



# Zero Robotics ISS Programming Challenge Australian Preliminary Competition 2016



## Conducting Optical Research on Nearby Asteroids (CORONA) GAME MANUAL

6 May 2016

To: Zero Robotics Teams

Re: CORONA program

Attention to all teams:

NASA has observed a recent spike in the number of asteroids knocked loose from the asteroid belt and into close proximity to Earth. Using highly sophisticated algorithms, they have used the mass of these space rocks to predict the composition of these asteroids with great accuracy. However, there exists amongst these rocks, a small percentage that are of comparable mass percentages but do not fit the model. Eager geologists believe scientists may have discovered a new element and immediately rally for further study.

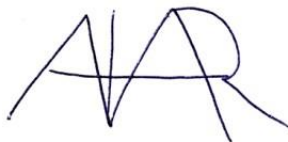
NASA calls upon its fellow scientists at MIT for a plan of action. With the help of these researchers, the joint CORONA program was born. CORONA (Conduction Optical Research On Nearby Asteroids) requires the use of MIT's most recent project, SPHERES, to capture visuals of the closest asteroids to Earth's atmosphere.

MIT researchers are fairly confident that they have stumbled upon something novel in these asteroids. However, some believe that there may actually be ice on these rocks and not a new element at all. The contrasting opinions lead to the creation of two separate SPHERES teams. Both teams will be using their respective satellites to take up-close photos of the asteroids at specific points of interest, as determined by NASA.

Soon after launch, however, NASA catches sight of incoming solar flares at the exact location where the SPHERES are intending to intercept the asteroids. The satellites risk imminent mechanical issues and loss of their stored photos if they impact with the solar flare while turned on. The MIT teams have to decide whether they want to use the shadow zone of the asteroid for protection or temporarily power down when a solar flare is near.

The choice is a tough one, but one that must be made. Each team is counting on its satellite to find visual proof of why either water or a new element exists on these space rocks. Proof of either would be invaluable, but conclusive evidence is essential.

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



Alvar  
MIT SPHERES Lead Scientist

Saenz-Otero



# Contents

|   |    |
|---|----|
| 1. Game Overview .....                        | 4  |
| 1.1 Game Layout.....                          | 4  |
| 1.2 Satellite.....                            | 5  |
| 1.2.1 ZR User API.....                        | 6  |
| 1.2.2 Time.....                               | 6  |
| 1.2.3 Fuel.....                               | 6  |
| 1.2.4 Inter-satellite Communications.....     | 6  |
| 1.2.5 Code Size .....                         | 7  |
| 1.3 Initial Position.....                     | 7  |
| 1.3.1 Player ID.....                          | 7  |
| 1.4 Game Play.....                            | 7  |
| 1.4.1 Picture Taking.....                     | 8  |
| 1.4.2 Memory Upgrade Packs.....               | 10 |
| 1.4.3 Solar Flares .....                      | 11 |
| 1.4.4 Upload .....                            | 12 |
| 1.4.5 Collisions.....                         | 13 |
| 1.4.6 End of Game.....                        | 14 |
| 1.5 Item Collection.....                      | 14 |
| 1.6 Out of Bounds.....                        | 15 |
| 2. Scoring .....                              | 16 |
| 3. Tournament.....                            | 16 |
| 4. Season Rules .....                         | 17 |
| 4.1 Tournament Rules .....                    | 17 |
| 4.2 Ethics Code.....                          | 17 |
| 5. ZR User API .....                          | 18 |
| 5.1 Standard Zero Robotics API Reference..... | 18 |
| 5.2 CoronaSPHERES API Reference .....         | 18 |

# 1. Game Overview

Matches will be played between two SPHERES satellites, controlled by code written by two different teams. Satellites start the game facing the asteroid. Each team will attempt to take pictures of special points of interest (POI). These points of interest change location once every minute. The SPHERES can hold up to two pictures in its memory at a time. During the game, teams have the option of picking up a memory upgrade pack which allows players to store extra pictures before needing to upload. There are two zones around the asteroid which determine the amount of points each picture is worth. Points are received only after uploading the pictures stored in each satellite's memory. Throughout the game, there are solar flares that can corrupt the pictures in memory, and if precautions are not taken, can also damage the satellite. Satellites are completely safe only in the shadow zone.

## 1.1 Game Layout

The Australian Preliminary competition mimics the operational volume available aboard the International Space Station. The arena includes X, Y, and Z components. The game arena encompasses the complete area where the SPHERE satellites can operate as shown in Figure 1. However, the game is played in a smaller area called the 'Interaction Zone'. If players leave the *Interaction Zone*, they may still be within the arena operational area, but they will be considered out of bounds.

The dimensions of the *Interaction Zone* are:

**Table 1: Interaction Zone Dimensions**

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

Within the interaction zone there are three zones: Danger, Inner, and Outer zones.

When satellites enter the danger zone thrusters are automatically turned on to move them out of the danger zone. Inner and Outer zones only affect the amount of points a picture is worth.

The shadow zone represents the area safe from Solar Flares.

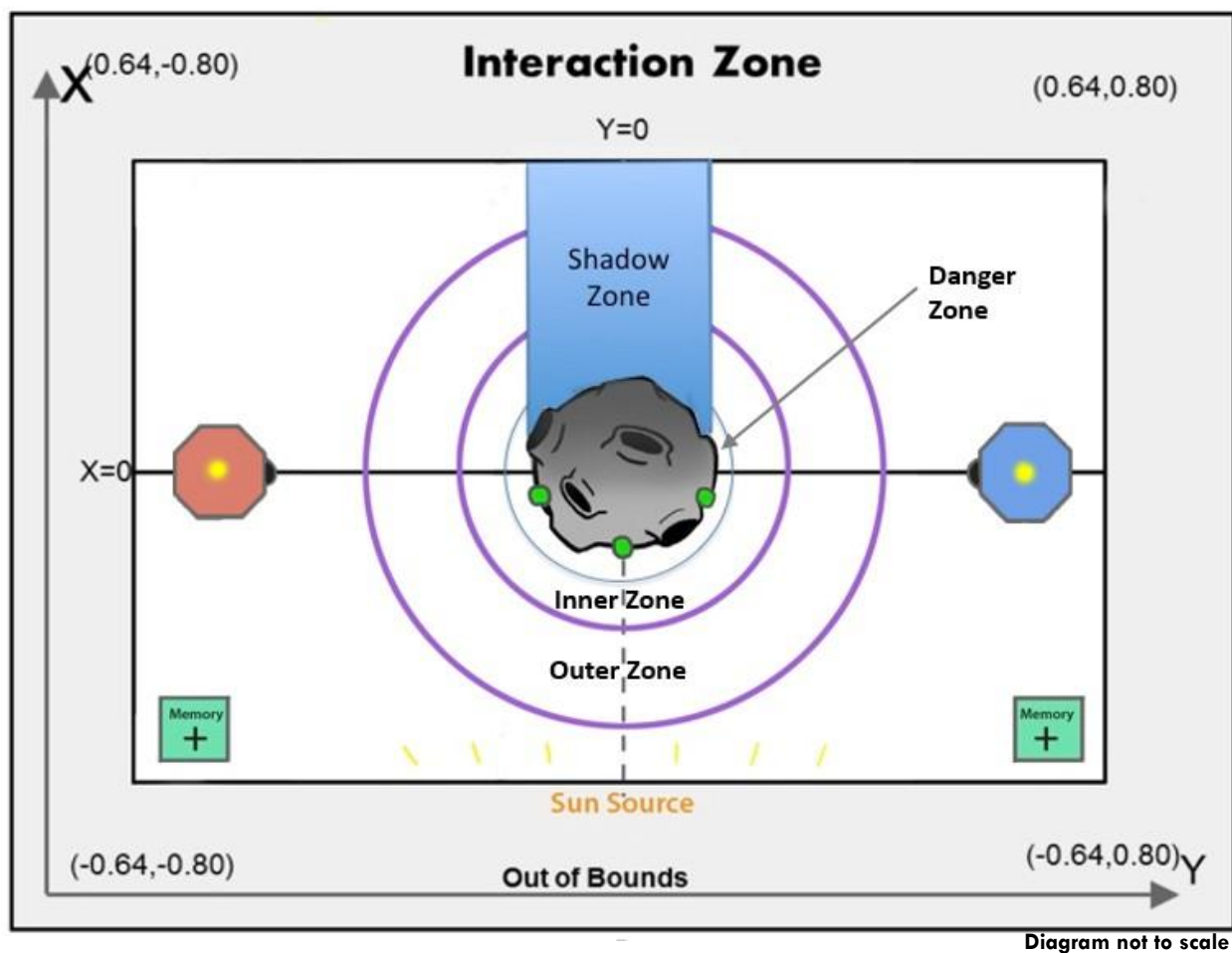
**Table 2: Zone Radii Positions**

|                                    |      |
|------------------------------------|------|
| Danger min radius                  | 0.2  |
| Danger max radius/Inner min radius | 0.31 |
| Inner max radius/Outer min radius  | 0.42 |
| Outer max radius                   | 0.53 |

**Table 3: Shadow Zone Dimensions**

|       |              |
|-------|--------------|
| X [m] | [0.0: +0.64] |
| Y [m] | [-0.2: +0.2] |
| Z [m] | [-0.2: +0.2] |

**Figure 1: Game Overview**



## 1.2 Satellite

Each team will write the software to command a SPHERES satellite to move in order to complete the game tasks. A SPHERE satellite can move in all directions using its twelve thrusters. The actual SPHERES satellites, 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 in order to wisely use virtual SPHERES resources.

## 1.2.1 ZR User API

The non-game specific functions used to control the SPHERES satellite in Zero Robotics can be found in a document titled “ZR User API” on the Tutorials webpage under “Other Resources”. Game specific functions, along with a link to the standard ZR User API functions, are also provided in section 5 of this document.

## 1.2.2 Time

Players have 240s to take and upload as many photos as possible. After 240s scores will be final and compared.

## 1.2.3 Fuel

Each player is assigned a virtual fuel allocation (Table 4) which is the total sum of fuel used in seconds of individual thruster firing. Calling the function `float getFuelRemaining()` returns remaining fuel. 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 colliding with other satellite.

**Table 4: Fuel Allocation**

|                     |      |
|---------------------|------|
| Fuel Allocation [s] | 120s |
|---------------------|------|

The virtual fuel allocation is consumed any time the thrusters are fired. Potential reasons include:

- Motion initiated by player
- Motion initiated by the SPHERES controller to avoid leaving the Interaction Zone (see section 1.6)
- Motion initiated by the SPHERES controller to avoid a collision with the other satellite.

## 1.2.4 Inter-satellite Communications

The satellites have the ability to communicate with each other using binary messages. The API functions `sendMessage` and `receiveMessage` may be used to send data between the satellites. The bandwidth available to the satellites is as follows: (adding function for cooperation)

**Table 5: Inter-Satellite communications bandwidth**

|              |               |
|--------------|---------------|
| Message size | Unsigned char |
|--------------|---------------|

## 1.2.5 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 is 100% or less of the total allocation.

## 1.3 Initial Position

Each satellite starts on the X axis on opposite sides of the asteroid. The SPHERES satellites are deployed at:

**Table 6: SPHERES Satellite Deployment Locations**

| Red   |      |
|-------|------|
| X [m] | 0.0  |
| Y [m] | -0.6 |
| Z [m] | 0.0  |
| Blue  |      |
| X [m] | 0.0  |
| Y [m] | 0.6  |
| Z [m] | 0.0  |

### 1.3.1 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.

## 1.4 Game Play

A *random* set of 3 POIs will be visible on the surface of the asteroid (sunny side) every 60 seconds starting at time = 0. Each set of POIs will be distributed symmetrically with respect to the x-z plane with two of the points equal distance from y=0 in the + y and - y directions and the third point located on the y=0.

Visible POIs are assigned on identification number (ID...). The ID # will be 0, 1, or 2.

Call the game function `getPOILoc(float pos[3], int id)` to find the location of each visible POI. The POI ID is used in the function for taking pictures. See section 1.4.1.

**Table 7: Number of Points of Interest**

|                                    |    |
|------------------------------------|----|
| Total number of POIs               | 12 |
| Number of POIs Visible at one time | 3  |

**Table 8: Asteroid and SPHERES Measurements**

|                     |      |
|---------------------|------|
| Asteroid radius (m) | 0.2  |
| SPHERE radius (m)   | 0.11 |

## 1.4.1 Picture Taking

Pictures are taken by calling the function `void takePic(int poiID)`

There are two possible orbits from which the satellite can take pictures: a low orbit in the Inner Zone and a high orbit in the Outer Zone (dimensions listed above in Table 2). In order for a picture to be valid there are three qualifications:

- The satellite must be in the Inner or Outer Zone
- The angle between the line connecting the centre of the SPHERE and the POI and the line connecting the POI and the centre of the asteroid must be smaller than the Max Angle for the zone the satellite is in as described in Table 9.
- The satellite must be facing the POI (with a tolerance of 0.5 radians).

**Table 9: Max Angle of Zones (in radians)**

|            |     |
|------------|-----|
| Inner Zone | 0.8 |
| Outer Zone | 0.4 |

For the satellite to be in the correct orbit, the centre point of the satellite must be within the bounds of the respective Zone (Figure 2). Scoring will be determined by which orbit the satellite is in; *more points will be awarded for pictures taken in high orbit*. The points values awarded for pictures taken in the two orbits (inner zone and outer zone) are provided in Table 10. The POI must be within the field of view for the pictures to score points. The field of view is different for the two orbits (Figure 2).

**Table 10: Picture Values in Inner and Outer Zones**

|            |   |
|------------|---|
| Inner Zone | 2 |
| Outer Zone | 3 |



**Figure 2: Picture Taking Zone**

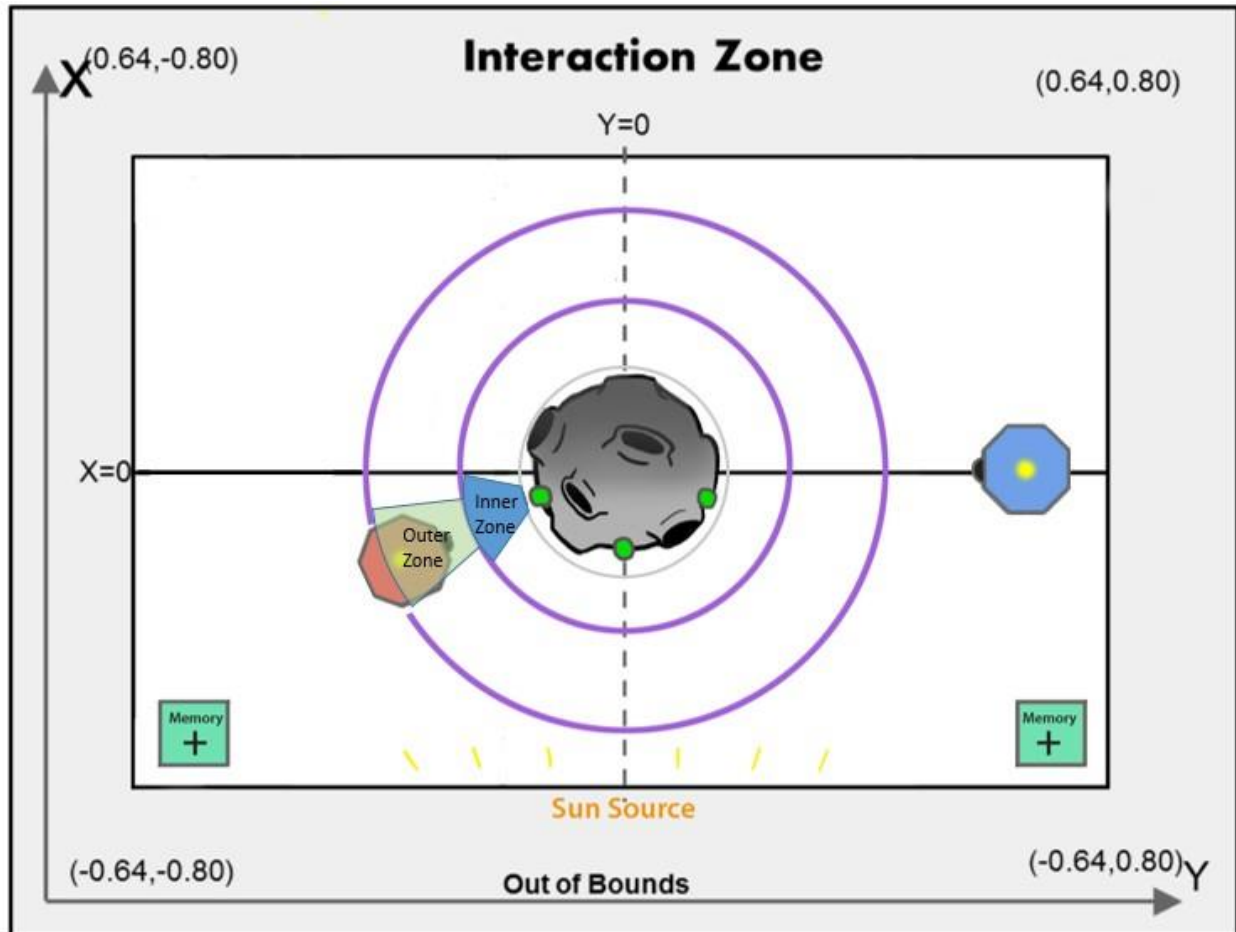


Diagram not to scale

The satellite can only store 2 photos at a time unless it has obtained an upgrade from a memory pack (see 1.4.2). The camera will be disabled once the photo storage is full until the pictures are uploaded.

- The function `int getMemorySize()` returns the current limit of picture storage.
- The function `int getMemoryFilled()` returns the number of valid pictures (taken the right distance and angle from the poi) currently saved in camera.

The camera will be disabled for 3 seconds each time after the `void takePic(int poiD)` function is called.

Pictures of a single point of interest cannot be taken from the same orbit within each 60s window.

There are six possible pictures to take in each 60s window, a picture of each of the points can be taken from each orbit once. These limitations are reset after each 60s window.

In order to take a picture, the point of interest must be within your field of view. This means that your opponent cannot be blocking your field of view (Figure 3).

**Figure 3: Blocking Opponent's Field of View**

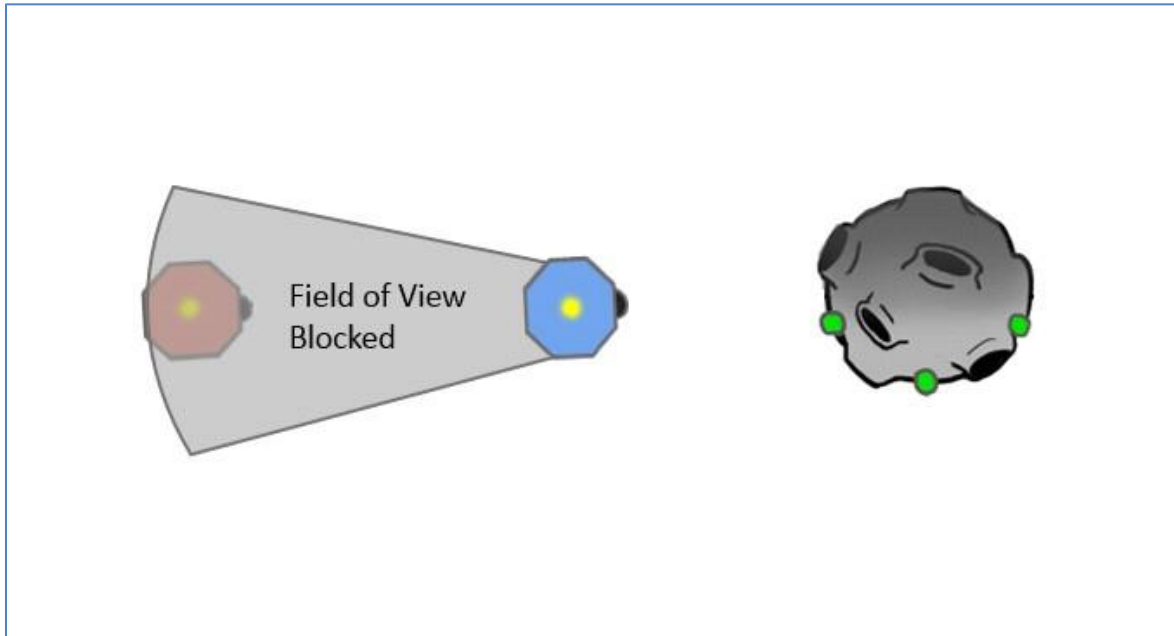


Diagram not to scale

To take pictures:

- Camera must be pointing at POI
- Game function `bool alignLine(int poiD)` Returns true if the SPHERE is facing the POI whose ID is given
- POI must be in field of view (an uninterrupted line can be made from camera to POI)
- Game function `void takePic(int poiD)` must be called

Hint: Consider using Spherical Coordinates to locate the POI (see tutorials for an explanation of how to use Spherical Coordinates).

## 1.4.2 Memory Upgrade Packs

Memory Upgrade packs allow players to store more pictures on their satellite. Memory Upgrade packs are located on the corners of the sunny side of the interaction zone. Memory packs will remain in play throughout the game play. Once a memory pack has been taken by a satellite, it is then in the possession of that satellite and may not be picked up by the other satellite. A satellite can hold up to 2 memory packs. Each pack gives the satellite enough memory to hold 1 more photo. Calling the function `bool hasMemoryPack(int plaeryId, int packID)` returns 'true' if specified player has specified memory pack.

**Table 11: Memory Upgrade Pack Locations**

| Memory Pack 1 (ID = 0) |      |
|------------------------|------|
| X [m]                  | -0.5 |
| Y [m]                  | -0.6 |
| Z[m]                   | 0    |
| Memory Pack 2(ID = 1)  |      |
| X [m]                  | -0.5 |
| Y [m]                  | 0.6  |
| Z[m]                   | 0    |

To collect Memory Upgrade packs see section 1.5 Item Collection.

### 1.4.3 Solar Flares

Solar Flares pose a danger to satellites. Solar Flares occur randomly every 70s starting 30 seconds after the start of the game. Each Solar Flare lasts 3 seconds. All satellites exposed to the solar radiation for any period of time lose the photos stored in their memory.

Satellites exposed to the solar radiation will lose 1 point every second they are exposed to the solar radiation, unless the satellites are powered off prior to exposure with the solar radiation.

While the satellite is *powered off*, players:

- Reduce damage (lose half the points of the satellites powered on)
- Lose any pictures currently stored on the satellite
- Must wait 5 seconds for their instruments to turn on and warm up
- Cannot take pictures
- Cannot stop their satellites from drifting (collisions may occur)

To power off:

- Game functions `void turnoff()` must be called

To power back on:

- Game function `void turnOn()` must be called.

The other way to prevent damage is to seek shelter in the shadow zone. Players are protected from solar radiation in the shadow zone, preserving pictures and points. To be deemed safe, the centre point of the satellite must be in the shadow zone.

Players will have a 30 second warning before a solar flare arrives. This warning can be received by calling the function: `int getNextFlare()`. During the 30 second period prior to the next solar flare `int getNextFlare()` returns the number of seconds until the next solar flare. Calling this function any time outside this 30 second window will return -1. Calling this function during the solar flare will also return -1.

**Figure 4: Solar Flare**

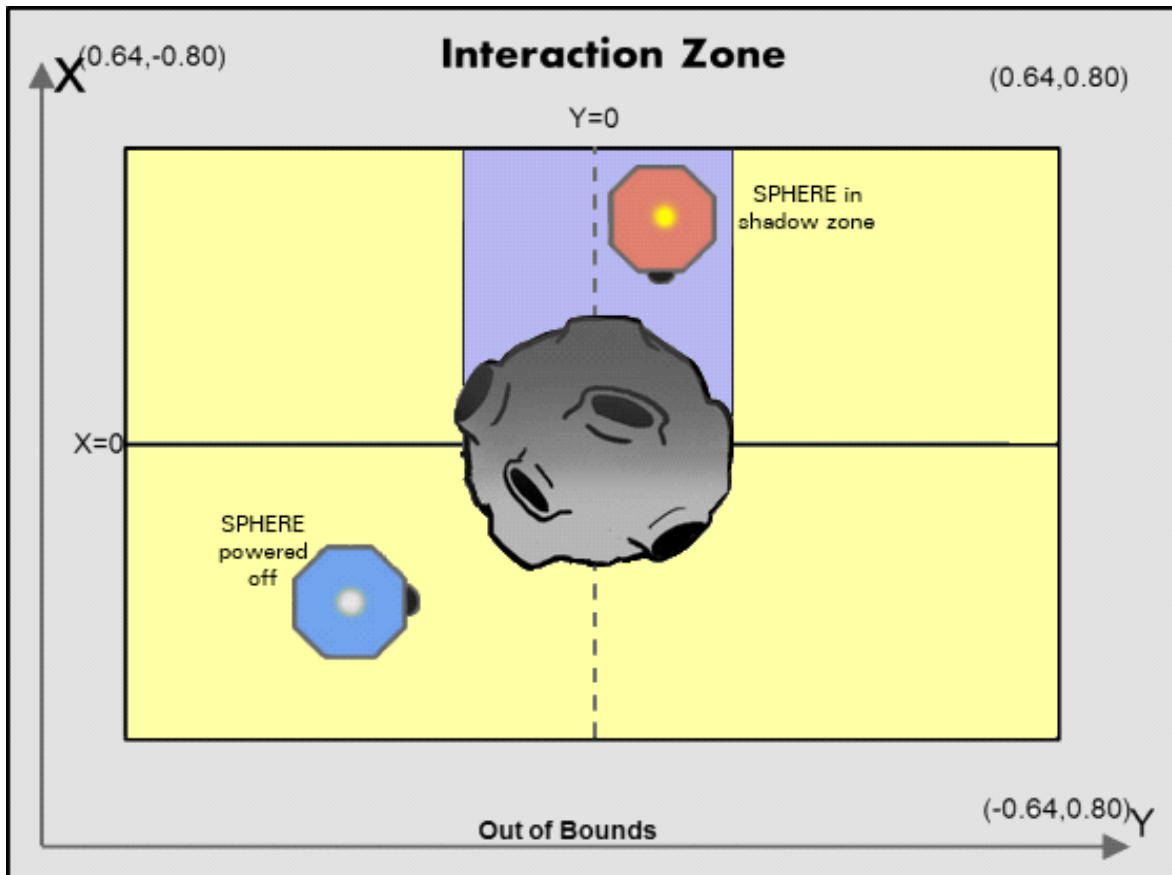


Diagram not to scale

## 1.4.4 Upload

Once the satellite's memory is full, the pictures must be uploaded. Pictures must be uploaded to score full points for taking the picture. Without uploading, only 0.1 points will be awarded for successfully taking a photo and .01 points will be awarded for attempting to take a photo.

During the upload process the camera will be disabled for 3 seconds.

There is a Discover Bonus for the first satellite to *upload* a picture of a specific point of interest within a cycle. The Discover Bonus is +0.5. The Discover Bonus resets with each 60s POI rotation cycle.

To upload photos:

- The centre of the SPHERE must be either outside the Danger Zone and both Picture Taking zones, or in the Shadow Zone.
- The game function `void uploadPic()` must be called

## 1.4.5 Collisions

While it is not possible to collide with the other satellite, collision with the asteroid is possible. If the centre point of your satellite enters the Danger Zone, it is considered a collision with the asteroid.

**Figure 5: Asteroid Collision**

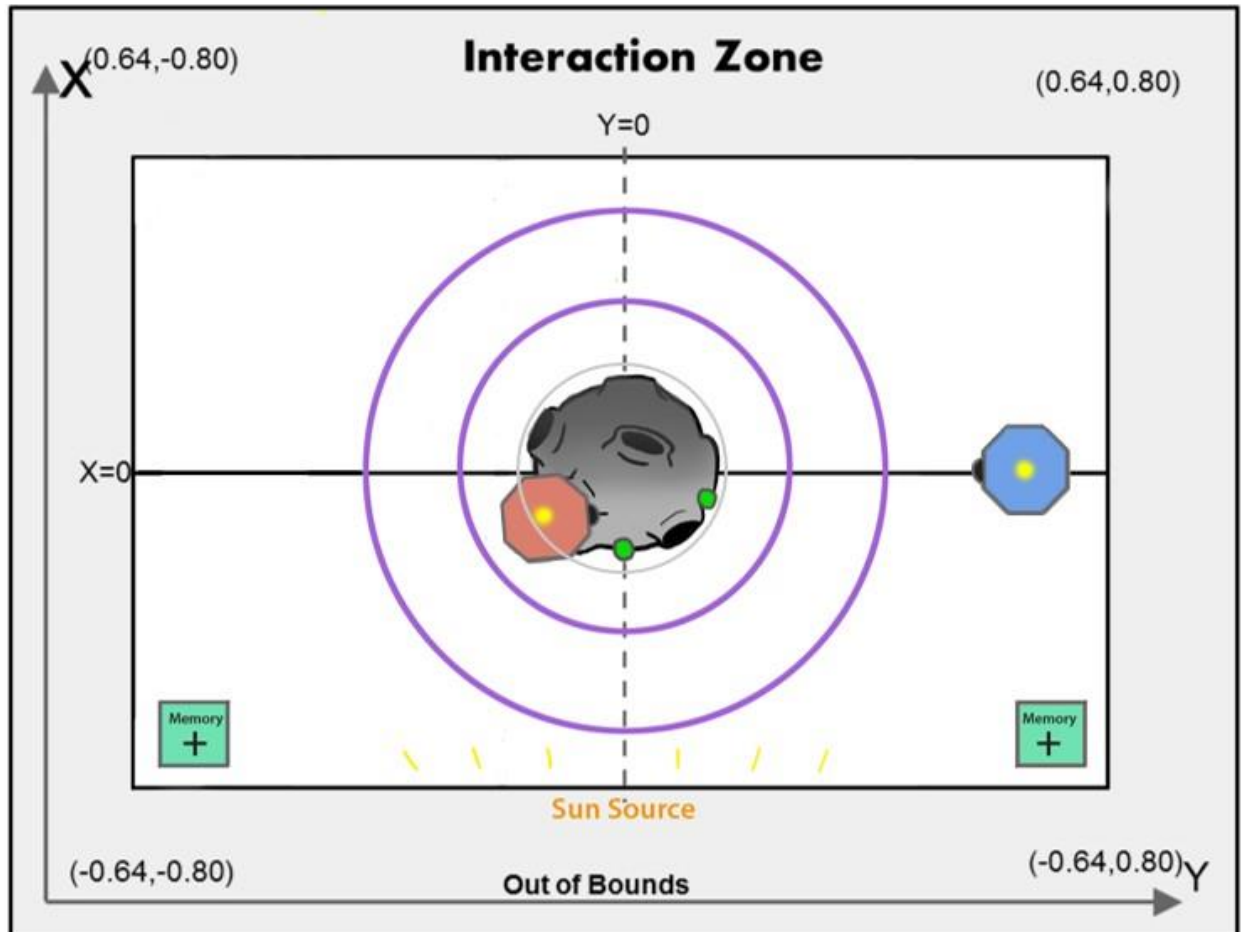


Diagram not to scale

If the satellite crashes into the asteroid:

- The satellite will come to a full stop
- The satellite can only move away from the asteroid
- There will be a fuel penalty equal to 2 thruster seconds of fuel per second in the asteroid
- Points will be lost equal to the number of POI affected in crash site

The crash site is all points on the asteroid within a satellite diameter (.22m) of the satellite's centre point.

**Figure 6: Asteroid Collision Close-up**

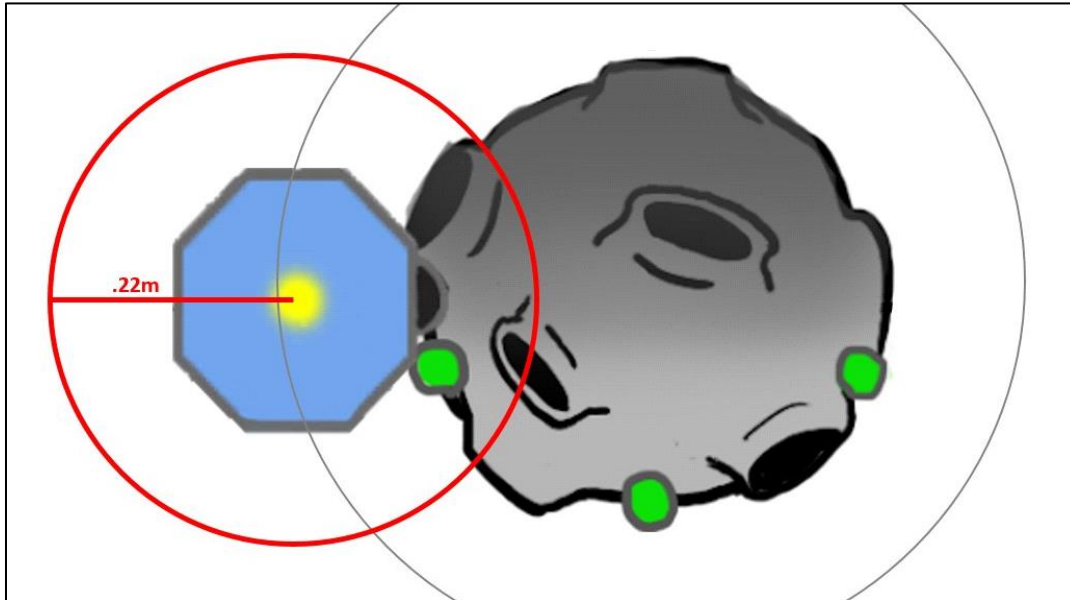


Diagram not to scale

In the figure above, the portion of red circle that intersects with the asteroid represents the crash site. All points of interest within this crash site will be counted against the team whose satellite crashes into the asteroid.

## 1.4.6 End of Game

The game ends when time runs out.

## 1.5 Item Collection

To increase the memory capacity of the satellites, teams have the opportunity to collect memory upgrade packs found in the 2 corners of the interaction zone closest to the sun.

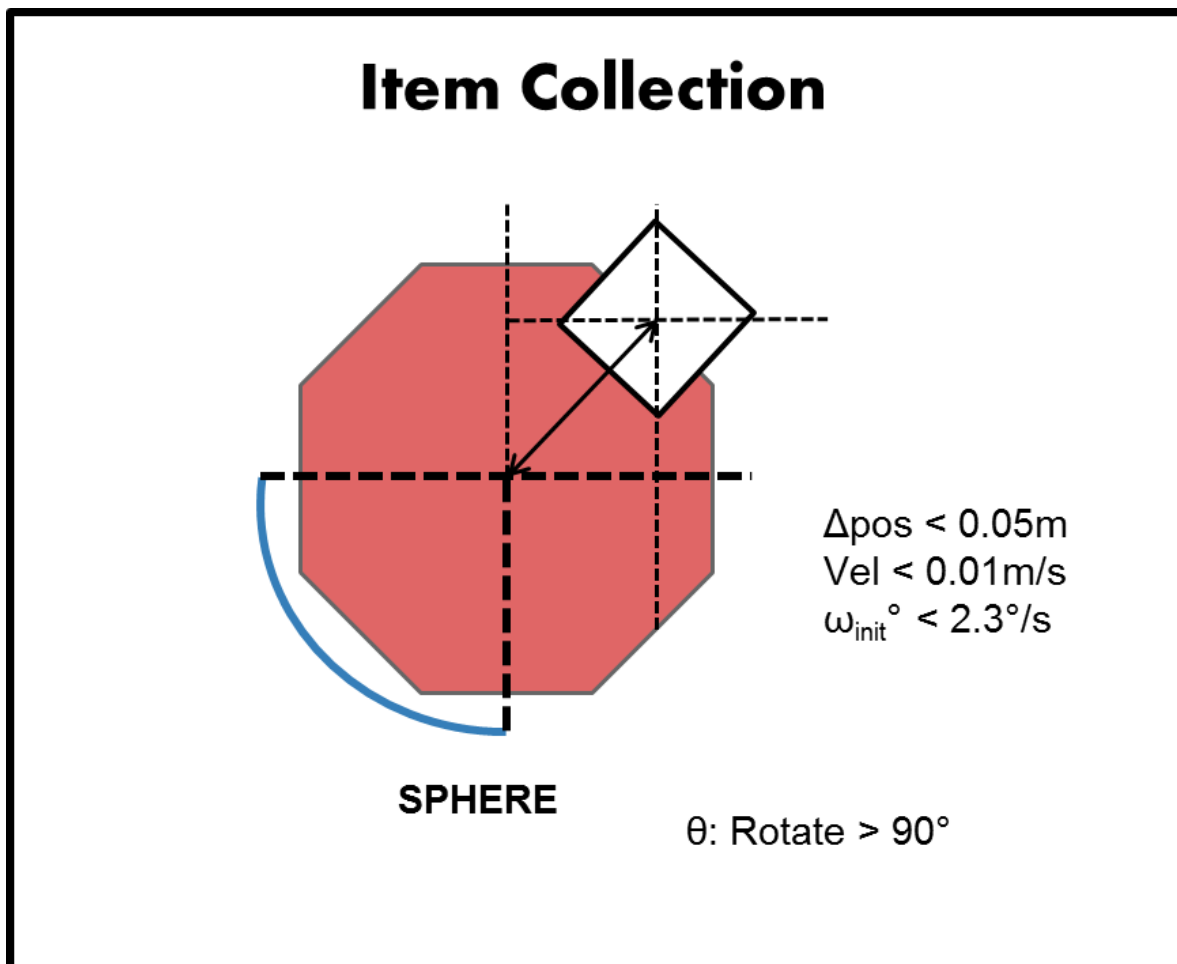
**Table 12: Memory Upgrade Pack Locations (repeated)**

| Memory Pack 1 |      |
|---------------|------|
| X [m]         | -0.5 |
| Y [m]         | -0.6 |
| Z[m]          | 0.0  |
| Memory Pack 2 |      |
| X [m]         | -0.5 |
| Y [m]         | 0.6  |
| Z[m]          | 0.0  |

In order to pick up an item, you need to perform a spinning maneuver (see Figure 7). The steps to collect the Memory Upgrade packs are:

- Position the satellite within 0.05m of the item's centre.
  - The satellite's velocity must be less than 0.01m/s
  - The satellite's angular velocity must start at less than  $2.3^\circ/\text{s}$
- Rotate the satellite  $>90^\circ$  along about the Z axis. Do not attempt to rotate faster than  $80^\circ/\text{s}$ .

**Figure 7: Maneuver to Collect Item**



## 1.6 Out of Bounds

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

If you exit out of bounds, the SPHERES controller will override your commands and force the 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 satellite begins with a score of 9 points. Your final score is based on the pictures you take and upload minus the damage obtained from solar flares and collisions. Pictures are worth different point values depending on which orbit they were taken from. The score will be totaled from the number and location of the photos taken. The seconds exposed to the solar radiation will subtract points from your score.

The scoring calculation is as follows:

**Table 13: Point Values**

| Component   | Points          |
|---|-----------------|
| Number of points at time = 0  | 9               |
| Pictures attempted (both valid and invalid pictures)                            | 0.01            |
| Non uploaded valid photos   | 0.1             |
| Uploaded: Taken in Inner Zone   | 2               |
| Uploaded: Taken in Outer Zone   | 3               |
| Collision with Asteroid: Points deducted per second per POI contacted           | -1              |
| Discover bonus  | +0.5            |
| Solar Flare: Points deducted per second: SPHERE un-powered, outside shadow zone | -0.5            |
| Solar Flare: Points deducted per second: SPHERE inside shadow zone              | 0               |
| To avoid ties   | See notes below |

Note: At the end of the match, to avoid ties, points will be subtracted from the score based on the following calculation: (satellite distance from the centre of asteroid) x (0.00001).

## 3. Tournament

The following table lists key dates for the Australian Preliminary Competition

|                    |   |
|--------------------|---|
| Friday 6 May       | Launch of competition                   |
| Thursday 12 May    | Game manual and mentor details provided |
| Monday 16 May      | Competition begins                      |
| COB Monday 27 June | Submission deadline                     |
| Thursday 30 June   | Final rankings announced                |

The leaderboard will be run in the days after the submission deadline. This will be conducted in a round-robin format with all teams playing each other. An average win/loss record will be used to determine final rankings.



## 4. Season Rules

### 4.1 Tournament Rules

All participants in the Australian Preliminary Competition 2016 must abide by these tournament rules:

- The Zero Robotics team (MIT/Top Coder/Aurora/University of Sydney) can use/reproduce/publish any submitted code.
- In the event of a contradiction between the intent of the game and the behaviour 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 behaviour of the game
  - The intent of the game shall override the behaviour 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.

### 4.2 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 un-reported bug may be considered as unethical behaviour
- 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 ZR website
- Code submitted to a competition must be written only by students

## 5. ZR User API

### 5.1 Standard Zero Robotics API Reference

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

[http://static.zerorobotics.mit.edu/docs/tutorials/ZR\\_user\\_API.pdf](http://static.zerorobotics.mit.edu/docs/tutorials/ZR_user_API.pdf)

Note: Math functions in this table do not need the 'api' prefix.

### 5.2 CoronaSPHERES API Reference

The functions in this section are called as members of the game object: `game.functionName(argument)s`:

| Name  | Description   |
|---|---|
| <code>int getNextFlare()</code>                           | During the 30 second period prior to the next solar flare this function returns the number of seconds until NextFlare. Warnings are only issued 30 seconds in advance. Calling this function any time outside this window will return -1. Calling this function during the flare will also return -1. |
| <code>int getMemoryFilled()</code>                        | Returns number of valid pictures (taken from the right distance and angle from the poi) currently saved in camera.  |
| <code>int getMemorySize()</code>                          | Returns current limit of picture storage  |
| <code>void takePic(int poiID)</code>                      | Takes picture and stores them in the satellite memory. Camera disabled if picture fills last memory slot. Camera is disabled for 3 seconds each time takePic is called.   |
| <code>void uploadPic()</code>                             | Uploads pictures from satellite, disables camera for 3 seconds  |
| <code>int numActivePOIs()</code>                          | Always returns 2 in 2D  |
| <code>void getPOILoc(float pos[3], int id)</code>         | Returns location of each visible POI. Visible POIs are assigned ID # 0, 1 or 2  |
| <code>float getScore()</code>                             | Returns player's score  |
| <code>float getOtherScore()</code>                        | Returns opponent's score  |
| <code>void turnOff()</code>                               | Turns off satellite to protect from solar flare. Satellite will drift.  |
| <code>void turnOn()</code>                                | Begins process to turn satellite on. Takes 5 seconds. (unless game just beginning)  |
| <code>float getFuelRemaining()</code>                     | Returns remaining fuel  |
| <code>bool hasMemoryPack(int playerId, int packID)</code> | Returns true if specified player has specified memory pack  |
| <code>bool alignLine(int poiID)</code>                    | Returns true if the SPHERE is facing the poi whose ID is given. It does not check if poi is in field of view or if sphere is in the correct angle or zone.  |
| <code>void sendMessage(unsigned char inputMsg)</code>     | Sends <b>inputMsg</b> to other satellite  |
| <code>unsigned char receiveMessage()</code>               | Returns the most recent message sent by other satellite   |