THE UNIVERSITY OF SYDNEY

ZERO ROBOTICS
ISS PROGRAMING CHALLENGE

Zero Robotics ISS Programming Challenge

Australian Preliminary Competition 2017

THE UNIVERSITY OF SYDNEY

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

Hello employees of BLU Industries, I have exciting news 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-SPHERETM 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-SPHERETM, 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-SPHERETM would easily get there first. Unfortunately, however, RED have their own MEGA-SPHERETM, 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-SPHERETM 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

THE UNIVERSITY OF SYDNEY

Contents

THE UNIVERSITY OF SYDNEY

1. Game Overview

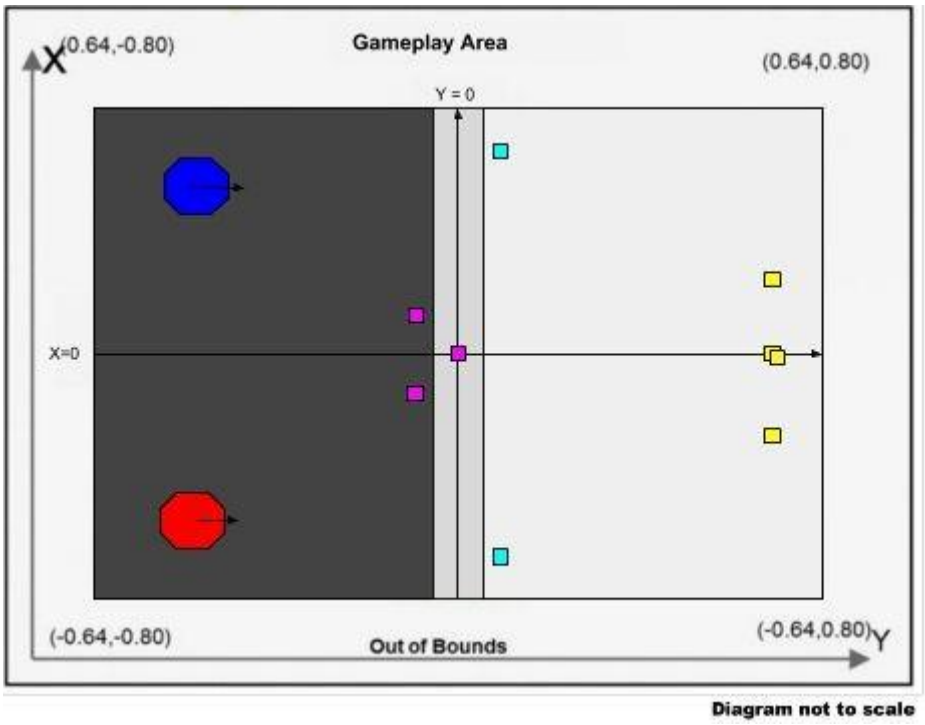

Figure 1: Game Overview

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 upload, and can only be successfully taken when the target is in the Light Zone. The satellites have a storage capability of 2 pictures, which must be uploaded to receive the corresponding points.

The Light, Dark, and Grey 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 Australian Preliminary competition mimics the operational volume available aboard the International Space Station. The arena includes X and Y 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]

Satellites that go out of these limits will lose 2% of their initial fuel per second.

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. The grey is between them. There is more info on the light and dark zones in all phases in 1.4.2. There are also nine items, seven Score+ and Energy Pack types and two Mirror types, information on which is in 1.4.3. A game mechanics summary is given in Table 2.

Table 2: Game Summary

Score Items	3
Mirror Items	Pictures are worth 0
Energy Items	4
Item Locations	Deterministic
Light Zone	Switching
Uploading	No Attitude Required

THE UNIVERSITY OF SYDNEY

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 4 of this document.

1.2.2 Time

Players have 180s gain as many points as possible. 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. Being out of bounds consumes an additional penalty of 2% of initial fuel per second.

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

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.

1.2.5 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.

THE UNIVERSITY OF SYDNEY

1.3 Initial Position

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

Table 3: SPHERES Satellite Deployment Locations

Red	
X [m]	-0.4
Y [m]	-0.6
Z [m]	0.0
Blue	
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 centre of the satellite.

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

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, memory, and their position in light or darkness.

1.4.1 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 manoeuvre the satellite, 0.15 energy is used per 1 second of fuel used. Other activities that cost energy include taking and uploading pictures.

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()`.

THE UNIVERSITY OF SYDNEY

1.4.2 Light and Dark Zones

The Light and Dark Zones, and the Grey Zone in between them, will be the main determinant of how well you can take pictures of the other satellite. Here is a table with their properties:

Table 4: Memory Upgrade Pack Locations

	Can a Picture be Taken of Me?	Does My Energy Recharge?
Light Zone	Yes	Yes
Grey Zone	Yes	No
Dark Zone	No	No

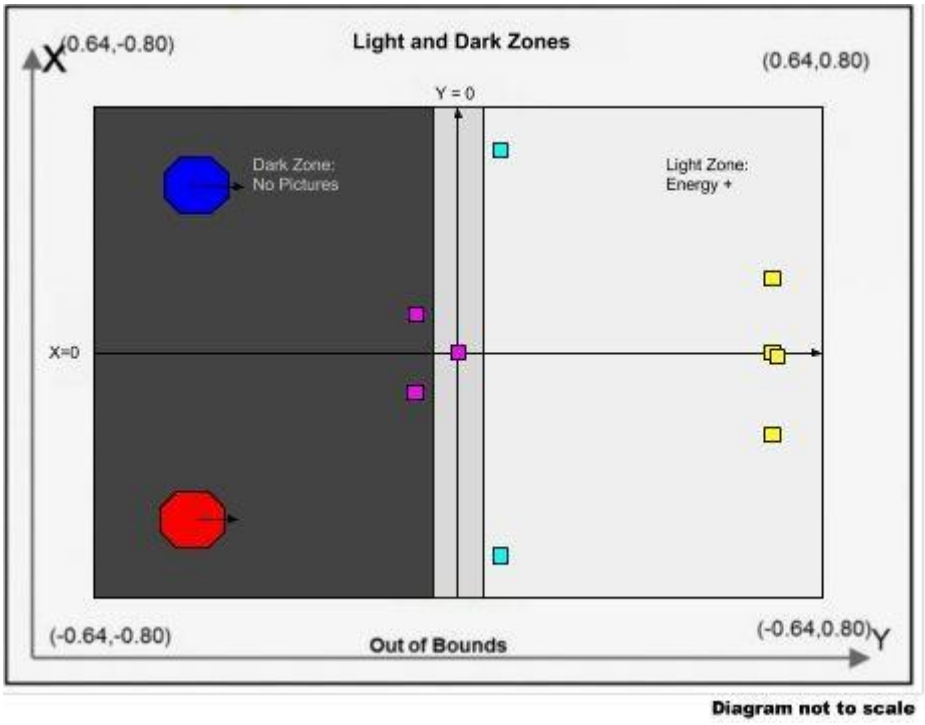

Figure 2: Light and Dark Zones

The negative Y sector starts out in the Dark Zone and the positive section starts out in the Light Zone. In between them, with a total width of 0.2 meters centred at the origin, is the Grey Zone. Between 30 and 60 seconds into the game, the Light Zone and the Dark Zone will switch position. The zones will switch again periodically every 60 seconds until the end of the game.

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

THE UNIVERSITY OF SYDNEY

1.4.3 Items

There are three types of items scattered around the interaction zone: Energy Packs, Score+, 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.

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, and in 3D a negative value whose magnitude is equal to the number of points the picture would have otherwise been worth. To clarify, the mirrors do not prevent pictures from being uploaded, merely from being taken. Some related functions are:
    - `void useMirror()` – Deploys a mirror.
    - `int getNumMirrorsHeld()` – Returns the number of mirrors the user has.

Call the game function `int hasItem(int item_id)` to determine whether the item is held by nobody (-1), you (0), or your opponent (1).

Call the game function `float[3] getItemLoc(int item_id)` to obtain the location of the item. Several other item-related functions are detailed in the API. In order to collect an item, the centre 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.

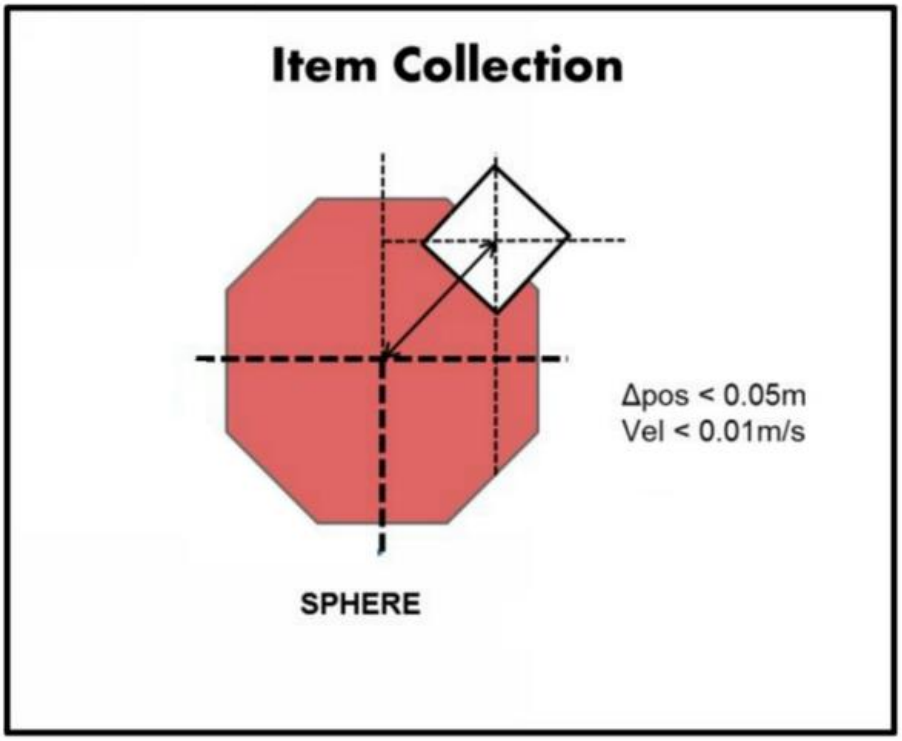

Figure 3: Light and Dark Zones

Table 4: Item Locations

0	Energy Pack	(0.3, 0.0)
1	Energy Pack	(-0.3, 0.0)
2	Energy Pack	(0.0, 0.3)
3	Energy Pack	(0.0, -0.3)
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)

THE UNIVERSITY OF SYDNEY

1.4.4 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 0.25 radians.)
- The opposing satellite must not be in the Dark Zone (that is, it must be in the Grey Zone or the Light Zone).
- The user satellite must have at minimum 1 energy.
- The user satellite must have at least 1 open memory slot. Each satellite has two memory slots total.
- The user satellite's camera must be on.
- The user must not have an **active** mirror item.
- 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, and store it in the first available memory slot. 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 will return 0 if picture-taking was unsuccessful, and the point value of the picture otherwise. Unsuccessful pictures do not use a memory slot. 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:

```
points = 2.0  + 0.1 / (distance - 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 zero.

The function `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.

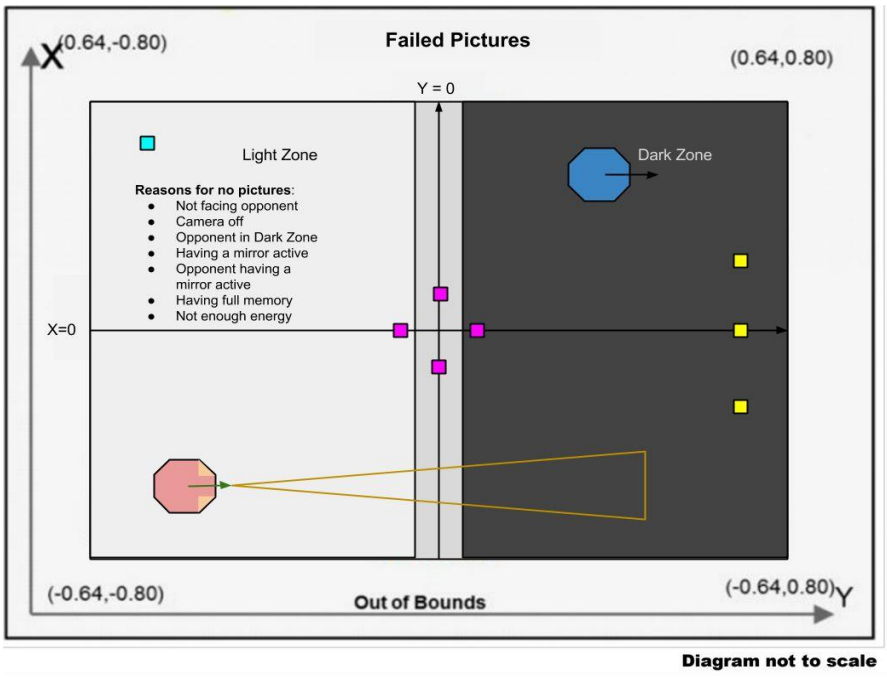

Figure 4: Reasons for Failed a Picture

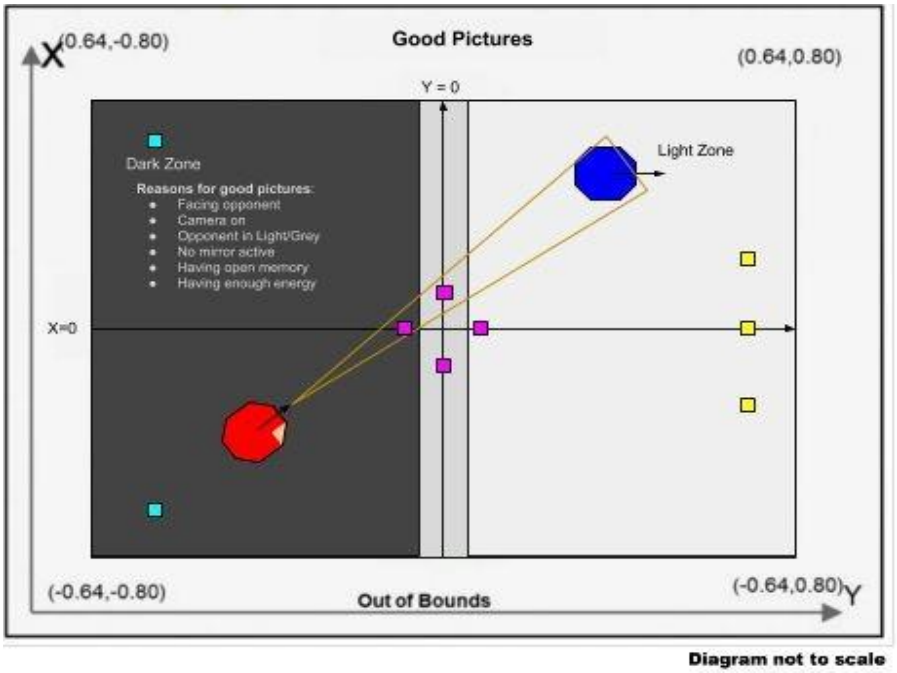

Figure 5: Conditions for a Successful Picture

THE UNIVERSITY OF SYDNEY

1.4.5 Uploading Pictures

In order to receive points for the pictures that have been taken, the sphere needs to upload them to Earth. When uploading there are also several requirements that must be met:

- The satellite must have at least 1.0 energy.
- The satellite must call `float uploadPics()`.

When calling the function, the camera will be disabled for 3 seconds, similarly to taking pictures Afterwards, `uploadPics()` will return the number of points that were uploaded. This process costs 1.0 energy. You do not receive points for pictures taken that are not uploaded.

1.4.6 End of Game

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

1.5 Scoring Summary

Method	Point Value
Attempting to Take a Picture (Valid or Not)	0.01
Taking a Valid Picture	0.1
Uploading Pictures	2.0 + 0.1/(distance – 0.5 + 0.1)
Score Items	1.5 per item

THE UNIVERSITY OF SYDNEY

2. Tournament

The following table lists key dates for the Australian Preliminary Competition

Friday 21 July	Launch of competition
Friday 25 August	First leaderboard run
Friday 1 September	Second leaderboard run
Friday 8 September	Submission deadline/Final leaderboard run
TBA	Final rankings announced

The leaderboard will be run on the dates listed above. Submissions close at 5pm AEST. Teams are not expected to have full working versions of their code ready for the initial leaderboard run. The teams that have begun writing and testing their initial strategy will be rewarded with additional tournament points from the early leaderboard runs. Perhaps the final code produced does not win the final leaderboard run however a consistent effort and accumulation of points could make this team the winner. The best strategy is to **SUBMIT EARLY, SUBMIT OFTEN.**

3. Season Rules

3.1 Tournament Rules

All participants in the Australian Preliminary Competition 2017 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.

THE UNIVERSITY OF SYDNEY

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 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

4. ZR User API

4.1 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

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

4.2 SpySPHERES API Reference

The functions in this section are called as members of the game object: game.functionName(argument/s). Full details can be found here: http://static.zerorobotics.mit.edu/docs/tutorials/refman.pdf

Name	Description
`float getFuelRemaining()`	Returns a float that is the number of seconds of thruster use remaining before fuel runs out.
`void sendMessage(unsigned char inputMsg)`	Sends a value from 0-255 to the other satellite.
`unsigned char receiveMessage()`	Receives a value from 0-255 from the other satellite.
`bool isFacingOther()`	Returns true if your camera is facing the other satellite, false otherwise.
`float takePic()`	Attempts to take a picture in the current position. Returns the amount of points the picture taken is worth.
`float getPicPoints()`	Returns how many points a picture would be worth if taken immediately. *Does not* take a picture.

`int getMemoryFilled()`	Returns how many memory slots are currently in use.
`int getMemorySize()`	Returns total (used and unused) number of memory slots available.
`float uploadPics()`	Attempts to upload pictures taken to Earth. Returns total score over the game so far.
`bool isCameraOn()`	Returns true if the camera is usable, false if not.
`float getEnergy()`	Returns the amount of energy you currently have.
`float getOtherEnergy()`	Returns the amount of energy your opponent currently has.
`bool posInLight(float pos[])`	Returns true if the coordinate *pos[]* is in the light zone, false otherwise.
`bool posInDark(float pos[])`	Returns true if the coordinate *pos[]* is in the dark zone, false otherwise.
`bool posInGrey(float pos[])`	Returns true if the coordinate *pos[]* is in the grey zone, false otherwise.
`int posInArea(float pos[])`	Returns 1 if the coordinate *pos[]* is in the light, -1 if in the dark and 0 otherwise.
`float getLightInterfacePosition()`	Returns the y-coordinate of the plane at the centre of the grey zone.
`float getDarkGreyBoundary()`	Returns the y-coordinate of the plane defining the transition from dark to grey zones.
`float getLightGreyBoundary()`	Returns the y-coordinate of the plane defining the transition from light to grey zones.
`float getLightSwitchTime()`	Returns in seconds the time until the light and dark zones switch sides.
`int getNumItem()`	Returns the number of Items in play, whether they have been picked up yet or not.
`bool useMirror()`	Attempts to use a currently held, unused mirror. Returns true if successful, false otherwise.
`int getMirrorTimeRemaining()`	Returns the amount of time in seconds your current mirror will remain active, zero if no mirror is active.
`int getNumMirrorsHeld()`	Returns the number of mirrors held.
`void getItemLoc(float pos[], int itemID)`	Overwrites the location of the item specified by *itemID* into the array specified by the pointer *pos[]*.

`int hasItem(int itemID)`	Returns 0 if you have picked up the item specified by *itemID*, 1 if the other player has and -1 if no one has.
`int getItemType(int itemID)`	Returns 0 if the item specified by *itemID* is an energy pack, 1 for a score+ pack and 2 for a mirror.
`float getScore()`	Returns the players current score.
`float getOtherScore()`	Returns your opponent's current score.
`int getCurrentTime()`	Returns the current amount of time passed in the game.