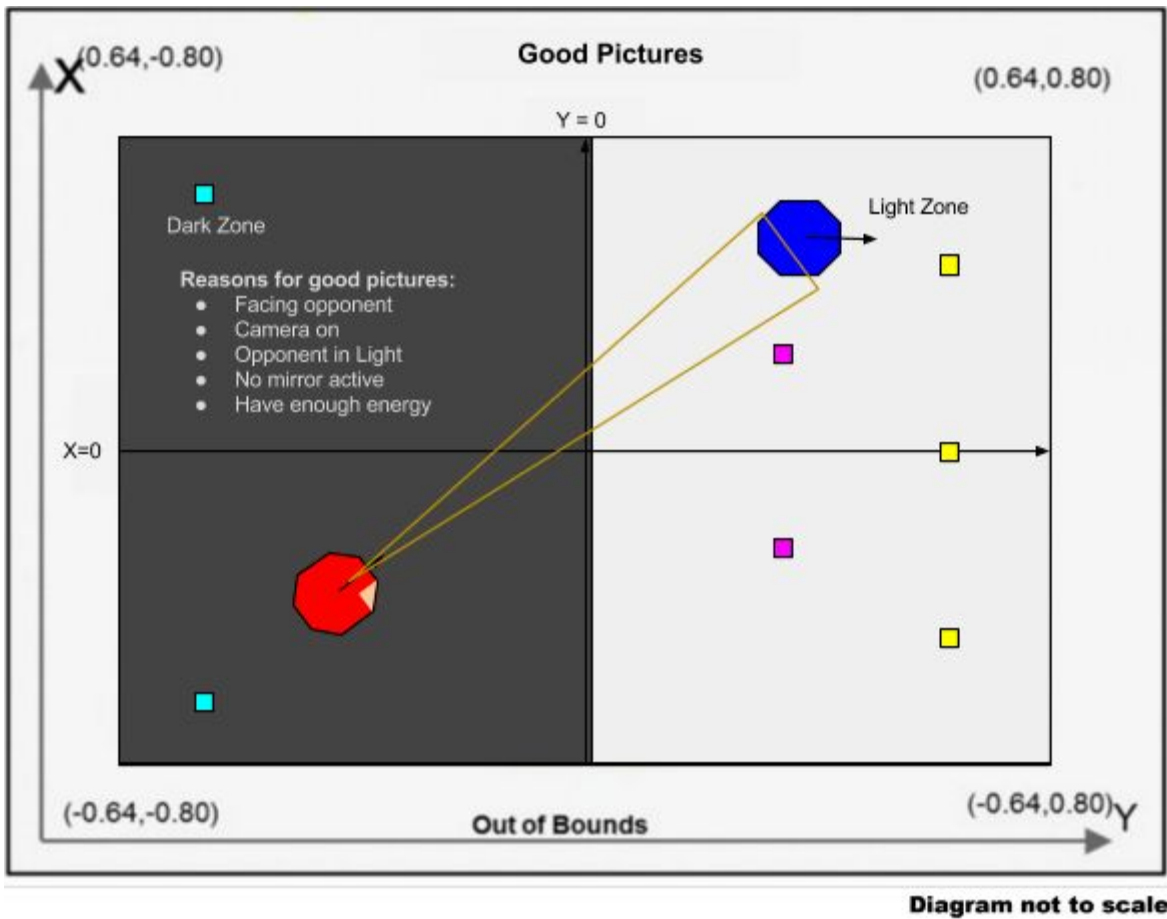


Diagram not to scale

Figure: Conditions for a Successful Picture



### 1.3.6 Scoring Summary

Method	Point Value
Attempting to Take a Picture (Valid or Not)	0.01
<b>Taking a Valid Picture</b>	$2.1 + \frac{0.1}{\text{distance} - 0.5 + 0.1}$ per picture
Score Items	1.5 per item

Whoever is closer to the center/origin of the playing field at game end wins in the event of a tie.





### 1.3.7 End of game

The game ends after 180 seconds. Whichever team has more points wins. In the unlikely case of a tie, the satellite that is closer to the origin wins.

## 2. Tournament

A Zero Robotics tournament consists of several phases called *competitions*. The following table lists the key deadlines for the 2016 tournament season:

Table: Tournament Key Dates 2016

Session 1		
June 13 (Mon)	Start of Session 1	Week 1
June 20 (Mon) or June 21 (Tues)	Field Day	Week 2
June 30 (Thu), 5:00 pm local time	Practice Code Deadline	Week 3
July 8 (Fri), 5:00 pm, local time	Regional Code Deadline	Week 4
July 14 (Thu), 5:00 pm, local time	ISS Code Deadline	Week 5
Mid-Aug	ISS Finals	

Session 2		
July 5 (Tues)	Start of Session 1	Week 1
July 11 (Mon) or July 12 (Tues)	Field Day	Week 2
July 22 (Fri), 5:00 pm local time	Practice Code Deadline	Week 3
July 29 (Fri), 5:00 pm, local time	Regional Code Deadline	Week 4
Aug 4 (Thu), 5:00 pm, local time	ISS Code Deadline	Week 5
Mid-Aug	ISS Finals	



## 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 1<sup>st</sup>, 2<sup>nd</sup>, and 3<sup>rd</sup> 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 1<sup>st</sup> 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 CO<sub>2</sub> 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 12 teams will compete on ISS during the Middle School ISS Final Competition this year. The twelve teams will be divided into 2 conferences for the ISS competition.

Each conference will include 2 brackets of 3 teams each (as shown in Figure 9). 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 Figure 9.) 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

A single semi-final match between the top 2 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 2<sup>nd</sup> place.

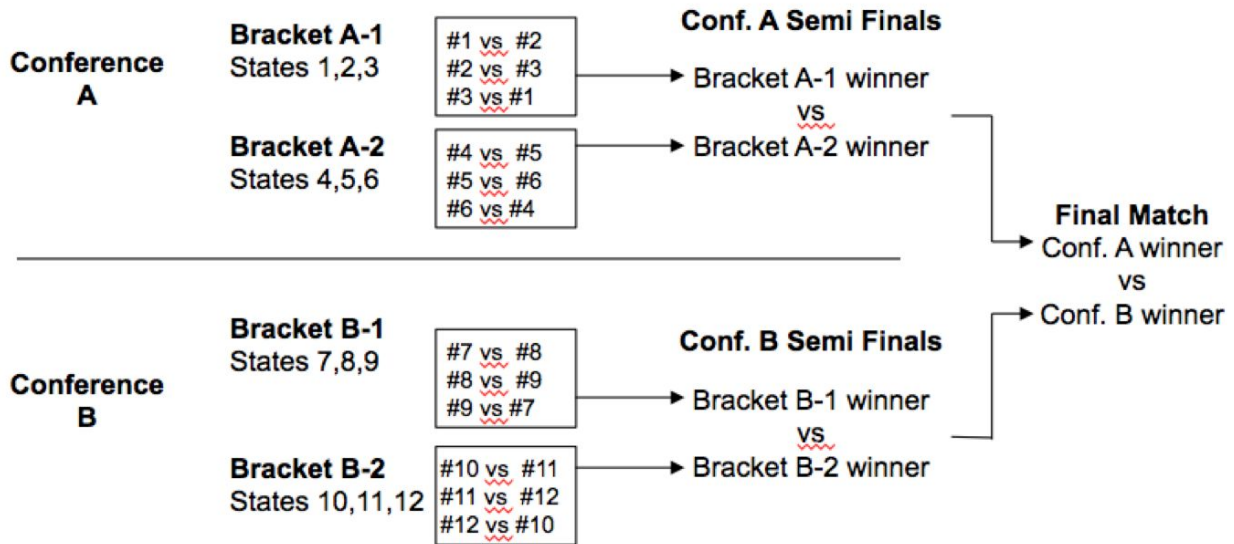


Figure: ISS Competition Bracket

**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<sub>2</sub> 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 High School Tournament 2016 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 SpySPHERES 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`

Name	Description	
NOTE: This function is only available for use with the graphical editor	Moves the player's satellite to the given x, y, and z coordinates.	
void setPositionTarget(float posTarget[3])	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.	
void setAttitudeTarget(float attTarget[3])	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 <i>More Simple Arrays and setAttitudeTarget Function</i> tutorial on the ZeroRobotics website.	
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 parenthesis.	

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

### Game Specific Functions- SpySPHERES MS 2D

The functions in the table below are specific to the game SpySHERES 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.takePic(); //instructs the game to take a picture

### SpySPHERES MS - Pictures


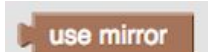




Name	Description	
float takePic()	Attempts to take a picture of the other satellite from its current position and disables the camera for 3 seconds. whether successful or not. Costs 1.0 energy. Returns the number of points that the picture taken is worth.	 
void getAttToOther(float AttToOther[3])	Returns the x, y, z components of the attitude vector needed to point the players SPHERES camera toward the opponent satellite from its current position. The attitude vector is stored in a 3 element array of your choice.	
bool isFacingOther()	Check if the players SPHERES camera is facing the other satellite. Returns true if the players SPHERES camera is facing the other satellite within tolerances. Return false otherwise.	




float getPicPoints()	Returns the amount of points a picture is worth if taken immediately when the camera is on, otherwise returns 0 when the camera is off. This does not take a picture. This costs 0.1 energy only when the camera is on.	
bool isCameraOn()	Checks if the camera is on. Returns true if the camera is on and is usable. Returns false if the camera is off.	

### SpySPHERES MS - Items

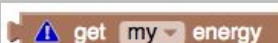



NOTE: **Item Id** is the number assigned to a particular Item (see Game Manual for details)

Name	Description	
bool checkHaveItem(itemID); bool checkHaveItemOther(itemID); bool checkHaveItemNoOne(int itemID)	Checks who (me/other/no-one) has a specified item (items 0-8). Returns true if me/other/no-one has picked up the specified item. Returns false otherwise.	
bool useMirror ()	Uses a held mirror item. Returns true if the item was held and was used and false otherwise.	 
int getNumMirrorsHeld()	Returns the number of mirrors currently held by the player that are available for use	
int getMirrorTimeRemaining ()	Returns the amount of time left on your current mirror. Returns remaining time in seconds is the mirror is active. Returns zero if the mirror is not active.	
void getItemLoc(float pos[3], int itemID)	Copies the location of a given items 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.	

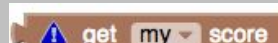
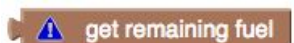


int getItemType(int itemID)	Returns what a given item does Score item returns 0, energy item returns 1, mirror item returns 2.	
-----------------------------	---	---

## SpySPHERES MS - Light/Dark

Name	Description	
float getEnergy() float getOtherEnergy()	Tells how much energy the player satellite currently has.	
bool checkInLight() bool checkInLightOther()	Returns true if the player is in the light zone. False if not.	
bool checkInDark() bool checkInDarkOther()	Returns true if the player is in the dark zone. False if not.	
Int getLightSwitchTime() ( )	Determines how long until the light and dark zone next switch. Returns the number of seconds until the light switches.	

## SpySPHERES MS - Other

Name	Description	
float getScore() float getOtherScore()	Returns player's current score if "my" (Player =0) is selected. Returns opponent's current score if "other's" (Player =1) is selected.	
float getFuelRemaining()	Tells the player how much fuel remains. Returns a float indicating how many seconds of fuel remain.	

## API Note Index

**API Note 1:** Hint: The function `getEnergy()` returns the amount of energy remaining. Once your energy drops below 1.0 you will not be able to activate thrusters.

Hint: If you call this function often, you can allow yourself enough time to make some changes before you run out of energy, for example move toward the light zone where you can



*recharge, stop the SPHERES so it won't drift out of bounds or rotate to point the SPHERE towards an opponent so you can attempt to take pictures.*



## Revision History

Revision	Date	Changes Made
1.0 DRAFT	5/11/16	Initial release
1.1	6/3/16	Updates shown in <b>blue font</b> . Updated Energy pack locations and some API functions.
1.2	6/5/16	Updates shown in <b>orange font</b> . Corrected valid picture scoring
1.3	7/14/16	Updates shown in <b>pink font</b> . Corrected valid picture scoring

