ZERO ROBOTICS ISS PROGRAMMING CHALLENGE
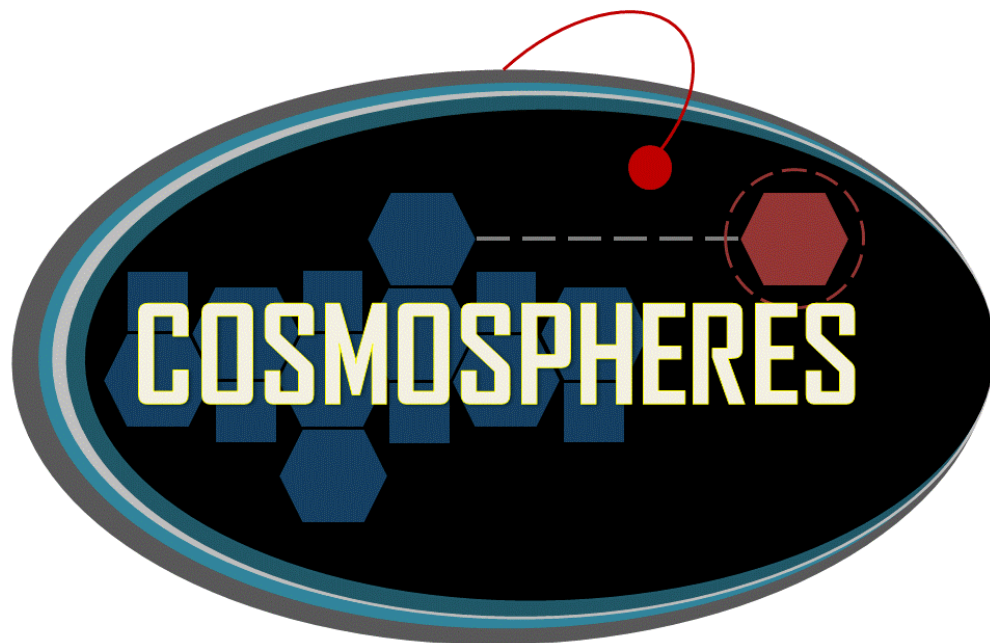
MIDDLE SCHOOL TOURNAMENT 2014:

Critical Operation to Save Mankind from Obliteration (COSMO)

GAME MANUAL

V1.0.0

Dr. Alvar Saenz-Otero   Space Systems Laboratory   Massachusetts Institute of Technology
                              37-381   77 Massachusetts Avenue
                                                           Cambridge, MA 02139

                                                           http://zerorobotics.mit.edu

April 7, 2014

To: Zero Robotics Teams

Re: COSMO program

Attention to all teams:

A comet headed toward Earth threatens humanity's very existence!

Until recently, an effective and plausible choice of defense against this potential disaster has remained elusive. While nuclear weapons have been proposed, serious scientists have not received this idea well.

Instead, recent advances in technology have triggered increasing enthusiasm about the use of robotic satellites to divert the comet. Scientists have suggested that robotic satellites can be launched and strategically placed in orbit and then used to alter an incoming comet's path. Two different approaches are proposed for changing the path of the comet: gravitational attraction and laser pellet repulsion. But which method or combination of methods is more successful?
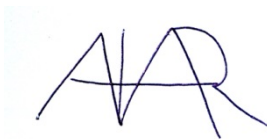
Enter the Critical Operation to Save Mankind from Obliteration (COSMO) program. NASA and DARPA have directed MIT to create two companies, BlueSecurity and RedSecurity. These companies will compete in a challenging demonstration mission with a lucrative comet defense contract at stake. Both companies will use specially outfitted CosmoSPHERES launched into orbit to demonstrate their selected strategy.

Once launched, companies choosing to pursue the gravitational attraction method may collect space debris to increase their mass. This can be done by either colliding with the debris or by lassoing them.

If companies choose to pursue the laser repulsion method, they must rendezvous with laser supply packs that have been pre-positioned as secondary payloads on previous commercial launches.

Once a comet is sighted, the satellite must use gravitational attraction, laser pellet repulsion or a combination of both methods to successfully redirect the comet away from its home base.

As a SPHERES expert, your skills will be in high demand. GOOD LUCK!

Alvar Saenz-Otero

MIT SPHERES Lead Scientist

Table of Contents

1   Game Play

Matches will be played between two SPHERES satellites. Each satellite is controlled by code written by a ZeroRobotics middle school team. Each team attempts to redirect the path of an incoming comet from its home base.

A match has two phases. In Phase 1, satellites begin the game near their home bases. They must navigate a field of space debris. The satellites may collect debris to increase their mass, which increases their ability to redirect the comet via gravitational interaction. Satellites can collect debris by either colliding with the debris or lassoing the debris. In addition, they can collect laser packs, which increases their ability to redirect the comet via laser shots. After 90 seconds have passed in the match, Phase 1 ends and Phase 2 begins. Satellites no longer have the opportunity to collect debris or laser packs. The comets will enter the arena simultaneously and head towards the satellites' home bases. In this phase of the game, the SPHERES must redirect the comets using gravity, which depends on the satellite mass, or by shooting with a laser. One match lasts until both comets have left the arena, or until 180 total seconds have passed in the game (90 seconds in Phase 1 and 90 seconds in Phase 2).

1.1   Game Layout

The Zero Robotics Middle School Tournament 2014 begins with competitions in simulation. The simulated competitions mimic the final competition, which will take place aboard the International Space Station in the summer of 2014.

The middle school game takes place in 2D. The game arena consists of two identical X-Y planes (planes with only X and Y dimensions). Each team's SPHERES satellite is confined to a different plane to prevent collisions between the satellites. The blue satellite operates at Z = +0.2 and the red satellite operates at Z = -0.2. The SPHERES controller ignores any commands that would cause the satellite to move out of its assigned plane or rotate to point in a direction not in its assigned plane. This means that ***it does not matter what Z coordinate you command the satellite to move to***. Because instructions to leave the plane are ignored, you can command Z = 0 or any other convenient value and the controller will set the correct value automatically.

It is not necessary for teams to account in their code for the possibility of being either satellite. Because the X- and Y- coordinates for everything (home base, debris, laser packs, comet entry location, etc.) are the same in each plane, code that teams write will work identically for the red SPHERES and the blue SPHERES.

The game arena encompasses the complete area where the SPHERES satellites can operate. However, the game is played in a smaller area called the *Interaction Zone*. If a satellite leaves the Interaction Zone, it may still be within the arena operational area, but it will be considered out of bounds and its thrusters will force it back inside.

The dimensions of the *Interaction Zone* are:

Table 1 Interaction Zone Dimensions

	2D
X [m]	[-0.64 : +0.64]
Y [m]	[-0.80 : +0.80]
Z[m]	±0.2

This year, the Interaction Zone has two different physical states – one state for Phase 1 and one state for Phase 2. These states are pictured in Figures 1 and 2 on the following pages. Note that the debris field exists only in Phase 1 (the debris and laser packs can only be collected in Phase 1), while the comets exist only in Phase 2. The Interaction Zone has the same dimensions during each phase. It transitions from Phase 1 to Phase 2 after 90 seconds have passed in the game.

Table 2 Debris Field Dimensions

	2D
X [m]	[-0.64 : +0.64]
Y [m]	[-0.50 : +0.50]
Z[m]	±0.2

The Interaction Zone is subdivided into Zones 1, 2, and 3. Zones 1 and 2 are present during Phase 1 and Zone 3 is present during Phase 2.

Zone 1 encompasses the Interaction Zone up until the end of the debris field. Zone 2 encompasses the entire area that lies outside of the debris field on the opposite side of the Interaction Zone as the SPHERES' home bases.

Zone 3 encompasses the entire playing field during Phase 2 of the match.
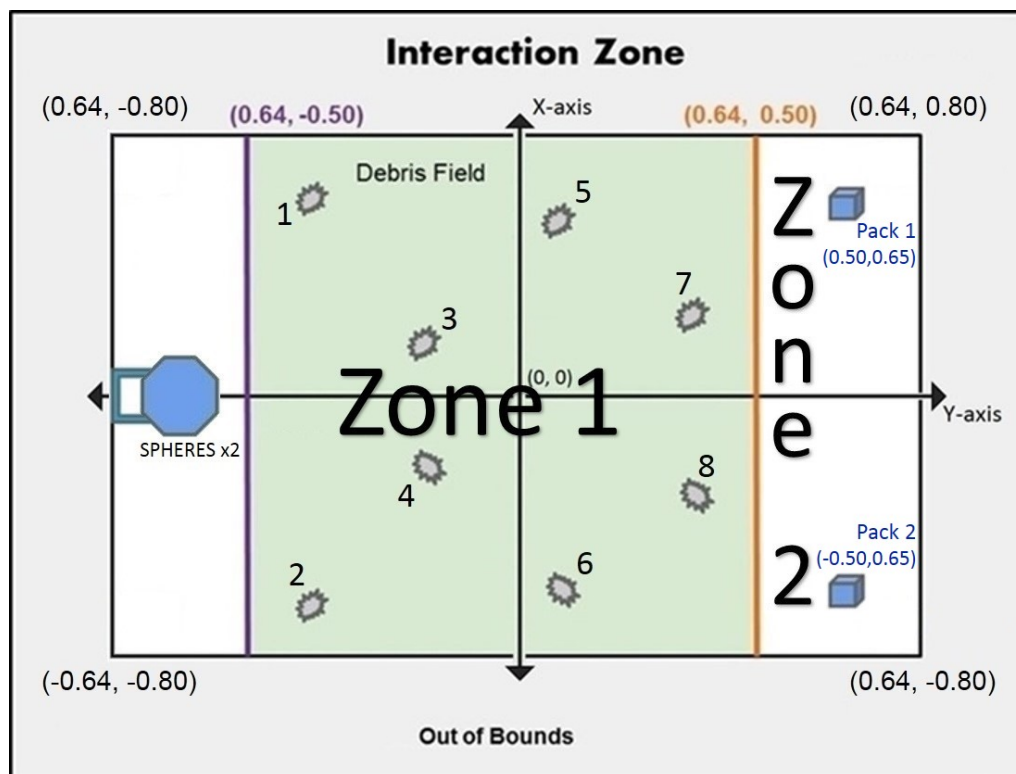
Figure 1 Game Overview Phase 1

Diagram not to scale

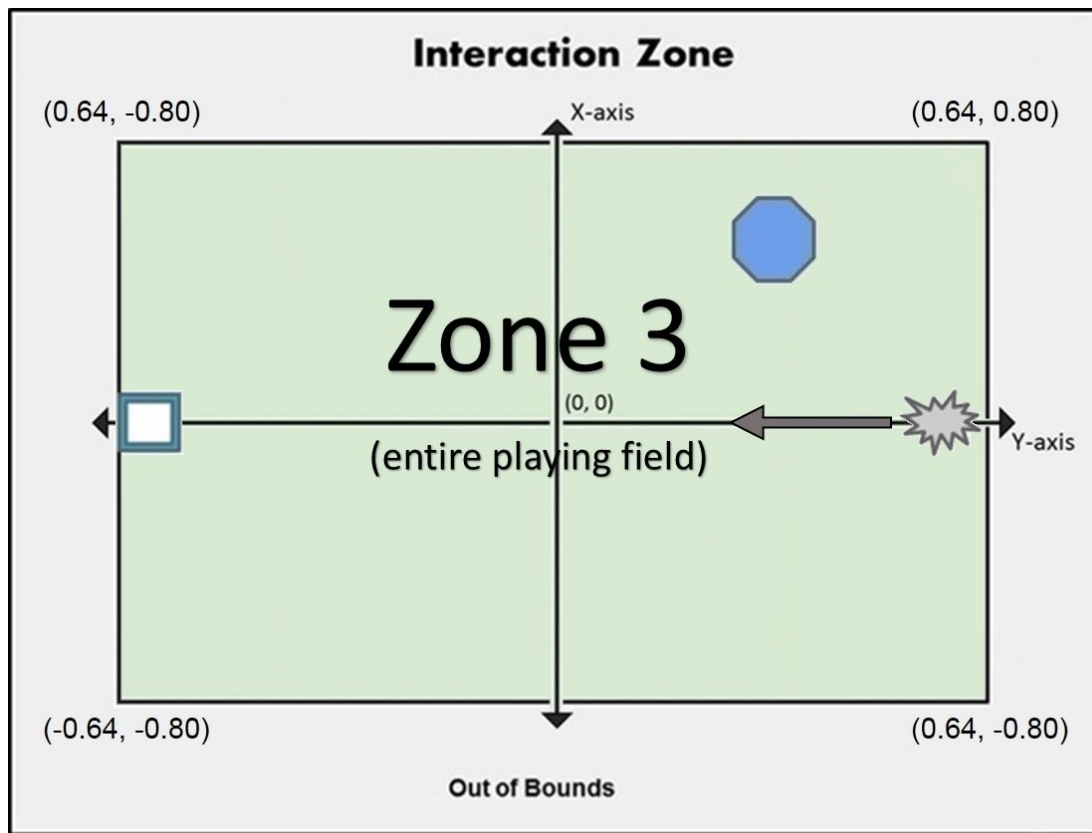**Figure 2 Game Overview: Phase 2 at Time = 90s**

Diagram not to scale

1.2 Satellite

Each team will write computer code to direct a SPHERES satellite through the game. A SPHERES satellite has twelve thrusters and can move in any direction. (For the middle school game, the ability to move to a different Z-coordinate has been disabled.) Like any other spacecraft, 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, Zero Robotics teams have a virtual fuel allocation, which mimics the fuel and battery constraints that the actual SPHERES have.

1.2.1   ZR User API

The non-game-specific functions used to control the SPHERES satellite in ZeroRobotics are located in a document titled "ZR User API" on the ZeroRobotics website (www.zerorobotics.org) under "Resources" → "Tutorials" → "Other Resources." Game specific functions, along with a link to the standard ZR User API functions, are provided in section 5 of this manual.

1.2.2   Time

If the comets have not left the field after 180s of play (Phase 1 and Phase 2 each last for 90 seconds), the game will time out and the scores at that moment will be final. This should be an unusual event.

1.2.3 Fuel

Each player is assigned a virtual fuel allocation (Table 3). The fuel allocation is 50 seconds of thruster firing time. Once the allocation is consumed, the satellite will not be able to respond to SPHERES control commands, but its gravity will still affect its comet. It will fire thrusters only to avoid leaving the Interaction Zone or colliding with the other satellite.

Table 3 Fuel Allocation

	2D
Fuel Allocation [s]	50s

Players consume virtual fuel any time the thrusters are fired. Potential reasons include:

- Motion initiated by player

- Slowdown after colliding with debris or comet (see sections 1.4.1 and 1.6.3)

- Motion initiated by the SPHERES controller to avoid leaving the Interaction Zone (see section 1.7)

1.2.4 Mass

Like all objects made of matter, a SPHERES has mass, which is crucial to its operations in this game. All SPHERES start with a standard initial mass; however, when collecting debris, this mass is increased.

Table 4 Normalized Mass* of Game Elements

	2D
Initial simulated SPHERES mass	1.0×10^{-5}
Comet	1
Debris mass	3.5×10^{-6}

*teams do not need to be concerned with units

1.2.5 Code Size

A SPHERES satellite can fit a limited amount of code in its memory. To reflect this, the COSMOSPHERES game has a specific code size allocation. When you compile your project, the compiler will provide the percentage of the code size allocation that your project is using. Teams may not exceed the maximum allocation when making a formal competition submission.

1.3 Phase 1: Initial Position

Each satellite starts near its home base, which is located at one end of the Interaction Zone.

The SPHERES satellites are deployed at:

Table 5 SPHERES Satellite Deployment Location

	2D
X [m]	0.0
Y [m]	-0.80
Z[m]	±0.2

1.4   Phase 1: Debris Field

Phase 1 of the game contains a debris field with eight identical pieces of debris, numbered 1-8. These debris are symmetric over the y-axis. There is at least one path through the debris field that a SPHERES satellite can navigate without any collisions.

Please note that the two separate XY planes each contain eight debris arranged at the same X- and Y- coordinates. Therefore, *each SPHERES is capable of picking up all eight of its own debris, but none of its opponent's debris.* Which debris your opponent has picked up has no relevance to what debris you are able to pick up.

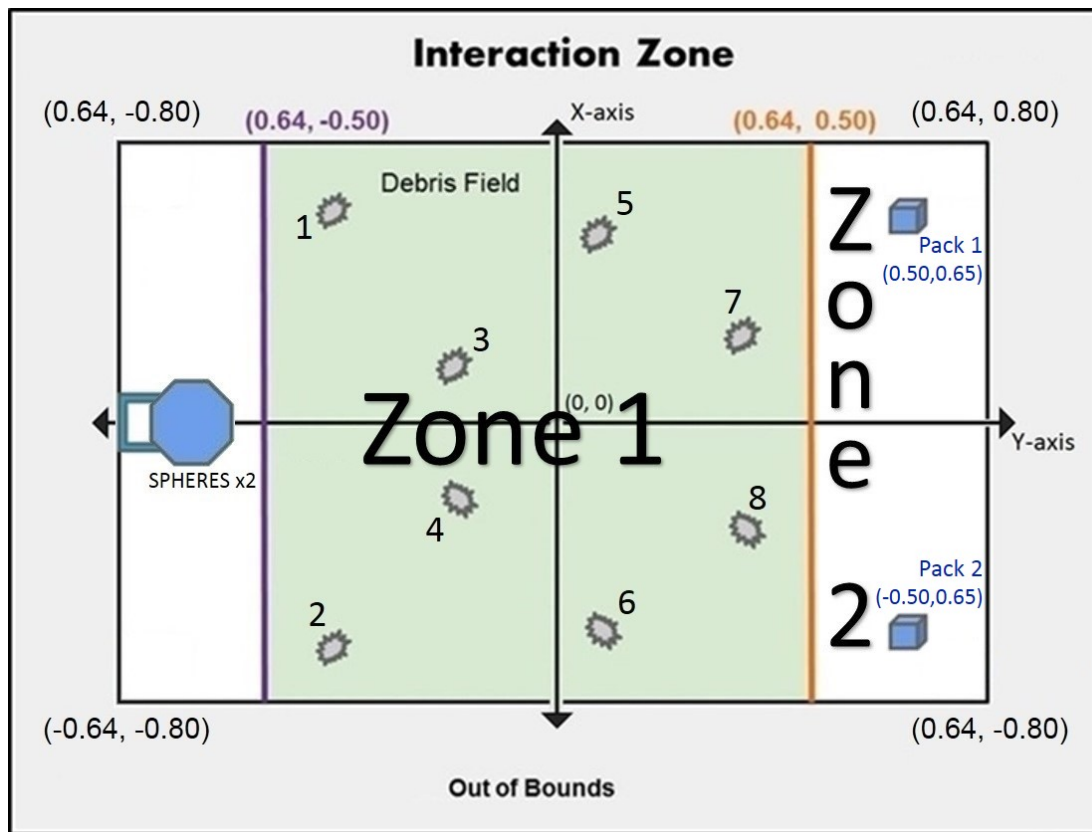**Figure 1: Game Overview Phase 1 (Diagram Repeated)**

Table 6 Debris and SPHERES Measurements

	2D
Debris radius (m)	0.03
SPHERES radius (m)	0.11

Table 7 Debris Numbers and Locations

Debris Number	Location
1	(0.523, -0.5)
2	(-0.523, -0.5)
3	(0.18, -0.2)
4	(-0.18, -0.2)
5	(0.43, 0.15)
6	(-0.43, 0.15)
7	(0.15, 0.37)
8	(-0.15, 0.37)

The satellite's increase in mass depends on the debris collection method. If a satellite collides with debris, it will gain less mass than if it had lassoed the debris. This occurs because a collision would have broken the original piece of debris into smaller pieces.

Table 8 Percentage of Mass Collected from Debris

Method of Collecting Debris	Percentage of Mass Gained
Collision (without Net)	50%
Lasso	100%

API Note: *The function getDebrisLocation(int debrisId, float loc[3]) stores the location of the debris with ID debrisId in the array loc[3]. This will be helpful for navigation. The function haveDebris(int debrisId) returns true if you have successfully picked up the debris with the ID debrisID.*

1.4.1 Collisions

If a SPHERES satellite collides with a piece of debris, the satellite will immediately slow down (both translationally and rotationally). Player controls will lock for three seconds: the student-written program will not be able to command the SPHERESto do anything. After this time period, normal control will return. The fuel needed to slow the satellite is counted against the virtual fuel allocation.
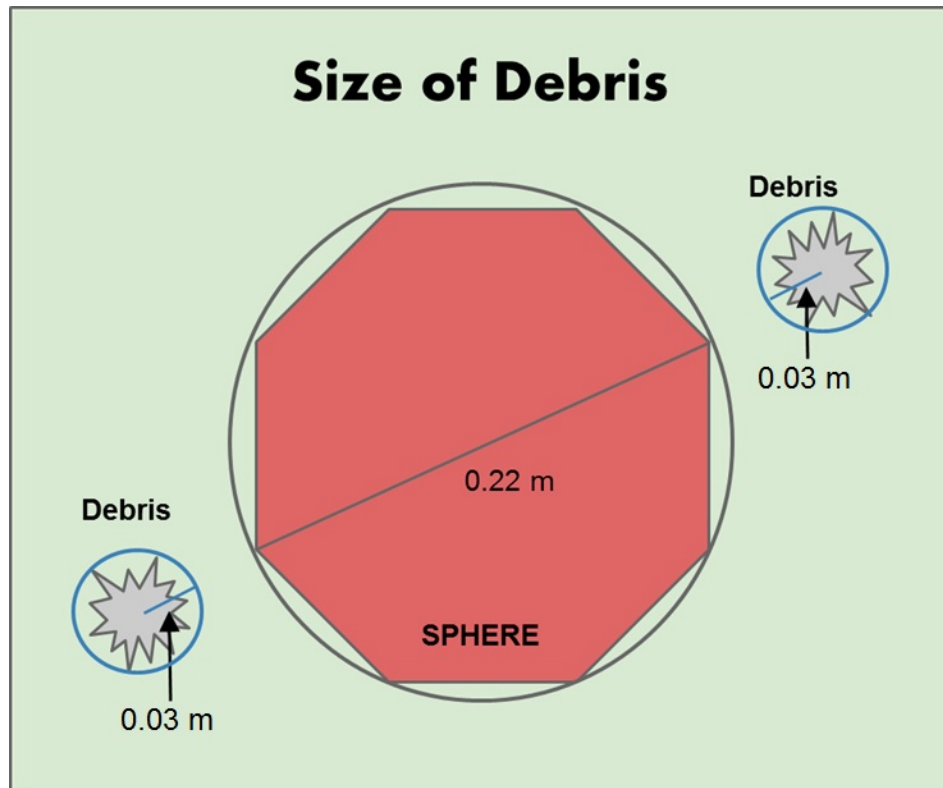
A collision between two objects will occur if the distance between their centers is less than the sum of their radii, in this case, 0.14 m.

Table 9 Maximum Distance Allowed for Collision (m)

	2D
Maximum distance between SPHERES and Debris	0.14

Figure 3 Size of Debris

Size of Debris

API Note: *the function isSlowDownActive() returns true if user thruster control is inactive due to a collision with a piece of debris, and false otherwise*

1.4.2 Lasso

Each SPHERES satellite has a piece of net that can be used for lassoing debris. As shown in Figure 4 (see next page), the satellite must lasso debris by traveling from one side of the debris to the other, in an approximately semi-circular path. During the entire procedure, the distance between the SPHERES and the debris must be within the minimum and maximum distances, specified in the table below.
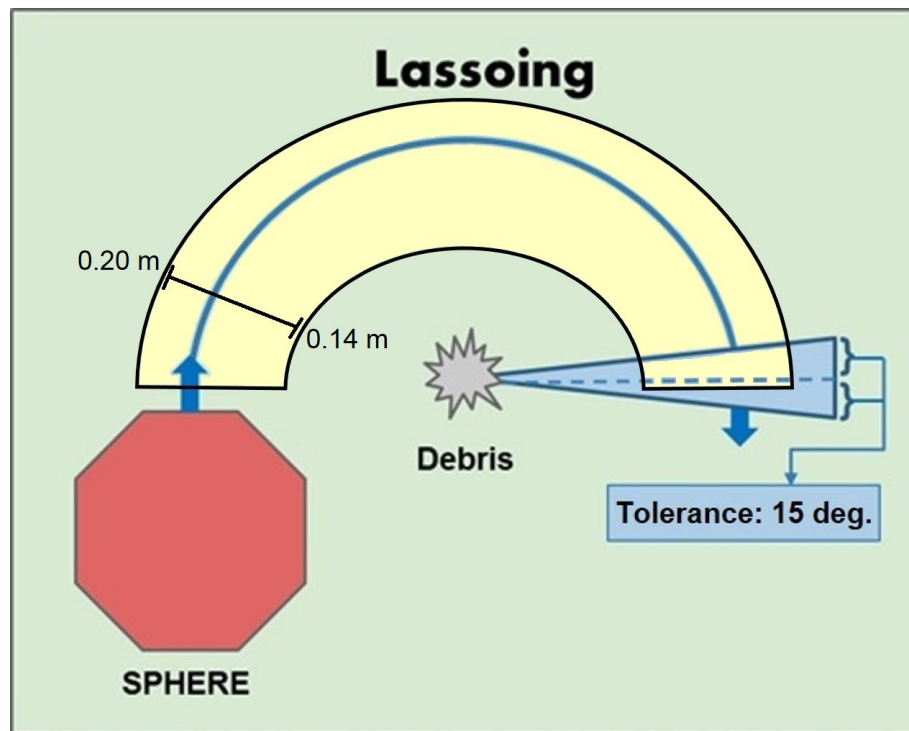
Table 10 Restrictions When Lassoing Debris (m)

	2D
Minimum distance between SPHERES and Debris	0.14
Maximum distance between SPHERES and Debris	0.20

If a collision occurs with the debris that is being lassoed, it will be counted as a collision and not as lassoing.

Figure 4 Half Aperture

Steps for Lassoing Debris:

1. Position the satellite at least 0.14 away from the debris' center
   - use startLasso()
2. Find the vector from the debris to the satellite (this is helpful to figure out where the SPHERES should travel next)
3. Move halfway (180° with a 15° margin of error) around the piece of debris, while staying within the specified distances
   - use setPositionTarget()

API Note: *the function startLasso(int debrisId) will initialize the lassoing process of the debris with ID debrisId. It will return true if lassoing is possible and false otherwise. Hint – use the function setPositionTarget() multiple times to stay inside the semicircular path*

1.5   Laser Pack Collection

After the SPHERES have crossed the debris field, they can pick up one or both laser packs to assist with comet deflection. There are two identical laser packs, each of which gives the player a fixed number of laser shots (see table below).

Please note – there are only two laser packs, and they are accessible to either SPHERES. This means that, if your opponent takes the laser pack at a particular location before you do, it is no longer accessible to you.

Table 5 Laser Pack Data

Laser shots per pack	10
Laser 1	
X [m]	0.50
Y [m]	0.65
Laser 2	
X [m]	-0.50
Y [m]	0.65

*Note – since either SPHERES can pick up either laser pack, the Z-coordinates of the packs are unimportant

In order to collect a laser pack, the satellite must perform a spinning maneuver (see Figure 5). The steps are:

1. Position the satellite within 0.05m of the laser pack's center.

    • Use setPositionTarget()

    • The satellite must be stopped (velocity is less than 0.01m/s)

    • The satellite cannot be spinning (angular velocity is less than 2.3°/s)

2. Rotate the satellite 90° (rotate around the Z-axis)

    • Use setAttitudeTarget()

Figure 5 Maneuver to Collect Laser Pack
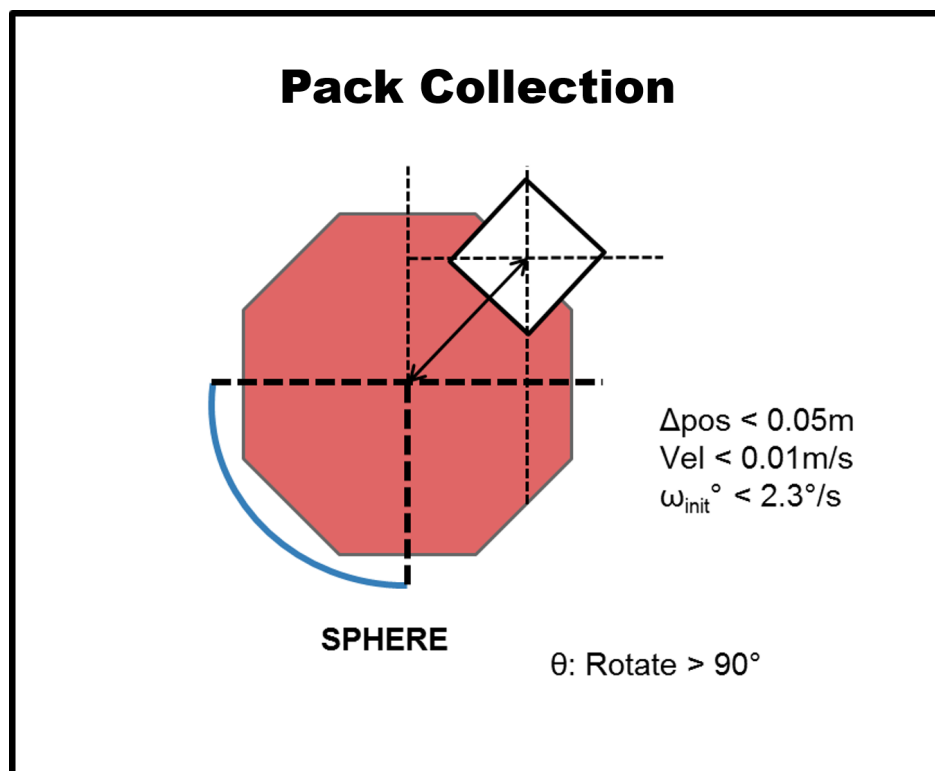
API Note: bool havePack(int player, int objectNum) returns true if player owns the laser pack with number objectNum Hint: use setAttitudeTarget() to rotate the satellite.

1.6   Comet Deflection

After 90 seconds have passed in the game, comets enter the competition field and head toward the home bases. The initial velocity of each comet is 0.3 m/s directly toward its satellite's home base. A satellite can only affect its own comet. (The blue SPHERES can only affect the blue comet; the red SPHERES can only affect the red comet.) Satellites may deflect the comet by using gravity, the laser, or a combination of the two. The comet has a radius of 0.07 m. Regardless of the method used to redirect the path of the comet, it is important to pay attention to the distance between the SPHERES's comet and the SPHERES itself, in order to avoid activating collision avoidance. Collision avoidance will be activated when the distance between the centers of the SPHERES and comet is 0.18 m (the sum of the radii of the SPHERES and the comet). Collision avoidance will only be activated between a SPHERES and its own comet. This means that each SPHERES can pass through the other SPHERES's comet without affecting itself or the comet.

Table 12 Home Base and Comet Entrance Locations

	Home Base	Comet Entrance
X [m]	0.0	0.0
Y [m]	-0.80	0.8
Z[m]	±0.2	±0.2

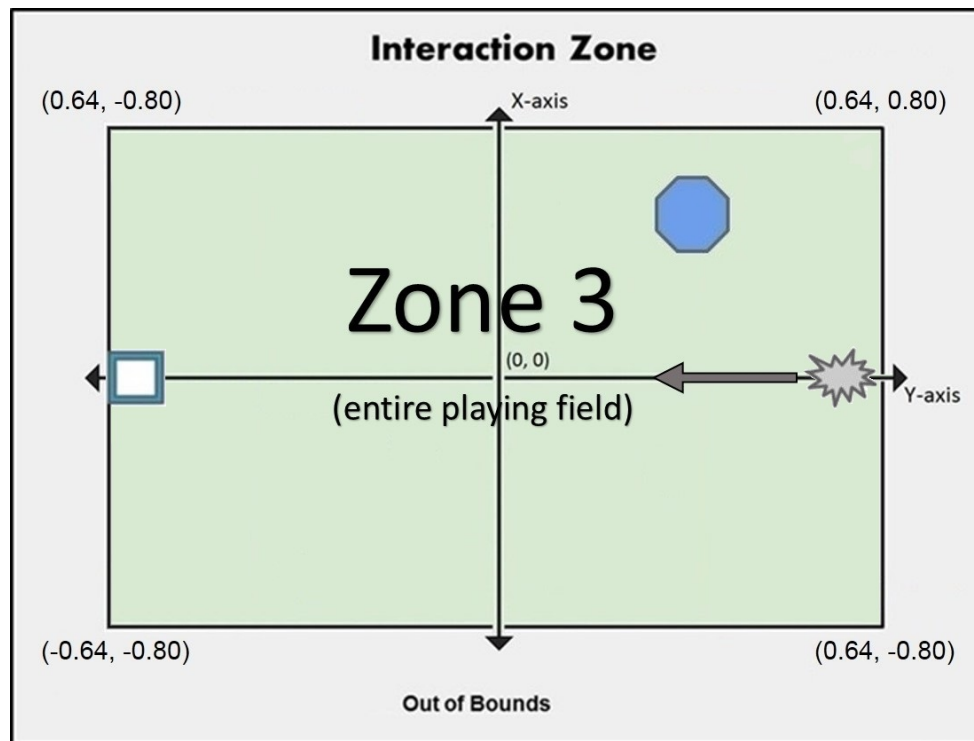**Figure 2 Game Overview: Phase 2 at Time = 90s (Diagram Repeated)**

Diagram not to scale

API Note: The function getCometState(float state[6]) stores the comet location of your comet into the array state. The more advanced function predictCometState(unsigned int dtSteps, float initState[6], float finalState[6]) uses the initState of a comet and stores the predicted location of the comet in dtSteps (seconds) into finalState based on linear extrapolation using the current velocity.

1.6.1 SPHERES Gravitational Force

The effect of gravity between the SPHERES satellite and the comet is based off of Newton's Law of Universal Gravitation:

$$F_{gravity} = G \frac{m_{satellite} m_{comet}}{distance^2}$$

In this equation, G is a constant, $m_{satellite}$ is the mass of the satellite (this incorporates the SPHERES's own mass as well as the the mass of the debris it has collected), m_{comet} is the mass of the comet, and distance is the distance between the center of the SPHERES and the center of the comet.

For the purposes of the COSMOSPHERES game, G has been normalized to 1. Please note that, by normalizing G to 1, we have intentionally made the effects of gravity stronger in this game than they are in real life. The mass of the comet is 1. The mass of the satellite is the mass of the satellite itself (1.0×10^{-5}) combined with the mass of all the debris it has collected. You need to figure out the mass of the debris that your satellite has collected; you also need to figure out the distance between your satellite and the comet. Taking all of these things into account, our "new" formula for the gravitational force betweent the SPHERES and the comet is:

$$F_{gravity} = \frac{(1.0 \times 10^{-5} + m_{collected\ debris})}{distance^2}$$

Simply put, the closer the SPHERES is to the comet, the more effect it will be able to have on the comet's trajectory. In addition, the more mass the SPHERES has, the more it will be able to affect the comet's trajectory.

Figure 6 SPHERES Gravitational Force
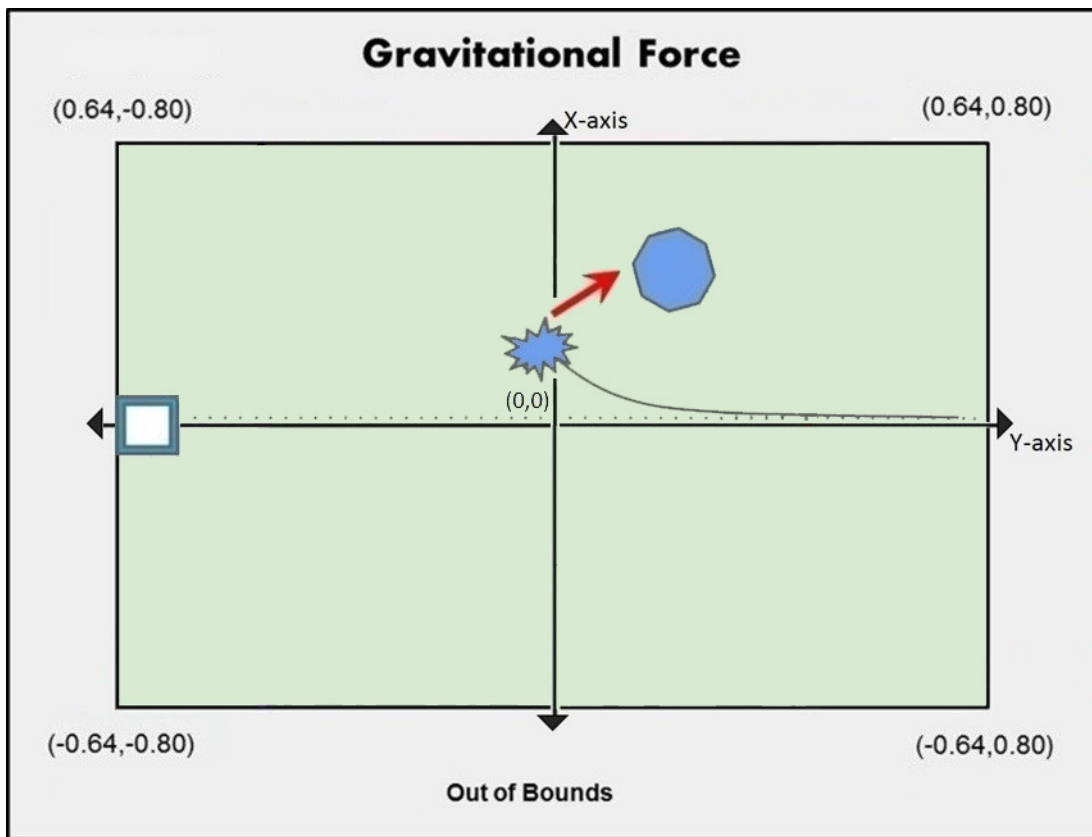
Deflection exaggerated for purposes of explanation

1.6.2 Shooting Lasers

The laser fires in short bursts. When the laser beam hits the comet, it changes the comet's velocity. (Addition of vectors can be used to calculate the comet's change in velocity.) Because the comet's velocity is changed, we can say that its linear momentum is also changed. We refer to this change in momentum as an impulse. The laser produces a fixed impulse (see Table 12 below). Because the mass of the comet is 1, the numerical value of the impulse is equal to the comet's change in vector velocity [impulse = change in (mass*velocity)]. Again, see Table 12.

The laser fires from the center of the satellite in the direction the satellite is pointing (it shoots from the -X face of the SPHERES). The laser is not guaranteed to hit the comet. The laser is considered to hit the comet if its beam touches any part of the comet. In addition, regardless of where the beam hits, the impulse is still along the line between the center of the SPHERES and the center of the comet. This means that, regardless of where the laser touches the comet, it is still considered a direct hit. The comet is sphereically shaped, with radius 0.07, as seen in Table 12 below.

Each laser can only be shot ten times. The laser can be fired once per second. It may be fired at any range. It will not affect the other SPHERES or the other comet.

Note: Keep in mind that the force of gravity between the SPHERES and the comet is always present, independent of the strategy selected.

API Note: *the function faceTarget(float target[3]) will rotate the satellite to face the target (which can be the comet if you input the comet's position). The function isFacingComet() returns true if the satellite is facing the comet. The function shootLaser() will shoot a laser at the comet and return true if the laser beam is successfully shot and it successfully hits the comet. The function laserShotsRemaining() will return the number of laser shots remaining. Note that calling shootLaser() expends a laser shot regardless of whether the comet is hit. Calling shootLaser() once all shots have been expended has no effect and will return false.*

Table 13 Laser Parameters

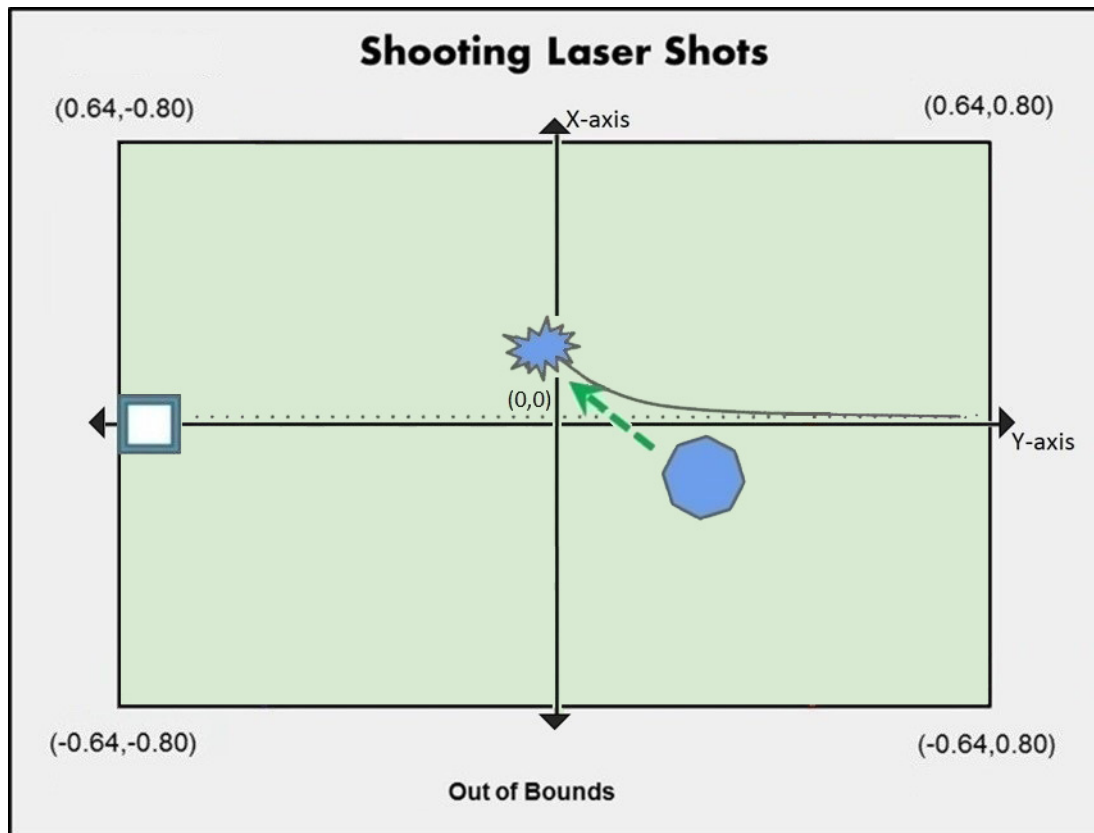|	
Laser impulse → change in comet's vector velocity	0.00123
Comet radius	0.07 m

Figure 7 Shooting Lasers

Diagram not to scale

1.6.3 Comet Collisions

A collision occurs when the distance between the centers of a SPHERES and its comet is 0.18m. If a SPHERES collides with its comet, the SPHERES controller will lock and the SPHERES will bounce back from the comet. In this case, bouncing means that the SPHERES will move back towards the direction from which it came, with the same speed that it was travelling at when it hit the comet. In mathematical terms, When bounce is activated, the translational vector of the SPHERES will be in the opposite direction of the original motion, but the magnitude of the velocity vector will remain the same. Player controls will lock until the circumstances for collision are no longer fulfilled. A SPHERES can only collide with its own comet; it cannot be affected by its opponent's comet. Note that gravity is active during all of Phase 2.

API Note: *bool isBounceActive() returns true if satellite is under collision conditions with its comet (user thruster control is inactive) and false otherwise*

1.6.4 End of Game

The game ends when both comets leave the Interaction Zone, or after a total of 180 seconds (90 in Phase 1, 90 in Phase 2) have passed in the match.

1.7 *Out of Bounds*

You must remain within the boundaries of the Interaction Zone to avoid a fuel penalty.

If you exit out of bounds (this includes trying to enter your opponent's Zone 1 during Phase One), the SPHERES controller will override your commands and force your satellite to stop its motion in the direction that would continue to push it out of bounds (other directions are not affected). The fuel used to stop this motion will be charged to your fuel usage. There is an additional fixed penalty of **2 thruster-seconds** (4% of total initial fuel) for each second spent out of bounds.

2   Scoring

Your score is based only on where your comet left the Interaction Zone. The distance between your home base and where your comet left the Interaction Zone (as measured around the perimeter of the Interaction Zone) is calculated. The ZeroRobotics team puts these distances into a special formula to calculate your score. In order to maximize your score, you want your comet to exit the Interaction Zone as far from your base as possible.

3   Tournament

A Zero Robotics tournament consists of several phases called *competitions*. The following table lists the key deadlines for the 2014 tournament season:

Table 64 Tournament Key Dates

Date (2014)	Schedule 1	Schedule 2
June 9 (Mon)	Program begins	
June 27 (Fri), 5 PM	Code submittal deadline: *Practice Regional Competition*	
July 7 (Mon)		Program begins
July 11 (Fri), 5 PM	Code submittal deadline: *Regional Competition*	
July 17 (Thurs), 5 PM	Code submittal deadline: *ISS Competition*	
July 25 (Fri), 5 PM		Code submittal deadline: *Practice Regional Competition*
Aug 1 (Fri), 5 PM		Code submittal deadline: *Regional Competition*
Aug 7th (Thurs), 5PM		Code submittal deadline: *ISS Competition*
Mid- August (TBD)	ISS Finals Event	ISS Finals Event

3.1   Regional Simulation Competition

3.1.1   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 Friday of Week 3 (the date may vary by region.)
- **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 Friday of Week 4.

3.1.2   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 ZeroRobotics 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.

3.1.3   Competition Format – Regional Competition

The regional competition will be a round robin, with every team playing every other 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 EDT 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.

3.2   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.

3.3   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.

3.3.1   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_2 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.

3.3.2   Competition Format

- **ISS pool play** The finalist teams (representing their region) are randomly divided into three pools of three teams each. Each pool plays a round robin on the ISS. The team with the most wins is the pool winner. If all three teams have one win, the team with the highest total score in its matches is the winner. If this procedure fails to break the tie, the winner will be determined randomly.
- **Championship round** The winners of the three pools play a round robin with the same rules as in pool play. The winner is the 2014 Zero Robotics Middle School Champion!

3.3.3   Definitions

Definition: *Successful Match (aboard the ISS)*

- 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

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.

3.3.4   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.

4   Season Rules

4.1   Tournament Rules

All participants in the Zero Robotics Middle School Tournament 2014 must abide by these tournament rules:

1. The Zero Robotics team (MIT / Top Coder / Aurora) can use/reproduce/publish any submitted code.

2. 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.

3. Teams are expected to report all bugs as soon as they are found.

3.1. A "bug" is defined as a contradiction between the intent of the game and behavior of the game.

3.2. The intent of the game shall override the behavior of any bugs up to code freeze.

3.3. 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).

4. Code and manual freeze will be in effect 3 days before the submission deadline of a competition.

4.1. Within the code freeze period the code shall override all other materials, including the manual and intent.

4.2. 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.

4.3. The code is finalized at the ISS Final Competition freeze (unless there is a critical issue which will affect the final tournament, including lessons learned from ground hardware testing and simulation.)

5. Changes in the game manual may occur as a result of lessons learned from earlier parts of the tournament.

4.2   Ethics Code

- When an unethical situation is reported, the ZeroRobotics team will work diligently to deal with the situation. Such situations will be dealt with on a case-by-case basis.

- Teams are strongly encouraged to report bugs as soon as they are found; intentional abuse of an un-reported 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.

5   CosmoSPHERES API Reference

The functions in this section are called as members of the game object, that is, game.functionName(arguments);

Functions highlighted in yellow are general game functions. Functions highlighted in green are Phase 1 functions; functions highlighted in blue are Phase 2 functions.

Name	Description
float getMass()	Returns the mass of the player's satellite
float getFuelRemaining()	Returns the fuel the player has left (in thruster-seconds)
float getScore()	Returns the player's score (this will be 0 in Phase 1)
float getOtherScore()	Returns the opponent's score (this will be 0 in Phase 1)
void getDebrisLocation(int debrisId, float loc[3])	Stores the location of the debris with the id **debrisId** in the array **loc**
bool haveDebris(int debrisId)	Returns true if you have successfully picked up the debris with the id **debrisId**
bool startLasso(int debrisId)	Initialize the lassoing process of the debris with id **debrisId**. Returns true if lassoing is possible, false otherwise
bool havePack(int player, int objectNum)	Returns true if **player** (ME = 0, OTHER = 1) has the laser pack **objectNum** (1 or 2)
bool isSlowDownActive()	Returns true if satellite collided with a debris, false otherwise
void predictCometState(unsigned int dtSteps, float initState[6], float finalState[6])	Uses the **initState** of a comet and stores the predicted location of the comet in **dtSteps** (seconds) into **finalState** based on linear extrapolation using the current velocity
void getCometState(float state[6])	Stores the location of your comet into the array **state**
bool shootLaser()	Shoots a laser at the comet, returns true if laser hits, returns false if no laser shots left or if the laser does not hit
int laserShotsRemaining()	Returns the number of laser shots the player has left
bool isBounceActive()	Returns true if satellite is under collision conditions with comet
bool isFacingComet()	Returns true if satellite is facing comet, false otherwise.
void faceTarget(float target[3])	Rotates the satellite to face the target (which can be the comet if you input the comet's position)

A guide to the standard Zero Robotics API functions is available here:

http://www.zerorobotics.org/documents/10429/374963/ZR_user_API.pdf

6   Lists of Figures and Tables

6.1   List of Figures

6.2   List of Tables

7 Revision History

Revision	Date	Changes	By
1.0.0	2014/05/05	Initial Release	erikah@mit.edu